JAVA SE 11 PROGRAMMER-1

1Z0-815

PRACTICE TEST
EXAM SIMULATION

UDAYAN KHATTRY

JAVA SE 11 PROGRAMMER-1

1Z0-815

PRACTICE TEST
EXAM SIMULATION

UDAYAN KHATTRY

Oracle Certified Professional: Java SE 11 Programmer I
1Z0-815

Practice Tests

By: Udayan Khattry

DEDICATION

This book is dedicated to my wife Neha, with love.

Imprint: Independently published

Cover design by Juhi Saxena

27-Jul-2019:	v1.00 (1st Release)
22-Aug-2019:	v1.01
08-Sep-2019:	v1.03
10-Oct-2019:	v1.04

Author’s Profile

Udayan Khattry

SCJP, SCWCD & Oracle Database SQL Certified Expert

Author has a master's degree in Computer Applications from Symbiosis International University, Pune, India and have completed following professional certifications:
- **SCJP** 1.6 (Sun Certified Programmer for J2SE 6.0)
- **SCWCD** 1.5 (Sun Certified Web Component Developer)
- **Oracle Database SQL Certified Expert**

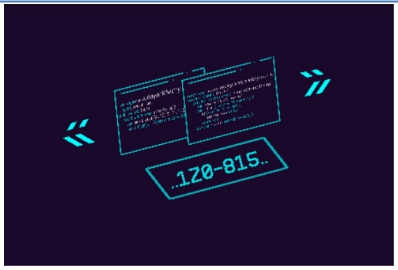
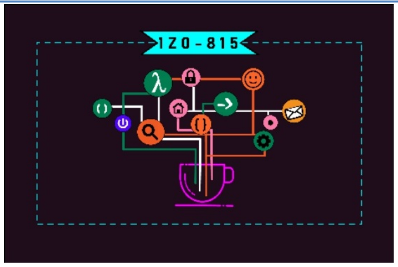
After working as a software developer and consultant for over 9 years for various companies in India, Dubai & Singapore, he decided to follow his lifelong passion of teaching.

In the last 3 years, author has published multiple books and online courses on Java and Java certifications. He currently has 28000+ students from 140+ countries.

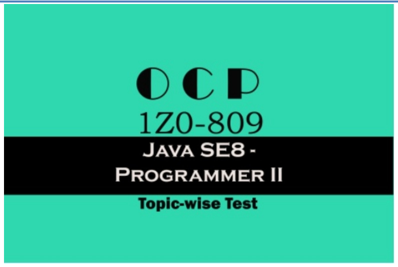
Audience

Anyone preparing for Java SE 11 Programmer I: 1Z0-815 exam or interviews can take this book to assess his/her preparation.

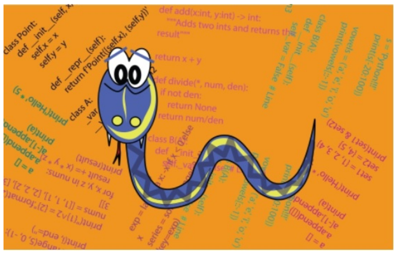
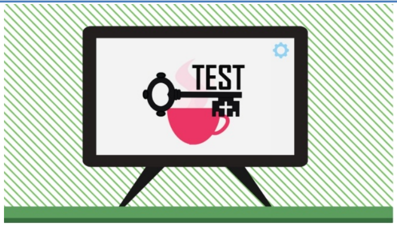
Udemy Courses By Author

	**Java Certification (1Z0-815) Exam Simulation** **480 multiple choice questions with explanation to assess Oracle Certified Professional, Java SE 11 Programmer I exam preparation** **Practice tests** are randomized to give a real examination feel. All topics listed above are divided appropriately in 6 **tests consisting 80 questions each i.e., 480 questions in total** . Questions are designed based on real examination questions in terms of pattern and complexity. **All Exam topics and sub-topics are covered.**
Course Link	https://www.udemy.com/course/java-se-11_1z0-815/?referralCode=F409C96F9DD47698A3AE
	**Java Certification (1Z0-815) Topic-wise Tests** **Multiple choice questions covering all the exam objectives of Java SE 11 Programmer I Exam** **This course covers all the EXAM topics in orderly fashion, which will help students assess their preparation for respective topic.** This course can be used as a learning aid while preparing for 1Z0-815 certification to test your preparation for each topic while preparing for that topic.
Course Link	https://www.udemy.com/course/java-11_1z0-

	815/? referralCode=B0A027B28ACC27976961
	Java Certification : OCA (1Z0-808) Exam Simulation [2019] **280 multiple choice questions with explanation to assess Oracle Certified Associate, Java SE 8 Programmer I preparation** **Practice tests** are randomized to give a real examination feel. All topics listed above are divided appropriately in **4 tests consisting 70 questions each i.e., 280 questions in total** . Questions are designed based on real examination questions in terms of pattern and complexity. **All Exam topics and sub-topics are covered.**
Course Link	https://www.udemy.com/course/java-oca/? referralCode=2337F77572B062EB41D6
OCA 1Z0-808 JAVA SE8 - PROGRAMMER I Topic-wise Test	**Java Certification - OCA (1Z0-808) Topic-wise Tests [2019]** **Multiple choice questions covering all the exam objectives of Oracle Certified Associate, Java SE 8 Programmer I** This course covers **all the EXAM topics in orderly fashion** , which will help students assess their preparation for respective topic. Questions are designed based on real examination questions in terms of pattern and complexity. **All Exam topics and sub-topics are covered.**
	https://www.udemy.com/course/java-ocajp/?

Course Link	referralCode=AAD655BA1CE88EEE7DDC
	Java Certification : OCP (1Z0-809) Exam Simulation [2019] **540 multiple choice questions with explanation to assess Oracle Certified Professional, Java SE 8 Programmer II prep** **Practice tests** are randomized to give a real examination feel. All topics listed above are divided appropriately in **6 tests consisting 90 questions each i.e., 540 questions in total.** Questions are designed based on real examination questions in terms of pattern and complexity. **All Exam topics and sub-topics are covered.**
Course Link	https://www.udemy.com/course/java-ocp/?referralCode=13982FCB1E0CAA5B94FB
 OCP 1Z0-809 JAVA SE8 - PROGRAMMER II Topic-wise Test	**Java Certification - OCP (1Z0-809) Topic-wise Tests [2019]** **Multiple choice questions covering all the exam objectives of Oracle Certified Professional, Java SE 8 Programmer II** This course covers **all the EXAM topics in orderly fashion** , which will help students assess their preparation for respective topic. Questions are designed based on real examination questions in terms of pattern and complexity. **All Exam topics and sub-topics are covered.**
Course Link	https://www.udemy.com/course/java-ocpjp/?referralCode=22BEEDC2D666C97BA703

	Test Java Functional Programming (Lambda & Stream) skills **180+ questions on Inner classes, Lambda expressions, Method References, Functional Interfaces & Stream API** **Practice tests** in this course will not only help you to assess your current knowledge of these topics but will also help you to revise the topics quickly. Questions are designed to *challenge your understanding of the topics* . Detailed explanations for all the questions are also provided for your reference.
Course Link	https://www.udemy.com/course/test-functional-programming/?referralCode=6A6A598EDD16CF40AA8E
	Java For Beginners - 1st step towards becoming a Java Guru! **Become a Java Expert with 22.5 hours of video content, 70+ coding challenges and 100+ Quiz questions** This course is for anyone who wants to learn Java from scratch, polish java skills, face java interviews and prepare for java certifications. Anyone can take this course and go from 0 developments skills to being expert in OOPs and core Java.
Course Link	https://www.udemy.com/course/corejava/?referralCode=831CD22E895230578AF2
	Python Quiz - Test your Python knowledge in 1 Day! **11th hour preparation for Python**

	**interviews, exams and tests with multiple choice questions** Basic to intermediate Python concepts are covered to assess your knowledge and skills. Questions are arranged in an orderly manner to provide ease of understanding.
Course Link	https://www.udemy.com/course/python-test/?referralCode=F070ABFC34905F36FF71
	**Test your Core Java skills** **139 multiple choice questions to test your Core Java skills** This course is for anyone who wants to test or brush up their **core Java skills or face java interviews** . Questions are organized based on core java topics rather than in an exam setting for maximum understanding.
Course Link	https://www.udemy.com/course/testcorejava/?referralCode=B8C939C8E3AEDA4EC4FD

NOTE: Please send an email to udayan.khattry@outlook.com to request for **MAXIMUM Discount coupon code ($9.99 or ₹360.00)** for above courses.

Introduction

Java SE 11 Programmer I- Exam Information:

- **Exam Code: 1Z0-815**
- **Duration: 180 minutes**
- **Questions #: 80 (Multiple Choice / Multiple Select)**
- **Passing score: 63%**

Exam Curriculum:

- **Understanding Java Technology and environment**
- **Creating a Simple Java Program**
- **Working With Java Primitive Data Types and String APIs**
- **Using Operators and Decision Constructs**
- **Working with Java Arrays**
- **Describing and Using Objects and Classes**
- **Creating and Using Methods**
- **Applying Encapsulation**
- **Reusing Implementations Through Inheritance**
- **Programming Abstractly Through**

Interfaces

- ## Handling Exceptions
- ## Understanding Modules

All topics listed above are divided appropriately in **6 Practice Tests consisting 80 questions each i.e., 480 questions** in total.

After each Practice Test, **correct answers** are provided with **explanation** for reference and understanding. **Relevant hints and how to approach a question in real examination setting is also provided in explanation.**

Completing all the tests successfully will boost your confidence to attempt 1Z0-815 examination .

More information on detailed curriculum and assumptions, to be followed, for examination is available on oracle certification page. https://education.oracle.com/java-se-11-programmer-i/pexam_1Z0-815

Disclaimer : These questions are not real examination questions / dumps. These questions are created to evaluate your preparation for certification exam.

For any questions / problems in above links send an email to: udayan.khattry@outlook.com .

Assumptions:

Questions mentioning line numbers: Because of wrapping one statement can be shown in multiple lines. If a question mentions line number, then consider the starting of line from the java statement / block perspective. In below code fragment, Line 14 represents ‘ catch(IllegalArgumentException | RuntimeException | Exception e) { ’ and not ‘ | Exception e) { ’.

There is no confusion for Line 15 , it represents ‘ System.out.println(e.getMessage()); ’ and Line 16 represents just the closing bracket, ‘ } ’

```
public static void main(String [] args) {
    try {
        convert("" );
    }
    catch (IllegalArgumentException | RuntimeException
        | Exception e) { //Line 14
         System.out .println(e.getMessage()); //Line 15
     } //Line 16
     catch (Exception e) {
        e.printStackTrace();
    }
}
```

1 Practice Test-1

1.1 80 Questions covering all topics.

1.1.1 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;
import java.io.IOException;

abstract class Base {
   public abstract void print() throws IOException;
}

class Derived extends Base {
    @Override
    public void print() throws IOException {
        throw new FileNotFoundException();
    }
}

public class Test {
    public static void main(String[] args) {
        Base b = new Derived();
        try {
            b.print();
        } catch (FileNotFoundException e) {
            System. out .print( "AWE" );
        } finally {
            System. out .print( "SOME" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. AWESOME

B. SOME

C. AWE

D. Compilation error

E. Program ends abruptly

1.1.2 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<String> list;
       list = new ArrayList<>(); //Line n1
        list.add( "A" );
       list.add( "E" );
       list.add( "I" );
       list.add( "O" );
       list.add( "U" );
       list.addAll(list.subList( 0 , 4 )); //Line n2
        System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. An exception is thrown at runtime by Line n2

D. [A, E, I, O, U]

E. [A, E, I, O, U, A, E, I, O, U]

F. [A, E, I, O, U, A, E, I, O]

1.1.3 Consider below codes of 3 java files:

```
//Buyable.java
package com.udayankhattry.ocp1;

public interface Buyable {
   int salePercentage = 85 ;
```

```
    public static String salePercentage() {
        return salePercentage + "%" ;
    }
}

//Book.java
package com.udayankhattry.ocp1;

public class Book implements Buyable {}

//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        Buyable [] arr = new Buyable[ 2 ];
        for (Buyable b : arr) {
            System. out .println(b.salePercentage); //Line n1
             System. out .println(b.salePercentage()); //Line n2
        }

          Book [] books = new Book[ 2 ];
        for (Book b : books) {
            System. out .println(b.salePercentage); //Line n3
             System. out .println(b.salePercentage()); //Line n4
        }
    }
}
```

Which of the following statements are correct?
Select ALL that apply.

A. There is a compilation error in Buyable.java file

B. There is a compilation error in Book.java file

C. There is a compilation error at Line n1

D. There is a compilation error at Line n2

E. There is a compilation error at Line n3

F. There is a compilation error at Line n4

1.1.4 What is the purpose of below lambda expression?

```
(x, y) -> x + y;
```

A. It accepts two int arguments, adds them and returns the int value

B. It accepts two String arguments, concatenates them and returns the String instance

C. It accepts a String and an int arguments, concatenates them and returns the String instance

D. Not possible to define the purpose

1.1.5 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String javaworld = "JavaWorld" ;
        String java = "Java" ;
        String world = "World" ;
        java += world;
        System. out .println(java == javaworld);
    }
}
```

What will be the result of compiling and executing Test class?

A. JavaWorld

B. Java

C. World

D. true

E. false

1.1.6 Consider incomplete code of M.java file:

```
class M {
}
```

```
________ class N {
}
```

Following options are available to fill the above blanks:

1. public
2. private
3. protected
4. final
5. abstract

How many above options can be used to fill above blank (separately and not together) such that there is no compilation error?

A. Only one option
B. Only two options
C. Only three options
D. Only four options
E. All five options

1.1.7 Consider below code snippet:

```
import java.util.*;

class Father {}

class Son extends Father {}

class GrandSon extends Son {}

abstract class Super {
    abstract List<Father> get();
}

class Sub extends Super {
    /*INSERT*/
}
```

And the definitions of get() method:

1. List<Father> get() { **return null** ;}
2. ArrayList<Father> get() { **return null** ;}
3. List<Son> get() { **return null** ;}
4. ArrayList<Son> get() { **return null** ;}
5. List<GrandSon> get() { **return null** ;}
6. ArrayList<GrandSon> get() { **return null** ;}
7. List<Object> get() { **return null** ;}
8. ArrayList<Object> get() { **return null** ;}

How many definitions of get() method can replace /*INSERT*/ such that there is no compilation error?

A. One definition

B. Two definitions

C. Three definitions

D. Four definitions

E. Five definitions

F. Six definitions

G. Seven definitions

H. Eight definitions

1.1.8 Fill in the blanks for the definition of java.lang.Error class:

```
public class java.lang.Error extends _____________ {...}
```

A. RuntimeException

B. Exception

C. Throwable

1.1.9 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
import java.util.List;
import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
      List<String> words = List. of ( "A" , "an" , "the" ,
                          "when" , "what" , "Where" , "whether" );

        processStringArray (words, /*INSERT*/ );
    }

    private static void processStringArray(
          List<String> list, Predicate<String> predicate) {
        for (String str : list) {
            if (predicate.test(str)) {
               System. out .println(str);
            }
        }
    }
}
```

Which of the following options can replace /*INSERT*/ such that on executing Test class all the list elements are displayed in the output?
Select ALL that apply.

A. `p -> true`
B. `p -> !!!! true`
C. `p -> !! false`
D. `p -> p.length() >= 1`
E. `p -> p.length() < 7`
F. `( var p) -> p.length() < 100`
G. `var p -> p.length() > 0`

1.1.10 Given code of Test.java file:

```
//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int x = 1 ;
        while ( checkAndIncrement (x)) {
            System. out .println(x);
        }
    }

    private static boolean checkAndIncrement( int x) {
        if (x < 5 ) {
            x++;
             return true ;
        } else {
             return false ;
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	2 3 4 5	B.	1 2 3 4
C.	1 2 3 4 5	D.	Infinite loop

1.1.11 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class M {
    public void main(String[] args) { //Line n1
        System. out .println( "M" );
    }
}
```

```
class N extends M {
    public static void main(String[] args) { //Line n2
        new M().main(args); //Line n3
    }
}

public class Test {
    public static void main(String[] args) {
        N. main (args); //Line n4
    }
}
```

Which of the following statements is true for above code?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Line n4 causes compilation error

E. It executes successfully and prints M on to the console

1.1.12 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Document {
    default String getType() {
        return "TEXT" ;
    }
}

interface WordDocument extends Document {
    String getType();
}

class Word implements WordDocument {}

public class Test {
    public static void main(String[] args) {
        Document doc = new Word(); //Line n1
        System. out .println(doc.getType()); //Line n2
    }
}
```

Which of the following statements is correct?

A. Interface Document causes compilation error

B. Interface WordDocument causes compilation error

C. Class Word causes compilation error

D. Test class compiles successfully and on execution prints TEXT on to the console

1.1.13 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        System. out .println( 53 / 2.0 );
        System. out .println( 53 % 2.0 );
    }
}
```

What will be the result of compiling and executing Test class?

A.	26 1	B.	26.5 1.0
C.	26.0 1.0	D.	26.5 0.0

1.1.14 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Character> list = new ArrayList<>();
        list.add( 0 , 'E' );
        list.add( 'X' );
        list.add( 1 , 'P' );
        list.add( 3 , 'O' );

          if (list.contains( 'O' )) {
           list.remove( 'O' );
        }

          for ( char ch : list) {
           System. out .print(ch);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Runtime error

C. EPX

D. EXP

E. EPXO

F. EXPO

1.1.15 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Operation {
    int operate( int x, int y);
}

public class Test {
     public static void main(String[] args) {
         int x = 10 ;
         int y = 20 ;
        Operation o1 = (x, y) -> x * y;
        System. out .println(o1.operate( 5 , 10 ));
    }
}
```

What will be the result of compiling and executing Test class?

A. 50
B. 200
C. Compilation error
D. Exception is thrown at runtime

1.1.16 Consider below codes of 2 java files:

```
//Flyable.java
package com.udayankhattry.ocp1;

public interface Flyable {
    static int horizontalDegree() { //Line n1
        return 20 ;
    }

    default void fly() {
        System. out .println( "Flying at "
            + horizontalDegree () + " degrees." ); //Line n2
    }

    void land();
}


//Aeroplane.java
package com.udayankhattry.ocp1;

public class Aeroplane implements Flyable {
    public void land() {
        System. out .println( "Landing at "
                + -Flyable.horizontalDegree()
                        + " degrees." ); //Line n3
    }

    public static void main(String[] args) {
        new Aeroplane().fly();
        new Aeroplane().land();
    }
}
```

What will be the result of compiling and executing Aeroplane class?

A. Compilation error at Line n1

B. Compilation error at Line n2

C. Compilation error at Line n3

D. Given code compiles successfully and on execution prints below in the output:

Flying at 20 degrees.

Landing at -20 degrees.

1.1.17 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String[] arr = { "A" , "B" , "C" , "D" }; // Line n1
        arr[ 0 ] = arr[ 1 ]; // Line n2
        arr[ 1 ] = "E" ; // Line n3
        for (String s : arr) {
           System. out .print(s + " " );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. B E C D

D. E E C D

E. A E C D

F. B B C D

G. A B C D

1.1.18 Does JDK 11 release offer separate JRE?

A. Yes

B. No

1.1.19 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder( "Dream BIG" );
        String s1 = sb.toString();
        String s2 = sb.toString();

            System. out .println(s1 == s2);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. true

C. false

D. An exception is thrown at runtime

1.1.20 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   +---x
|   |   |   module-info.java
|   |   |
|   |   \---com
|   |       \---circus
|   |               Animal.java
|   |               Lion.java
|   |
|   \---y
|       |   module-info.java
|       |
|       \---com
|           \---circus
|               \---tricks
|                       Tricks.java
|
\---bin
```

Below are the file details:-

C:\src\x\com\circus\Animal.java:

```
package com.circus;

public abstract class Animal {
   public abstract void eat();
}
```

C:\src\x\com\circus\Lion.java:

```
package com.circus;

public class Lion extends Animal {
    @Override
    public void eat() {
       System. out .println( "Lion eats meat" );
    }
}
```

C:\src\x\module-info.java:

```
module x {
    exports com.circus;
}
```

C:\src\y\com\circus\tricks\Tricks.java:

```
package com.circus.tricks;

import com.circus.*;

public class Tricks {
    public static void main(String... args) {
        doTricks ( new Lion());
    }

    private static void doTricks(Animal animal) {
        animal.eat();
    }
}
```

C:\src\y\module-info.java:

```
module y {
    requires x;
}
```

Which of following commands if executed from C:\ will compile both the modules x and y?

A. javac -d bin --module-source-path src --module x

B. javac -d bin --module-source-path src --module y

C. javac -d bin --module-source-path src --module *

D. javac -d bin --module-source-path src --module x;y

1.1.21 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        do {
            System. out .print( "SLOW-" );
        } while ( false );

        System. out .println( "DOWN" );
    }
}
```

Which of the following statements is correct for above code?

A. Compiles successfully and on execution prints DOWN

B. Compiles successfully and on execution prints SLOW- in infinite loop

C. Unreachable code compilation error

D. Compiles successfully and on execution prints SLOW-DOWN

1.1.22 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static void m( int x) {
        System. out .println( "INT VERSION" );
    }

    private static void m( char x) {
        System. out .println( "CHAR VERSION" );
    }

    public static void main(String [] args) {
        int i = '5' ;
        m (i);
        m ( '5' );
    }
}
```

What will be the result of compiling and executing Test class?

A.	INT VERSION INT VERSION	B.	CHAR VERSION CHAR VERSION
C.	INT VERSION CHAR VERSION	D.	CHAR VERSION INT VERSION
E.	Compilation error		

1.1.23 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    static String msg ; //Line 2
    public static void main(String[] args) {
        String msg; //Line 4
         if (args. length > 0 ) {
            msg = args[ 0 ]; //Line 6
         }
        System. out .println(msg); //Line 8
    }
}
```

What will be the result of compiling and executing Test class?

A. null

B. Line 2 causes compilation failure

C. Line 4 causes compilation failure

D. An exception is thrown at runtime by Line 6

E. Line 8 causes compilation failure

1.1.24 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        byte b1 = 10 ; //Line n1
        int i1 = b1; //Line n2
        byte b2 = i1; //Line n3
        System. out .println(b1 + i1 + b2);
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. 30 is printed on to the console

1.1.25 For the given code fragment:

```
package com.udayankhattry.ocp1;
```

```
abstract class Player {
   protected void play() {

    }

    abstract void run();
}

class FootballPlayer extends Player {
   void play() {

    }

    protected void run() {

    }
}
```

Which 2 modifications, done independently, enable the code to compile?

A. Make the play() method of Player class public

B. Make the run() method of Player class public

C. Make the play() method of FootballPlayer class public

D. Make the play() method of FootballPlayer class protected

E. Make the run() method of FootballPlayer class public

1.1.26 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static void test() throws Exception {
        throw new Exception();
   }

    public static void main(String [] args) {
        try {
            test ();
       } finally {
           System. out .println( "GAME ON" );
       }
    }
}
```

What will be the result of compiling and executing Test class?

A. GAME ON is printed to the console and program ends normally

B. GAME ON is printed to the console, stack trace is printed and then program ends normally

C. GAME ON is printed to the console, stack trace is printed and then program ends abruptly

D. Compilation error

1.1.27 Given below the directory/file structure on Windows platform:

```
C:
+---source_files
|   \---com.udayan.test
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                       Main.java
|
\---class_files
```

Below is the code of C:\source_files\com.udayan.test\module-info.java file:

```
module com.udayan.test {
}
```

and below is the code of C:\source_files\com.udayan.test\com\udayankhattry\ocp1\Main.java file:

```
package com.udayankhattry.ocp1;

public class Main {
    public static void main(String... args) {
        var module = "MODULE" ; //Line n1
        var requires = "REQUIRES" ; //Line n2
        var exports = "EXPORTS" ; //Line n3

          System. out .println(String. join ( ":" , module,
             requires, exports)); //Line n4
    }
}
```

And the below two commands executed from C:\

```
javac -d class_files --module-source-path source_files --
module com.udayan.test
java -p class_files --module
com.udayan.test/com.udayankhattry.ocp1.Main
```

What is the result?

A. Compilation error in module-info.java file

B. Compilation error in Main.java file

C. An exception is thrown at runtime

D. MODULE:REQUIRES:EXPORTS is printed on to the console

1.1.28 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        for ( int x = 10 , y = 11 , z = 12 ;
             y > x && z > y; y++, z -= 2 ) {
            System. out .println(x + y + z);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	32	B.	33
C.	34	D.	33 32

E.	Compilation error		

1.1.29 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Perishable1 {
    default int maxDays() {
        return 1 ;
    }
}

interface Perishable2 extends Perishable1 {
    default int maxDays() {
        return 2 ;
    }
}

class Milk implements Perishable2, Perishable1 {}

public class Test {
    public static void main(String[] args) {
        Perishable1 obj = new Milk();
        System. out .println(obj.maxDays());
    }
}
```

Which of the following statements is correct?

A. Interface Perishable2 causes compilation error

B. Class Milk causes compilation error

C. Class Test causes compilation error

D. Given code compiles successfully and on execution Test class prints 1 on to the console

E. Given code compiles successfully and on execution Test class prints 2 on to the console

1.1.30 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<Integer> list = new ArrayList<>();
       list.add( 15 );
       list.add( 25 );
       list.add( 15 );
       list.add( 25 );
       list.remove(Integer. valueOf ( 15 ));

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [25, 15, 25]

B. [15, 25, 25]

C. [25, 25]

D. [25]

E. Compilation error

F. Exception is thrown at runtime

1.1.31 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        boolean flag1 = "Java" ==
             "Java" .replace( 'J' , 'J' ); //Line n1
        boolean flag2 = "Java" ==
             "Java" .replace( "J" , "J" ); //Line n2
        System. out .println(flag1 && flag2);
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error
B. Line n2 causes compilation error
C. true
D. false

1.1.32 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Shape {
    int side = 0 ; //Line n1

     int getSide() { //Line n2
        return side ;
    }
}

class Square extends Shape {
    private int side = 4 ; //Line n3

     protected int getSide() { //Line n4
        return side ;
    }
}

public class Test {
     public static void main(String[] args) {
        Shape s = new Square();
        System. out .println(s. side + ":" + s.getSide());
    }
}
```

What will be the result of compiling and executing above code?

A. Compilation error at Line n3
B. Compilation error at Line n4
C. 0:0
D. 0:4
E. 4:4
F. 4:0

1.1.33 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        byte var = 127 ;
        System. out .println( /*INSERT*/ );
    }
}
```

Range of byte data type is from -128 to 127.

Following options are available to replace /*INSERT*/:

1. var = var - 1
2. var = var + 1
3. ++var
4. --var
5. var *= 2
6. var -= 10
7. var += 2
8. var

How many above options can be used to replace /*INSERT*/ (separately and not together) such that there is no compilation error?

A. One option only

B. Two options only

C. Three options only

D. Four options only

E. Five options only

F. Six options only

G. Seven options only

H. All eight options

1.1.34 Consider below code snippet:

```
int i = 10 ;
```

```
System. out .println(i > 3 != false );
```

What is the result?

A. Compilation error
B. true
C. false
D. null

1.1.35 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Parent {
    default void earn() {
      System. out .println( "Earning for the family" );
   }

    static void plan() {
      planRetirement ();
      planChildrenEducation ();
   }

    private static void planChildrenEducation() {
      //valid codes
    }

    private static void planRetirement() {
      //valid codes
    }

    String toString();
}

interface Child extends Parent {
    void play();
}

public class Test {
    public static void main(String[] args) {
      Child child = () -> System. out .println(
                "PLAYING CRICKET..." ); //Line n1
        child.play(); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in Parent interface

B. Compilation error in Child interface

C. Compilation error in Test class

D. On execution, Test class prints PLAYING CRICKET... on to the console

E. On execution, Test class throws an Exception

1.1.36 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        try {
            for (;;) {
                sb.append( "1Z0-815" );
            }
        } catch (Exception e) {
            System. out .println( "Exception!!!" );
        }
        System. out .println( "Main ends!!!" );
    }
}
```

What will be the result of compiling and executing Test class?

A. "Main ends!!!" is printed on to the console and program terminates successfully

B. "Exception!!!" and "Main ends!!!" are printed on to the console and program terminates successfully

C. "Exception!!!" is printed on to the console and program terminates successfully

D. "Exception!!!" is printed on to the console and program terminates abruptly

E. Program terminates abruptly

1.1.37 On Windows platform below directories/files are available:

C:\src\com.training\com\udayankhattry\ocp1\Training.java

Which of the following correctly defines the module descriptor for 'com.training' module?
Please note that commented line in each option represents the module descriptor file name.

A.	*//module-info.java*
	module com.training {
	}
		B.	*//module.java*
			module com.training {
			requires java.base;
			}

C.	*//Module-Info.java*
	module com.training {
	}
		D.	*//Module.java*
			module training {
			}

E.	*//module-descriptor.java*
	module com.training {
	}
		F.	*//module-descriptor.java*
			module com.training {
			requires java.base;
			}

1.1.38 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

abstract class Animal {
   abstract void jump() throws RuntimeException;
}

class Deer extends Animal {
    void jump() { //Line n1
        System. out .println( "DEER JUMPS" );
   }

     void jump( int i) {
        System. out .println( "DEER JUMPS TO " + i + " FEET" );
    }
}

public class Test {
    public static void main(String[] args) {
        Animal animal = new Deer();
        ((Deer)animal).jump(); //Line n2
         ((Deer)animal).jump( 5 ); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A.	Line n1 causes compilation error
B.	Line n2 causes compilation error
C.	Line n3 causes compilation error
D.	An exception is thrown at runtime
E.	Test class executes successfully and prints: DEER JUMPS DEER JUMPS TO 5 FEET

1.1.39 Which of the following is a valid module descriptor file contents for 'agriculture' module?

A.	public module agriculture {
	}

B.	module agriculture {
	}
	
	module farming {
	}

C.	module agriculture extends java.se{ }

D.	module agriculture {
	    imports java.se;
	}

E.	module agriculture {

	import java.se;
	}

F.	import java.lang.*;
	module agriculture {
	}

G.	None of the other options

1.1.40 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.sql.SQLException;

public class Test {
    private static void availableSeats() throws SQLException {
        throw null ; //Line 7
    }

    public static void main(String[] args) {
        try {
            availableSeats (); //Line 12
        } catch (SQLException e) {
            System. out .println( "SEATS NOT AVAILABLE" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. SEATS NOT AVAILABLE is printed on to the console and program terminates successfully

B. Program ends abruptly

C. Line 7 causes compilation failure

D. Line 12 causes compilation failure

1.1.41 Consider below code of Person.java file:

```
package com.udayankhattry.ocp1;

public class Person {
    public String name ;
    public void Person() {
       name = "James" ;
    }

    public static void main(String [] args) {
      Person obj = new Person();
      System. out .println(obj. name );
    }
}
```

What will be the result of compiling and executing Person class?

A. null

B. James

C. Compilation error

D. None of the other options

1.1.42 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
      List<String> list = new ArrayList<>();
      list.add( "ONE" );
      list.add( "TWO" );
      list.add( "THREE" );
      list.add( "THREE" );

        if (list.remove( 2 )) {
         list.remove( "THREE" );
      }

        System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [ONE, TWO, THREE, THREE]

B. [ONE, TWO, THREE]

C. [ONE, TWO]

D. Compilation error

E. An exception is thrown at runtime

1.1.43 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       Boolean [] arr = new Boolean[ 2 ];
       System. out .println(arr[ 0 ] + ":" + arr[ 1 ]);
   }
}
```

What will be the result of compiling and executing Test class?

A. An Exception is thrown at runtime

B. true:true

C. false:false

D. null:null

1.1.44 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String s1 = new String( "Java" ); //Line n3
        String s2 = "JaVa" ; //Line n4
        String s3 = "JaVa" ; //Line n5
        String s4 = "Java" ; //Line n6
        String s5 = "Java" ; //Line n7

          int i = 1 ; //Line n9
    }
}
```

How many String objects are there in the HEAP memory, when control is at Line n9?

A. 2

B. 3

C. 4

D. 5

1.1.45 Consider below code of "01_Msg" file:

```
package com.udayankhattry.ocp1;

class /*INSERT*/ {
    public static void main(String [] args) {
        System.out.println( "ALL IS WELL" );
    }
}
```

And the command:

```
java --source 11 01_Msg
```

Following options are available to replace /*INSERT*/:

1. Msg
2. 01_Msg
3. 01Msg
4. Msg01
5. Msg_01
6. Message
7. $msg

How many of the above options can be used to replace /*INSERT*/ (separately and not together) such that given command prints ALL IS WELL on to the console?

A. One option only

B. Two options only

C. Three options only

D. Four options only

E. Five options only

F. Six options only

G. All seven options

1.1.46 Given:

```
class A{}
class B extends A{}
```

Which of the following is not a valid statement based on above code?

A. B obj1 = new A();

B. A obj2 = new B();

C. B obj3 = new B();

D. A obj4 = new A();

1.1.47 Which of the following statement declares a constant field in Java?

A. const int x = 10;

B. static int x = 10;

C. final static int x = 10;

D. int x = 10;

1.1.48 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int val = 25 ;
        if (val++ < 26 ) {
            System. out .println(val++);
        }
    }
}
```

A. 25

B. 26

C. 27

D. Program executes successfully but nothing is printed on to the console

1.1.49 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        byte var = 100 ;
        switch (var) {
            case 100 :
               System. out .println( "var is 100" );
                break ;
            case 200 :
               System. out .println( "var is 200" );
                break ;
            default :
               System. out .println( "In default" );
        }
    }
}
```

A. var is 100

B. var is 200

C. In default

D. Compilation error

1.1.50 A module is a set of _____________.

A. packages

B. classes

C. interfaces

D. source code files

1.1.51 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static void div( int i, int j) {
        try {
           System. out .println(i / j);
        } catch (ArithmeticException e) {
           Exception ex = new Exception(e);
            throw ex;
        }
    }
    public static void main(String[] args) {
        try {
            div ( 5 , 0 );
        } catch (Exception e) {
           System. out .println( "END" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. END is printed and program terminates successfully

C. END is printed and program terminates abruptly

D. END is not printed and program terminates abruptly

1.1.52 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
      List<String> dryFruits = new ArrayList<>();
      dryFruits.add( "Walnut" );
      dryFruits.add( "Apricot" );
      dryFruits.add( "Almond" );
      dryFruits.add( "Date" );

        for (String dryFruit : dryFruits) {
         if (dryFruit.startsWith( "A" )) {
           dryFruits.remove(dryFruit);
         }
      }

        System. out .println(dryFruits);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Walnut, Apricot, Almond, Date]

B. [Walnut, Date]

C. An exception is thrown at runtime

D. Compilation error

1.1.53 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       int sum = 0 ;
       for ( var i = 0 ; i < 7 ; i++) { //Line n1
          if (i == 4 )
             break ;
          else
            continue ;
         sum += i; //Line n2
       }
      System. out .println(sum); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A. 10
B. 21
C. 17
D. 6
E. 4
F. 0
G. Compilation error at Line n1
H. Compilation error at Line n2
I. Compilation error at Line n3

1.1.54 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var var = 3 ; //Line n1
        String [][] arr = new String[--var][var++]; //Line n2
        arr[ 1 ][ 1 ] = "X" ; //Line n3
        arr[ 1 ][ 2 ] = "Y" ; //Line n4
        for (String [] arr1 : arr) {
            for (String s : arr1) {
                if (s != null )
                    System. out .print(s);
            }
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. It causes compilation error at single statement
B. It causes compilation error at multiple statements
C. It throws an exception at runtime

D. It prints XY on to the console and program terminates successfully

E. It prints XY on to the console and program terminates abruptly

1.1.55 Consider below code of Test.java file:

```
class Test3 {
    public static void main(String [] args) {
        System. out .println( "Test3" );
    }
}

class Test2 {
    public static void main(String [] args) {
        System. out .println( "Test2" );
    }
}

class Test1 {
    public static void main(String [] args) {
        System. out .println( "Test1" );
    }
}
```

And the command:

```
java Test.java
```

What is the result?

A.	Test1	B.	Test2
C.	Test3	D.	Test1 Test2 Test3
E.	Test3 Test2 Test1	F.	Above command causes error

1.1.56 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Document {
     int pages ;
    Document( int pages) {
        this . pages = pages;
    }
}

class Word extends Document {
    String type ;
    Word(String type) {
        super ( 20 ); //default pages
        this . type = type;
    }

     Word( int pages, String type) {
        // TODO
     }
}

public class Test {
     public static void main(String[] args) {
        Word obj = new Word( 25 , "TEXT" );
        System. out .println(String. join ( "," ,
              obj. type , obj. pages + "" ));
    }
}
```

Currently above code causes compilation error.
Which of the options can successfully replace //TODO such that on executing Test class, output is TEXT,25?

A.	this (type);
	super . pages = pages;
		B.	this (type);
			super (pages);

C.	super (pages);
	this (type);
		D.	super . pages = pages;
			this (type);

1.1.57 Given code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
      StringBuilder sb = new StringBuilder();
      System. out .println(sb.append( "" )
            .append( "" ).append( "" ).length());
    }
}
```

What will be the result of compiling and executing Test class?

A. 0

B. 1

C. 2

D. 3

1.1.58 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
       boolean status = true ;
      System. out .println(status = false
            || status = true | status = false );
      System. out .println(status);
    }
}
```

What will be the result of compiling and executing Test class?

A.	true false	B.	false true
C.	true true	D.	false false
E.	Compilation error		

1.1.59 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Super {
    void Super() {
       System. out .print( "KEEP_" );
    }
}

class Base extends Super {
   Base() {
       Super();
       System. out .print( "GOING_" );
    }
}

public class Test {
    public static void main(String[] args) {
        new Base();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation Error in Super class

B. Compilation Error in Base class

C. Compilation Error in Test class

D. It prints KEEP_GOING_ on to the console

E. It prints GOING_KEEP_ on to the console

F. It prints KEEP_KEEP_GOING_ on to the console

G. It prints GOING_ on to the console

1.1.60 Given below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static String print(String... args) {
        return String. join ( "-" , args); //Line n1
    }
    private static Object print(Object... args) {
      String str = "" ;
       for (Object obj : args) {
          if (obj instanceof String) { //Line n2
             str += (String) obj; //Line n3
          }
      }
       return str; //Line n4
    }

    public static void main(String... args) {
      Object obj1 = new String( "SPORT" ); //Line n5
       Object obj2 = new String( "MAD" ); //Line n6
       System. out .println( print (obj1, obj2)); //Line n7
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Line n7 causes compilation error

E. Code compiles successfully and on execution prints SPORTMAD on to the console

F. Code compiles successfully and on execution prints SPORT-MAD on to the console

G. An exception is thrown at runtime

1.1.61 _____________ uses access modifiers to protect variables and hide them within a class.

Which of the following options accurately fill in the blank above?

A. Polymorphism

B. Inheritance

C. Encapsulation

D. Abstraction

1.1.62 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String s1 = "OCP" ;
       String s2 = "ocp" ;
       System. out .println( /*INSERT*/ );
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print true on to the console?
Select ALL that apply.

A. s1.equals(s2)

B. s1.equals(s2.toUpper())

C. s2.equals(s1.toLower())

D. s1.length() == s2.length()

E. s1.equalsIgnoreCase(s2)

F. s1.contentEquals(s2)

1.1.63 Consider below code of Guest.java file:

```
class Message {
    public static void main(String [] args) {
       System. out .println( "Welcome! " + args[ 1 ]);
    }
}

public class Guest {
    public static void main(String [] args) {
       Message. main (args);
    }
}
```

And the commands:

```
javac Guest.java
```

```
java Guest Arya Stark
```

What is the result?

A. Welcome! Arya
B. Welcome! Stark
C. An exception is thrown at runtime
D. Message.main(args); causes compilation error in Guest class

1.1.64 Consider below code of TestNewsPaper.java file:

```
package com.udayankhattry.ocp1;

class NewsPaper {

}

public class TestNewsPaper {
    public static void main(String[] args) {
        new NewsPaper(); //Line n1
        NewsPaper p = new NewsPaper(); // Line n2
        change (p); //Line n3
        System. out .println( "About to end." ); //Line n4
    }

    public static void change(NewsPaper np) { //Line n5
        np = new NewsPaper(); //Line n6
    }
}
```

On executing TestNewsPaper class, how many objects of NewsPaper class will be eligible for Garbage Collection at Line n4?

A. 0
B. 1
C. 2
D. 3

1.1.65 Given below the directory/file structure on Windows

platform:

```
C:
+---source
|   \---tools
|       |  module-info.java
|       |
|       \---com
|           \---it
|               \---tools
|                       Tools.java
|                       Eclipse.java
|
\---classes
```

Below are the file contents:-

C:\source\tools\module-info.java:

```
module tools {
    exports com.it; //Line n1
    exports com.it.tools; //Line n2
    exports com.it.tools.ide; //Line n3
}
```

C:\source\tools\com\it\tools\Tools.java:

```
package com.it.tools;

public abstract class Tools {
   //Lots of valid codes
}
```

C:\source\tools\com\it\tools\Eclipse.java:

```
package com.it.tools.ide;

public class Eclipse {
   //Lots of valid codes
}
```

And below javac command is executed from C:\

```
javac -d classes --module-source-path source -m tools
```

What is the result?

A. Code compiles without any warning
B. Compilation error for Line n1 only
C. Compilation error for Line n3 only
D. Compilation error for both Line n1 and Line n3
E. Code compiles but with a warning for Line n1 only
F. Code compiles but with a warning for Line n3 only
G. Code compiles but with a warning for both Line n1 and Line n3
H. Compilation error for Eclipse.java file

1.1.66 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        byte [] arr = new byte [ 0 ];
       System. out .println(arr[ 0 ]);
    }
}
```

Which of the following is true for the above code?

A. Above code causes compilation error
B. 0 is printed on to the console
C. null is printed on to the console
D. An exception is thrown at runtime

1.1.67 Given below the directory/file structure on Windows platform:

```
C:
|-------src
|    └------calculator
|          |module-info.java
|          |
|          └------com
|
│          └───udayankhattry
│              └───ocp1
│                      Calculator.java
│
└───out
    └───calc-module
```

Below is the code of Calculator.java file:

```
package com.udayankhattry.ocp1;

public class Calculator {
    public static int add( int n1, int n2) {
        return n1 + n2;
    }

     public static void main(String [] args) {
       System. out .println(Calculator. add ( 20 , 40 ));
    }
}
```

And below is the code of module-info.java file:

```
module calc {
}
```

Which of the following set of commands if executed from C:, successfully compile and execute the given code such that output is: 60?

A.	javac -d out\calc-module src\calculator\com\udayankhattry\ocp1\Calculator.java src\calculator\module-info.java java --module-path out --module calc/com.udayankhattry.ocp1.Calculator
B.	javac -d out\calc-module src\calculator\com\udayankhattry\ocp1\Calculator.java java --module-path out --module calculator/com.udayankhattry.ocp1.Calculator
C.	javac -d out\calc-module src\calculator\module-info.java

	java --module-path out --module calc-module/com.udayankhattry.ocp1.Calculator
D.	javac -d out --module-source-path src --module calculator java -p out -m calc/com.udayankhattry.ocp1.Calculator
E.	javac -d out --module-source-path src --module calc java -p out -m calculator/com.udayankhattry.ocp1.Calculator
F.	javac -d out --module-source-path src --module calc-module java -p out -m calc-module/com.udayankhattry.ocp1.Calculator

1.1.68 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;

public class Test {
    public static void main(String[] args) {
        try {
            findSecretFile ();
        } catch (Throwable ex) {
            System. out .println(ex.getMessage());
            return ;
        } finally {
            System. out .println( "LEVEL 1" );
        }
        System. out .println( "DONE" );
    }

    static void findSecretFile() throws FileNotFoundException {
        throw new FileNotFoundException( "ACCESS REQUIRED" );
    }
}
```

What will be the result of compiling and executing Test class?

A.	ACCESS REQUIRED LEVEL 1 DONE	B.	ACCESS REQUIRED LEVEL 1

C.	ACCESS REQUIRED	D.	ACCESS REQUIRED DONE
E.	Compilation error		

1.1.69 Consider codes of 3 java files:

```
//Order.java
package orders;

public class Order {

}


//Item.java
package orders.items;

public class Item {

}


//Shop.java
package shopping;

/*INSERT*/

public class Shop {
    Order order = null ;
    Item item = null ;

}
```

For the class Shop, which options, if used to replace /*INSERT*/, will resolve all the compilation errors?
Select 2 options.

A.	**import** orders.Order;
	import orders.items.Item;

B.	**import** orders.*;
C.	**import** orders.items.*;

D.	**import** orders.*;
	import orders.items.*;

E.	**import** orders.*;
	import items.*;

1.1.70 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.List;

public class Test {
    public static void main(String[] args) {
       var festivals = List. of ( "DIWALI" , "CHRISTMAS" , "EID" );
      festivals.removeIf(str -> str.length() == 3 );
      System. out .println(festivals);
   }
}
```

What will be the result of compiling and executing Test class?

A. [DIWALI, CHRISTMAS, EID]

B. [DIWALI, CHRISTMAS]

C. []

D. Compilation error

E. An exception is thrown at runtime

1.1.71 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String [][] countries = {{ "SINGAPORE" , "AUSTRALIA" },
                { "CHINA" , "RUSSIA" }};
       /*INSERT*/
    }
}
```

For the class Test, which options, if used to replace /*INSERT*/, will

print "SINGAPORE AUSTRALIA CHINA RUSSIA " on to the console?

A	. for (int i = 0 ; i < countries. length ; i++)
	for (int j = 0 ; j < countries[i]. length ; j++)
	System. *out* .print(countries[i][j] + " ");

B	. for (int i = 1 ; i <= countries. length ; i++)
	for (int j = 1 ; j <= countries[i]. length ; j++)
	System. *out* .print(countries[i][j] + " ");

C	. for (int i = 1 ; i < countries. length ; i++)
	for (int j = 1 ; j < countries[i]. length ; j++)
	System. *out* .print(countries[i][j] + " ");

D	. for (String [] arr : countries)
	for (String country : arr)
	System. *out* .print(country + " ");

1.1.72 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       int i = 0 ;
       for (System. out .print(i++); i < 2 ;
             System. out .print(i++)) {
          System. out .print(i);
       }
    }
}
```

What will be the result of compiling and executing Test class?

A. 112

B. 012

C. 011

D. 12

E. 01

F. Compilation error

1.1.73 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        operate ( 10 , 20 );
        operate ( "A" , "B" );
        operate ( 'A' , 1 ); //ASCII code of 'A' is 65
    }

    public static void operate(var v1, var v2) {
        System. out .println(v1 + v2);
    }
}
```

What is the result of compiling and executing above code?

A.	30 AB 66	B.	30 AB A1
C.	1020 AB A1	D.	1020 AB 66
E.	Compilation error	F.	Runtime error

1.1.74 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String str = "USER GENERATED" ;
        str = String. join ( "-" , str.split( " " ));
        System. out .println(str);
    }
}
```

What is the result of compiling and executing Test class?

A. USER GENERATED

B. USER-GENERATED

C. -

D. Compilation error

E. An exception is thrown at runtime

1.1.75 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Car {
    void speed(Byte val) { //Line n1
        System. out .println( "DARK" ); //Line n2
    } //Line n3

    void speed( byte ... vals) {
      System. out .println( "LIGHT" );
    }
}

public class Test {
    public static void main(String[] args) {
        byte b = 10 ; //Line n4
        new Car().speed(b); //Line n5
    }
}
```

Which of the following needs to be done so that LIGHT is printed on to the console?

A. No changes are required as given code prints LIGHT on execution

B. Delete Line n1, Line n2 and Line n3

C. Replace Line n4 with byte... b = 10;

D. Replace Line n5 with new Car().speed((byte...)b);

1.1.76 For the class Test, which of the following options, if used to replace /*INSERT*/, will print "BUY 2 GET 1 FREE" on to the console?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       /*INSERT*/
       switch (day) {
          case '3' :
            System. out .println( "BUY 2 GET 1 FREE" );
             break ;
          default :
            System. out .println( "SORRY!!! NO SALE" );
       }
    }
}
```

What will be the result of compiling and executing Test class?

A. int day = 3;

B. Integer day = 3;

C. int day = '3';

D. None of the other options

1.1.77 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Counter {
    int count = 0 ;
}
interface SetCounter {
    void set(Counter ctr);
}

public class Test {
    public static void main(String[] args) {
      Counter ctr = new Counter(); //Line n1
       SetCounter obj = x -> x. count = 20 ; //Line n2
       ctr. count = 100 ; //Line n3
       obj.set(ctr); //Line n4
```

```
        System. out .println(ctr. count ); //Line n5
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Exception is thrown at runtime

C. 20

D. 100

1.1.78 Given below the directory/file structure on Windows platform:

```
C:
+---dir1
|   \---shopping
|       |  module-info.java
|       |
|       \---com
|           \---shopping
|               \---online
|                       Product.java
|
\---dir2
```

Contents of C:\dir1\shopping\module-info.java file:

```
module shopping {
    exports com.shopping.online;
}
```

Contents of C:\dir1\shopping\com\shopping\online\Product.java file:

```
package com.shopping.online;

public class Product {
   //Lots of valid codes
}
```

And the command executed from C:\

```
javac -d dir2 --module-source-path dir1 -m shopping
```

Which of the following statements is correct?

A.	javac command fails as module-info.java file is not valid
B.	Code compiles successfully and path of generated module-info.class file is C:\dir2\shopping\module-info.class
C.	Code compiles successfully and path of generated module-info.class file is C:\dir2\module-info.class
D.	Code compiles successfully and path of generated module-info.class file is C:\dir2\shopping\com\shopping\online\module-info.class

1.1.79 What will be the result of compiling and executing Test class?

```
//Test.java
package com.udayankhattry.ocp1;

class Student {
    String name ;
     int marks ;

      Student(String name, int marks) {
         this . name = name;
         this . marks = marks;
    }
}

public class Test {
     public static void main(String[] args) {
        Student student = new Student( "James" , 25 );
         int marks = 25 ;
         review (student, marks);
        System. out .println(marks + "-" + student. marks );
    }

     private static void review(Student stud, int marks) {
        marks = marks + 10 ;
        stud. marks +=marks;
    }
}
```

What will be the result of compiling and executing Test class?

A. 25-25

B. 35-25

C. 35-60

D. 25-60

1.1.80 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;

class Counter {
     int count ;
    Counter( int count) {
         this . count = count;
    }

     public String toString() {
         return "Counter-" + count ;
    }
}

public class Test {
     public static void main(String[] args) {
        ArrayList<Counter> original
                = new ArrayList<>(); //Line n1
         original.add( new Counter( 10 )); //Line n2

            ArrayList<Counter> cloned
                = (ArrayList<Counter>) original.clone();
        cloned.get( 0 ). count = 5 ;
           System. out .println(original);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Counter-5]

B. [Counter-10]

C. Compilation error

D. An exception is thrown at runtime

1.2 Answers of Practice Test - 1 with Explanation

1.1.1 Answer: D

Reason :
Method print() of Derived class correctly overrides the method print() of Base class, so there is no compilation error in Derived class.

java.io.FileNotFoundException extends java.io.IOException
and
java.io.IOException extends java.lang.Exception

Even though an instance of FileNotFoundException is thrown by method print() at runtime, but method print() declares to throw IOException.
Reference variable 'b' is of Base type and hence for compiler, call to b.print(); is to method print() of Base, which throws IOException.
And as IOException is checked exception hence calling code should handle it.

As calling code doesn't handle IOException or its supertype, so b.print(); causes compilation error.

1.1.2 Answer: F

Reason :
Starting with JDK 7, Java allows to not specify type while initializing the ArrayList. As variable list is of List<String> type, therefore type of ArrayList is considered as String. Line n1 compiles successfully.

sublist method is declared in List interface:
List<E> subList(int fromIndex, int toIndex)
fromIndex is inclusive and toIndex is exclusive
It returns a view of the portion of this list between the specified fromIndex and toIndex. The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list and vice-versa .
If returned list (or view) is structurally modified, then modification are reflected in this list as well but if this list is structurally modified,

then the semantics of the list returned by this method become undefined.
If fromIndex == toIndex, then returned list is empty.
If fromIndex < 0 OR toIndex > size of the list OR fromIndex > toIndex, then IndexOutOfBoundsException is thrown.

At Line n2, list.subList(0, 4) --> [A, E, I, O] (toIndex is Exclusive, therefore start index is 0 and end index is 3].

list.addAll(list.subList(0, 4)); is almost equal to list.addAll(5, [A, E, I, O]); => Inserts A at index 5, E takes index 6, I takes index 7 and O is placed at index 8. list --> [A, E, I, O, U, A, E, I, O]

Last statement inside main(String []) method prints [A, E, I, O, U, A, E, I, O] on to the console.

1.1.3 Answer: D,F

Reason :
Variable 'salePercentage' declared inside interface Buyable is implicitly public, static and final. As per Java 8, default and static methods were added in the interface and as per Java 9, private methods were added in the interface. There is no compilation error in Buyable.java file.
class Book implements Buyable interface but as there is no abstract method in Buyable interface, hence Book class is not needed to implement any method. Book.java file compiles successfully.

`Buyable [] arr = new Buyable[2];` creates one dimensional array of 2 elements of Buyable type and both the elements are initialized to null.

There are some difference in which static variables and static methods of the interface are accessed.
Correct and only way to access static method of an Interface is by using the name of the interface, such as Buyable.salePercentage().
Line n2 and Line n4 cause compilation error.

As far as public static final variable of interface is concerned, even through the correct way to access static variable is by using the name of the interface, such as Buyable.salePercentage but it can also be

accessed by using following :
Reference variable of the interface: Buyable obj1 = null;
System.out.println(obj1.salePercentage);
Name of the implementer class:
System.out.println(Book.salePercentage);
Reference variable of the implementer class: Book obj2 = null;
System.out.println(obj2.salePercentage);
Hence, Line n1 and Line n3 compile successfully.

1.1.4 Answer: D

Reason :
Lambda expression doesn't work without target type and target type must be a functional interface, so in this case as the given lambda expression is not assigned to any target type, hence its purpose is not clear. In fact, given lambda expression causes compilation error without its target type.

1.1.5 Answer: E

Reason :
Please note that Strings computed by concatenation at compile time are referred by String Pool. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc. This means the result of constant expression is calculated at compile time and later referred by String Pool.
Where as Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

`java += world;` is same as `java = java + world;` and `java + world` is not a constant expression and hence is calculated at runtime and returns a non pool String object "JavaWorld", which is referred by variable 'java'.

On the other hand, variable 'javaworld' refers to String Pool object "JavaWorld". As both the variables 'java' and 'javaworld' refer to different String objects, hence `java == javaworld` returns false.

1.1.6 Answer: B

Reason :
Top-level class can use only two access modifiers [public and default(don't specify anything)]. private and protected cannot be used.
As file name is M.java, hence class N cannot be public .
Top-level class can be final, hence it is a correct option.
Top-level class can be abstract and hence it is also a correct option.

1.1.7 Answer: B

Reason :
There are 2 rules related to return types of overriding method:
1. If return type of overridden method is of primitive type, then overriding method should use same primitive type.
2. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

ArrayList is a subtype of List, hence overriding method can use List<Father> or ArrayList<Father> as return type. Definitions 1 and 2 are valid.

Please note: even though Son is a subtype of Father, List<Son> is not subtype of List<Father>. Hence definitions 3 and 4 are NOT valid.

On similar lines, even though GrandSon is a subtype of Father, List<GrandSon> is not subtype of List<Father>. Hence definitions 5 and 6 are also NOT valid.

List<Object> is not subtype of List<Father>, definition 7 is NOT valid.

ArrayList<Object> is not subtype of List<Father>, definition 7 is also NOT valid.

1.1.8 Answer: C

Reason :
An Error is a subclass of Throwable class.

1.1.9 Answer: A,B,D,F

Reason :
Static List.of() overloaded methods were added in Java 9 and these

return an unmodifiable list containing passed elements.

Let's check all the options one by one:
p -> true ✓ Means test method returns true for the passed String. It will print all the elements of the List .
p -> !!!!true ✓ !!!!true => !!!false => !!true => !false => true, means test method returns true for the passed String. It will print all the elements of the List.
p -> !!false ✗ !!false => !true => false, means test method returns false for the passed String. It will not print even a single element of the list.
p -> p.length() >= 1 ✓ Means test method returns true if passed String's length is greater than or equal to 1 and this is true for all the list elements.
p -> p.length() < 7 ✗ Means test method returns true if passed String's length is less than 7 and this is not true for "whether". "whether" will not be displayed in the output.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

(var p) -> p.length() < 100 ✓ Type of p is inferred from the target type, which is Predicate<String>. This means 'p' is of String type.
var p -> p.length() > 0 ✗ Round brackets or parenthesis are missing around 'var p'. This causes compilation error.

1.1.10 Answer: D

Reason :
x of checkAndIncrement method contains the copy of variable x defined in main method. So, changes done to x in checkAndIncrement method are not reflected in the variable x of main. x of main remains 1 as code inside main is not changing its value.

Every time checkAndIncrement method is invoked with argument value 1, so true is returned always and hence while loop executes indefinitely.

1.1.11 Answer: B

Reason :
The static method of subclass cannot hide the instance method of superclass. static main(String []) method at Line n2 tries to hide the instance main(String []) method at Line n1 and hence Line n2 causes compilation error .

There is no issue with Line n3 as it is a valid syntax to invoke the instance main(String []) method of M class.

No issue with Line n4 as well as it correctly invokes static main(String []) method of N class.

1.1.12 Answer: C

Reason :
As per Java 8, default methods were added in the interface. Interface Document defines default method getType(), there is no compilation error in interface Document. Method getType() is implicitly public in Document.

interface WordDocument extends Document and it overrides the default method getType() of Document, overriding method in WordDocument is implicitly abstract and public. An interface in java can override the default method of super type with abstract modifier. interface WordDocument compiles successfully.

class Word implements WordDocument and as WordDocument interface has abstract method getType(), and as class Word doesn't implement the getType() method hence it causes compilation failure.

1.1.13 Answer: B

Reason :
As floating point numbers are used in the expression, hence result should be in floating point number.
Correct result is:
53 / 2.0 = 26.5
53 % 2.0 = 1.0

1.1.14 Answer: B

Reason :

list.add(0, 'E'); => char 'E' is converted to Character object and stored as the first element in the list. list --> [E].
list.add('X'); => char 'X' is auto-boxed to Character object and stored at the end of the list. list --> [E,X].
list.add(1, 'P'); => char 'P' is auto-boxed to Character object and inserted at index 1 of the list, this shifts X to the right. list --> [E,P,X].
list.add(3, 'O'); => char 'O' is auto-boxed to Character object and added at index 3 of the list. list --> [E,P,X,O].
list.contains('O') => char 'O' is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove('O');.

remove method is overloaded: remove(int) and remove(Object). char can be easily assigned to int so compiler tags remove(int) method. list.remove(<ASCCI value of 'O'>); ASCCI value of 'A' is 65 (this everybody knows) so ASCII value of 'O' will be more than 65.

list.remove('O') throws runtime exception, as it tries to remove an item from the index greater than 65 but allowed index is 0 to 3 only.

1.1.15 Answer: C

Reason :

Lambda expression's variables x and y cannot redeclare another local variables defined in the enclosing scope.
`Operation o1 = (x, y) -> x * y;` causes compilation error.

1.1.16 Answer: D

Reason :

As per Java 8, default and static methods were added in the interface and default methods can invoke static method as well. Hence, there is no issue with the Flyable interface.

class Aeroplane implements Flyable interface, hence it inherits the default method fly() and static method horizontalDegree() can be accessed using Flyable.horizontalDegree(). It also provides the implementation of land() method. There is no issue with Aeroplane class as well.

On execution below text is printed on to the console:
Flying at 20 degrees.
Landing at -20 degrees.

1.1.17 Answer: C

Reason :
Line n1 creates a String [] object of 4 elements. arr[0] --> "A", arr[1] --> "B", arr[2] --> "C" and arr[3] --> "D".
At Line n2, arr[0] --> "B".
At Line n3, arr[1] --> "E".
So, after Line n3, arr refers to {"B", "E", "C", "D"}.
for-each loop prints the array elements referred by arr and hence the output is: B E C D

1.1.18 Answer: B

Reason :
JRE is no longer offered separately in JDK 11, only the JDK is offered. Smaller custom runtimes are created using jlink.

1.1.19 Answer: C

Reason :
toString() method defined in StringBuilder class doesn't use String literal rather uses the constructor of String class to create the instance of String class.

So both 's1' and 's2' refer to different String instances even though their contents are same. s1 == s2 returns false.

1.1.20 Answer: B

Reason :
Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this

case, module source path is 'src' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma .

-d <directory>:
Specify where to place generated class files. In this case, it is 'bin' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
It is not used in this case as given module is not dependent upon other compiled/packaged application modules.

Let's check all the commands one by one:
javac -d bin --module-source-path src --module x: ✗ Module descriptor of module 'x' doesn't have 'requires y;' directive, hence this command will compile all the java files of module 'x' only.

javac -d bin --module-source-path src --module y: ✓ Module descriptor of module 'y' has 'requires x;' directive, so it will compile both the modules 'y' and 'x'.

javac -d bin --module-source-path src --module *: ✗ Wild-card (*) is not valid, multiple module names are separated by comma.

javac -d bin --module-source-path src --module x;y: ✗ x;y is not valid, multiple module names are separated by comma. If you replace 'x;y' with 'x,y', then both the modules will compile.

1.1.21 Answer: D

Reason :
As do-while loop executes at least once, hence none of the code is unreachable in this case.
Java runtime prints "SLOW-" to the console, then it checks boolean expression, which is false.
Hence control goes out of do-while block. Java runtime executes 2nd System.out.println statement to print "DOWN" on to the console.

1.1.22 Answer: C

Reason :
Method m is overloaded and which overloaded method to invoke, is decided at compile time.
m(i) is tagged to m(int) as i is of int type and m('5') is tagged to m(char) as '5' is char literal.

`m(i);` prints int version on to the console.
`m('5');` prints char version on to the console.

1.1.23 Answer: E

Reason :
Line 4 code shadows the variable at Line 2. 'msg' variable created at Line 4 is a local variable and should be initialized before it is used.
Initialization code is inside if-block, so compiler is not sure about the initialization of 'msg' variable initialization. Hence, Line 8 causes compilation failure.

1.1.24 Answer: C

Reason :
Let us first check Line n1: byte b1 = 10;
Above statement compiles successfully, even though 10 is an int literal (32 bits) and b1 is of byte primitive type which can store only 8 bits of data.
Here java does some background task, if value of int literal can be easily fit to byte primitive type (-128 to 127), then int literal is implicitly casted to byte type.
So above statement is internally converted to:
byte b1 = (byte)10;

But if you specify any out of range value then it would not be

allowed, e.g.
byte b = 128; // It would cause compilation failure as 128 is out of range value for byte type.

There is no issue with Line n2 as byte type (8 bits) can be easily assigned to int type (32 bits) .

For line n3, `byte b2 = i1;`, expression on right hand side (i1) is neither a withing range literal value nor constant expression, hence it causes compilation failure.
To compile successfully, this expression needs to be explicitly casted, such as: `byte b2 = (byte)i1;`

1.1.25 Answer: C,D

Reason :
Player class has no compilation error, so no need to change anything in Player class.
run() method of Player class is declared with default access modifier, so overriding method in FootballPlayer class must be declared with default, protected or public modifier.
run() method of FootballPlayer class is already declared with protected access modifier so there is no issue with this method.

play() method of Player class is declared with protected access modifier, so overriding method in FootballPlayer class must be declared with protected or public access modifier.
Currently play() method of FootballPlayer class is declared with default access modifier, and hence compilation error. Provide protected or public for play() method of FootballPlayer class.

1.1.26 Answer: D

Reason :
Method test() throws Exception (checked) and it declares to throw it, so no issues with method test().
But main(String []) method neither provides catch handler nor throws clause and hence main(String []) method causes compilation error.
Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

1.1.27 Answer: D

Reason :
Modules related keywords such as module, requires, exports etc. are restrictive keywords and these are treated as keywords inside module-info.java file only and can be used as valid identifiers in other java files .

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

At Line n1, module infers to String type.
At Line n2, requires infers to String type.
At Line n3, exports infers to String type.

Hence, Line n2, Line n2 and Line n3 compile successfully.

Static overloaded method join(...) was added in JDK 1.8 and one of the overloaded methods has below declarations:
public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

Hence, Line n4 compiles successfully.

It is also valid to have empty body for the module description file:
module com.udayan.test {
}

Module name is 'com.udayan.test' and it implicitly requires java.base module. There is no compilation error in module-info.java file as well.

Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path> :
Specify where to find input source files for multiple modules. In this case, module source path is 'source_files' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'class_files' directory.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
It is not used in this case as given module is not dependent upon other compiled/packaged application modules.

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
    (to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries.
For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar' .

There is no issues with the commands as well; MODULE:REQUIRES:EXPORTS is printed on to the console.

1.1.28 Answer: B

Reason :
Basic/Regular for loop has following form:
for ([ForInit] ; [Expression] ; [ForUpdate]) {...}
[ForInit] can be local variable initialization or the following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpression
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

[ForUpdate] can be following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpression
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

Multiple comma separated statements are allowed for [ForInit] and [ForUpdate] expressions, where as [Expression] must be single expression which results in boolean or Boolean.

In the given for loop :
[ForInit] = int x = 10, y = 11, z = 12: It is allowed. 3 variables are declared and initialized. x = 10, y = 11 & z = 12.
[Expression] = y > x && z > y = (y > x) && (z > y) [Relational operator has higher precedence than logical AND]. This expression is valid and results in boolean value.
[ForUpdate] = y++, z -= 2. It is allowed. y is incremented by 1 and z is decremented by 2.

Let's check the loop's iteration:
1st iteration: x = 10, y = 11, z = 12. (y > x) && (z > y) = (11 > 10) && (12 > 11) = true && true = true. Loop's body is executed and prints x + y + z = 10 + 11 + 12 = 33 on to the console.
2nd iteration: [ForUpdate] is executed. y = 12, z = 10. (y > x) && (z > y) = (12 > 10) && (10 > 12) = true && false = false.
Control goes out of for loop and program terminates successfully.

Loop's body executes once and prints 33 on to the console.

1.1.29 Answer: E

Reason :
As per Java 8, default methods were added in the interface. Interface Perishable1 defines default method maxDays(), there is no compilation error in interface Perishable1. Method maxDays() is implicitly public in Perishable1.

interface Perishable2 extends Perishable1 and it overrides the default method maxDays() of Document, overriding method in Perishable2 is implicitly public. Interface Perishable2 compiles successfully.

Class Milk implements Perishable2 and Perishable1. Although it is

redundant for Milk class to implement Preishable1 as Perishable2 already extends Perishable1.
There is no conflict in Milk class as it inherits the default method maxDays() of Perishable2 interface. Milk class compiles successfully.

`Perishable1 obj = new Milk();` It compiles fine as Perishable1 is supertype and Milk is subtype.
`obj.maxDays()` executes the default maxDays() method of Perishable2 interface and it returns 2.
`System.out.println(obj.maxDays());` prints 2 on to the console.

1.1.30 Answer: A

Reason :
List cannot accept primitives, it can accept objects only. So, when 15 and 25 are added to the list, then auto-boxing feature converts these to wrapper objects of Integer type.
So, 4 items gets added to the list: [15, 25, 15, 25].
`list.remove(Integer.valueOf(15));` matches with list.remove(Object) and this removes the first occurrence of 15 from the list, which means the 1st element of the list.
After removal list contains: [25, 15, 25].
NOTE: String class and all the wrapper classes override equals(Object) method, hence at the time of removal when another instance is passed [Integer.valueOf(15)], there is no issue in removing the matching item.

1.1.31 Answer: D

Reason :
String class has following two overloaded replace methods:
1. public String replace(char oldChar, char newChar) {}:
Returns a string resulting from replacing all occurrences of oldChar in this string with newChar. If no replacement is done, then source String object is returned. e.g.
"Java".replace('a', 'A') --> returns new String object "JAvA".
"Java".replace('a', 'a') --> returns the source String object "Java" (no change).
"Java".replace('m', "M") --> returns the source String object "Java"

(no change).

2. public String replace(CharSequence target, CharSequence replacement) {}:
Returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.
"Java".replace("a", "A") --> returns new String object "JAvA".
"Java".replace("a", "a") --> returns new String object "Java" (it replaces "a" with "a").
"Java".replace("m", "M") --> returns the source String object "Java" (no change).

For Line n1, as both oldChar and newChar are same, hence source String ("Java") is returned by `"Java".replace('J', 'J');` without any change. flag1 stores true.
For Line n2, even though target and replacement are same but as "J" is found in the source String, hence a new String object "Java" is returned by `"Java".replace("J", "J");` after replacing "J" with "J". flag2 stores false.
flag1 && flag2 evaluates to false.

1.1.32 Answer: D

Reason :
Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n3 hides the variable created at Line n1 and Line n4 overrides the getSide() method of Line n2. There is no compilation error for Square class as it correctly overrides getSide() method. You can use any access modifier at Line n3 as well, there are no rules for variable hiding.

's' is of Shape type, hence s.side equals to 0 and s.getSide() invokes overriding method of Square class and it returns 4. Hence output is: 0:4.

1.1.33 Answer: F

Reason :
var = var - 1: 'var - 1' results in int type and it cannot be assigned to

byte type without explicit casting, hence it causes compilation error.
var = var + 1: 'var + 1' results in int type and it cannot be assigned to byte type without explicit casting, hence it causes compilation error.
Please note that implicit casting is added by the compiler for prefix, postfix and compound assignment operators.
++var: Compiler converts it to var = (byte) (var + 1) and hence it compiles successfully.
--var: Compiler converts it to var = (byte) (var - 1) and hence it compiles successfully.
var *= 2: Compiler converts it to var = (byte) (var * 2) and hence it compiles successfully.
var -= 10: Compiler converts it to var = (byte) (var - 10) and hence it compiles successfully.
var += 2: Compiler converts it to var = (byte) (var + 2) and hence it compiles successfully.
var: No issues at all, it also compiles successfully.

Hence, out of the given 8 options, 6 compiles successfully.

1.1.34 Answer: B

Reason :
Given Expression:
i > 3 != false
It has 2 operators > and !=. > has higher precedence over !=, hence given expression can be written as :
(i > 3) != false
Let's solve above expression:
true != false
true

Hence true is printed on to the console.

1.1.35 Answer: D

Reason :
At Line n1, interface Child is the target type for lambda expression, which means Child must be a Functional Interface. Let's check if Child is a Functional interface or not.

Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].

Interface Parent is a valid interface and it contains a default method, 3 static methods out of which 2 are private and one overriding abstract method [toString()] but it doesn't contain non-overriding abstract method.

Interface Child extends Parent and it declares a non-overriding abstract method play(). As Parent interface doesn't have any non-overriding abstract method, therefore, Child interface has only one non-overriding abstract method and therefore it's a valid Functional interface.

There is no compilation error in the given code.

Lambda expression at Line n1, is the correct implementation of play() method and on execution Line n2 prints PLAYING CRICKET... on to the console.

1.1.36 Answer: E

Reason :
for(;;) is an infinite loop and hence `sb.append("1Z0-815");` causes OutOfMemoryError which is a subclass of Error class.
main(String []) method throws OutOfMemoryError and program terminates abruptly.

1.1.37 Answer: A

Reason :
Module descriptor lives in a file called 'module-info.java' (all characters in lower-case) and by default all the modules implicitly requires java.base module. Providing explicit `requires java.base;` doesn't cause any compilation error.
Hence, correct option is:

```
//module-info.java
module com.training {
}
```

1.1.38 Answer: E

Reason :
Method jump() in Animal class declares to throw RuntimeException. Overriding method may or may not throw any RuntimeException. Only thing to remember is that if overridden method throws any unchecked exception or Error, then overriding method must not throw any checked exceptions. Line n1 compiles successfully as it correctly overrides the jump() method of Animal class.
Class Deer also provides overloaded jump(int) method.

Inside main(String []) method, reference variable 'animal' is of Animal class (supertype) and it refers to an instance of Deer class (subtype), this is polymorphism and allowed in Java.

As instance is of Deer class, hence 'animal' reference can easily be casted to Deer type. Line n2 and Line n3 compiles successfully and on execution prints below on to the console:

```
DEER JUMPS
DEER JUMPS TO 5 FEET
```

1.1.39 Answer: F

Reason :
Let's check all the options one by one:

```
public module agriculture {
}
```

✗ Access and non-access modifiers are not allowed with module declaration .

```
module agriculture {
}

module farming {
}
```

✗ Only one module declaration is allowed in a module descriptor file.

```
module agriculture extends java.se{
}
```

✗ A module doesn't extend/implement another module

```
module agriculture {
    imports java.se;
}
```

✗ 'imports' is not a valid module directive, it should be 'requires'.

```
module agriculture {
    import java.se;
}
```

✗ 'import' is not a valid module directive, it should be 'requires'.

```
import java.lang.*;

module agriculture {
}
```

✓ import statements are allowed in module descriptor file. These import statements are useful when you use 'provides' directive for Services but these are not in the scope of 1Z0-815 exam. For this exam you should know that import statements are allowed in module-info.java file.

1.1.40 Answer: B

Reason :

Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 7 compiles successfully.

No issues with Line 12 as method availableSeats() declares to throw SQLException and main(String []) method code correctly handles it .

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.

1.1.41 Answer: A

Reason :

public void Person() is method and not constructor, as return type is void.

In java it is allowed to use same method name as the class name (though not a recommended practice), so no issues with Person() method declaration.

As there are no constructors available for this class, java compiler adds following constructor:
public Person() {super();}

Person obj = new Person(); invokes the default constructor but it doesn't change the value of name property (by default null is assigned to name property).

System.out.println(obj.name); prints null.

1.1.42 Answer: D

Reason :
list.remove(Object) method returns boolean result but list.remove(int index) returns the removed item from the list, which in this case is of String type and not Boolean type and hence if(list.remove(2)) causes compilation error.

1.1.43 Answer: D

Reason :
Array elements are initialized to their default values.
'arr' is referring to an array of Boolean type, which is reference type and hence both the array elements are initialized to null and therefore in the output null:null is printed.

1.1.44 Answer: B

Reason :
String s1 = new String("Java"); --> Creates 2 objects: 1 String Pool and 1 non-pool. 's1' refers to non-pool object.
String s2 = "JaVa"; --> Creates 1 String pool object and 's2' refers to it.
String s3 = "JaVa"; --> Doesn't create a new object, 's3' refers to same String pool object referred by 's2'.
String s4 = "Java"; --> Doesn't create a new object, s4 refers to String Pool object created at Line n3.
String s5 = "Java"; --> Doesn't create a new object, s5 also refers to String Pool object created at Line n3.

So, at Line n9, 3 String objects are available in the HEAP memory: 2 String pool and 1 non-pool.

1.1.45 Answer: E

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
    (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

If you pass a file name with .java extension to java command, then java command consider the passed file as 'single-file source-code program'. But if any other file name is passed, then it considers the passed file as the class name. So, in this case, if you run the command `java 01_Msg`, then java command searches for the class with the name '01_Msg' and would thrown an error.

```
F:\>java 01_Msg
Error: Could not find or load main class 01_Msg
Caused by: java.lang.ClassNotFoundException: 01_Msg
```

If the file does not have the .java extension, the --source option must be used to force source-file mode .
`java --source 11 01_Msg` instructs the java command to process the 01_Msg as Java 11 source code.

Please note that file name can be any name supported by underlying operating system, hence 01_Msg is a valid file name.
Class names can be any name following Java identifier naming rules:
- It should with a letter, the dollar sign "$", or the underscore character "_".
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- It must not be a java keyword.

Based on above identifier naming rules 01_Msg and 01Msg are invalid identifiers and rest are valid, hence out of 7 options, 5 options can successfully replace /*INSERT*/.

Please note that source-file can contain package statement and multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

1.1.46 Answer: A

Reason :
B obj1 = new A(); => child class reference cannot refer to parent class object. This causes compilation error.
A obj2 = new B(); => parent class reference can refer to child class object. This is Polymorphism.
B obj3 = new B(); => No issues at all.
A obj4 = new A(); => No issues at all.

1.1.47 Answer: C

Reason :
Fields declared with final are constant fields.

1.1.48 Answer: B

Reason :
Initially val = 25.
'if(val++ < 26)' means 'if(25 < 26)', value of val (25) is used in the boolean expression and then value of val is incremented by 1, so val = 26.
25 < 26 is true, control goes inside if-block and executes System.out.println(val++); This prints current value of val to the console, which is 26 and after that increments the value of val by 1, so val becomes 27.

1.1.49 Answer: D

Reason :
case values must evaluate to the same/compatible type as the switch expression can use.
switch expression can accept following:

char or Character
byte or Byte
short or Short
int or Integer
An enum only from Java 6
A String expression only from Java 7

In this case, switch expression [switch (var)] is of byte type.
byte range is from -128 to 127. But in case expression [case 200], 200 is outside byte range and hence compilation error.

1.1.50 Answer: A

Reason :
A module is a set of packages and the module system decides which packages to export and which packages to keep as un-exported.

1.1.51 Answer: A

Reason :
throw ex; causes compilation error as div method doesn't declare to throw Exception (checked) type.

1.1.52 Answer: C

Reason :
ConcurrentModificationException exception may be thrown for following condition:
1. Collection is being iterated using Iterator/ListIterator or by using for-each loop.
And
2. Execution of Iterator.next(), Iterator.remove(), ListIterator.previous(), ListIterator.set(E) & ListIterator.add(E) methods. These methods may throw java.util.ConcurrentModificationException in case Collection had been modified by means other than the iterator itself, such as Collection.add(E) or Collection.remove(Object) or List.remove(int) etc.

PLEASE NOTE: for-each loop internally implements Iterator and invokes hasNext() and next() methods.

For the given code, 'dryFruits' list is being iterated using for-each

loop (internally as an Iterator).
hasNext() method of Iterator has following implementation:

```
public boolean hasNext() {
    return cursor != size;
}
```

Where cursor is the index of next element to return and initially it is 0.

1st Iteration: cursor = 0, size = 4, hasNext() returns true.
iterator.next() increments the cursor by 1 and returns "Walnut".
2nd Iteration: cursor = 1, size = 4, hasNext() returns true.
iterator.next() increments the cursor by 1 and returns "Apricot". As "Apricot" starts with "A", hence dryFruits.remove(dryFruit) removes "Apricot" from the list and hence reducing the list's size by 1, size becomes 3.
3rd Iteration: cursor = 2, size = 3, hasNext() returns true.
iterator.next() method throws java.util.ConcurrentModificationException.

If you want to successfully remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove(...) method.
Using List.remove(...) method while iterating the list (using the Iterator/ListIterator or for-each) may throw java.util.ConcurrentModificationException.

1.1.53 Answer: H

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,

```
var x = "Java"; //x infers to String
var m = 10; //m infers to int
```

The identifier var is not a keyword, hence var can still be used as variable name, method name, package name or loop's label but it cannot be used as a class or interface name.
At Line n1, variable 'i' infers to int type, so no issues at Line n1.

if-else block uses break; and continue; statements. break; will exit the loop and will take the control to Line n3 on the other hand continue; will take the control to Line n1. In both the cases Line n2 will never be executed.
As Compiler knows about it, hence it tags Line n2 as unreachable, which causes compilation error.

1.1.54 Answer: C

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

Hence, there is no issue with Line n1, variable 'var' is of int type and it stores value 3.

Line n2:
String [][] arr = new String[--var][var++]; //var = 3
Access array element operator [] is left to right associative .
=> String [][] arr = new String[2][var++]; //var = 2, var is decremented first and then used in the expression.
=> String [][] arr = new String[2][2]; //var = 3, value of var is used first and then it is incremented by 1

Hence, arr refers to 2-dimensional String array object {{null, null}, {null, null}}.

At Line n3, arr[1][1] = "X"; assigns "X" to element at index [1][1], therefore arr --> {{null, null}, {null, "X"}}

At Line n4, arr[1][2] = "Y"; causes ArrayIndexOutOfBoundsException as 2nd index 2 is out of range.

As Line n4 throws Exception at runtime, hence for loop will not be executed.

1.1.55 Answer: C

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
   (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence, `java Test.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Test.java
java -cp <memory> Test3
```

Hence in this case, output is: Test3.
Please note that source-file can contain multiple classes (even public) but first class found must have special main method 'public static void main(String [])'. Also note that file name can be any name supported by underlying operating system, such as if you rename 'Test.java' to '123.java' .
java 123.java will be equivalent to:

```
javac -d <memory> 123.java
java -cp <memory> Test3
```

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

1.1.56 Answer: A

Reason :
Java compiler adds super(); as the first statement inside constructor,

if call to another constructor using this(...) or super(...) is not available.
Compiler adds super(); as the first line in Word's constructor: Word(int pages, String type) { super(); } but Document class doesn't have a no-argument constructor and that is why Word's constructor `Word(int pages, String type)` causes compilation error.

As the first statement inside Word(int pages, String type){} constructor, you can either have super(pages); or this(type) but not both.
Also note that Constructor call using this(...) or super(...) must be the first statement inside the constructor. Based on these observations, you should replace //TODO with:
this(type);
super.pages = pages;

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling "

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String

composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Based on above points, `System.out.println(String.join(",", obj.type, obj.pages + ""));` will print TEXT,25 on to the console.

1.1.57 Answer: A

Reason :
As "" is empty string, hence nothing is appended to the StringBuilder instance and length() method returns 0.

1.1.58 Answer: E

Reason :
Given statement:
System.out.println(status = false || status = true | status = false);
As it contains multiple operators, hence let's group the operators first.
System.out.println(status = false || status = (true | status) = false);
//Bitwise inclusive OR | has highest precedence over logical or || and assignment =
For assignment operator to work, left operand must be variable but in above case, `(true | status) = false` causes compilation failure as left operand (true | status) evaluates to a boolean value and not boolean

variable.

1.1.59 Answer: D

Reason :
Super class defines a method with name Super() but not any constructor. Hence compiler adds below default constructor in Super class:

```
Super() {
    super();
}
```

Class Super extends from Object class and Object class has no-argument constructor, which is called by the super(); statement in above default constructor.

Java compiler also adds `super();` as the first statement inside the no-argument constructor of Base class:

```
Base() {
    super();
    Super();
    System.out.print("GOING_");
}
```

As Base extends Super and both the classes are in the same package, hence `super();` invokes the no-argument constructor of Super class and `Super();` invokes the Super() method of Super class. Base class inherits the Super() method of Super class.

No compilation error in any of the classes.

On executing Test class, main(String[]) is invoked, which executes `new Base();` statement.
No-argument constructor of Base class is invoked, which executes `super();`, hence no-argument constructor of Super class is invoked.
Next, `Super();` is executed and this invokes the Super() method of Super class and hence KEEP_ is printed on to the console.
After that, `System.out.print("GOING_");` is executed and GOING_ is printed on to the console.

main(String []) method finishes its execution and program terminates successfully after printing KEEP_GOING_ on to the console.

1.1.60 Answer: E

Reason :
Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);

String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Based on above points Line n1 compiles successfully .

As String class extends Object class, hence instanceof check at Line n2 compiles successfully.

Syntax at Line n3 is: str += (String) obj; => str = str + (String) obj;
No issues with Line n3 as well.

Super class reference variable can easily refer to an instance of sub class and hence syntax of Line n5 and Line n6 is also valid.

Which overloaded method to invoke, is decided at compile time. As print(...) method is overloaded in Test class, print(obj1, obj2) tags to print(Object...) because obj1 and obj2 are of Object type.

Given code compiles successfully and on execution prints SPORTMAD on to the console.

1.1.61 Answer: C

Reason :
Encapsulation is all about having private instance variable and providing public getter and setter methods.

1.1.62 Answer: D,E

Reason :
Let's check all the statements one by one:

s1.equals(s2): equals(String) method of String class matches two String objects and it takes character's case into account while matching. Alphabet A in upper case and alphabet a in lower case are not equal according to this method. As String objects referred by s1 and s2 have different cases, hence output is false.

s1.equals(s2.toUpper()): Compilation error as there is no toUpper() method available in String class. Correct method name is: toUpperCase().

s2.equals(s1.toLower()): Compilation error as there is no toLower() method available in String class. Correct method name is: toLowerCase() .

s1.length() == s2.length(): length() method returns the number of characters in the String object. s1.length() returns 3 and s2.length() also returns 3, hence output is true.

s1.equalsIgnoreCase(s2): Compares s1 and s2, ignoring case consideration and hence returns true.

s1.contentEquals(s2): String class contains two methods: contentEquals(StringBuffer) and contentEquals(CharSequence). Please note that String, StringBuilder and StringBuffer classes implement CharSequence interface, hence contentEquals(CharSequence) method defined in String class cab be invoked with the argument of either String or StringBuilder or StringBuffer. In this case, it is invoked with String argument and hence it is comparing the contents of two String objects. This method also takes character's case into account while matching. As String objects referred by s1 and s2 have different cases, hence output is false.

1.1.63 Answer: B

Reason :
Special main method (called by JVM on execution) should be static and should have public access modifier. It also takes argument of String [] type (Varargs syntax String... can also be used).
String [] or String... argument can use any identifier name, even though in most of the cases you will see "args" is used.

Both the classes (Guest and Message) contain special main method .
No compilation error with the code: file is correctly named as Guest.java (name of public class).
java Guest Arya Stark passes new String [] {"Arya", "Stark"} to args of Guest.main method. Apart from being special main method, Message.main is static method, so Guest.main method invokes Message.main method with the same argument: new String [] {"Arya", "Stark"}. args[1] is "Stark" hence "Welcome! Stark" gets

printed on to the console.

1.1.64 Answer: C

Reason :
Object created at Line n1 becomes eligible for Garbage collection after Line n1 only, as there are no references to it. So We have one object marked for GC.
Line n2 creates another NewsPaper object .
Line n3 invokes change(NewsPaper) method. Parameter variable 'np' refers to the same NewsPaper object created at Line n2.
Line n6 creates a new NewsPaper object and variable 'np' refers to it.
Object created at Line n6 becomes unreachable after change(NewsPaper) method pops out of the STACK, and this happens after Line n3.
So at Line n4, we have two NewsPaper objects eligible for Garbage collection: Created at Line n1 and Created at Line n6.

1.1.65 Answer: B

Reason :
Eclipse class is defined under 'com.it.tools.ide' package. Though it is the best practice to keep the source code files under package directory structure, e.g. 'C:\source\tools\com\it\tools\ide\Eclipse.java' is the logical path for Eclipse.java file but it can legally be placed at any path. Hence, 'C:\source\tools\com\it\tools\Eclipse.java' ('ide' is missing) is also allowed. There is no compilation error for Eclipse.java file.
Please note that generated class files must be under proper directory structure.

Package 'com.it.tools' contains Tools class and package 'com.it.tools.ide' contains Eclipse class.
But package 'com.it' is empty.

In the module descriptor file, if package specified in the exports directive is empty or doesn't exist, then compiler complains about it. As the package 'com.it' is empty, hence Line n1 causes compilation error.

1.1.66 Answer: D

Reason :
Given code compiles successfully.
arr refers to an array object of size 0, this means arr stores some memory address. So we will not get NullPointerException in this case.
But index 0 is not available for an array object of size 0 and thus ArrayIndexOutOfBoundsException is thrown at runtime.

1.1.67 Answer: A

Reason :
There is a single module and by looking at the module-info.java file, you can say that name of the module is 'calc'. But name of the module directory (under which module-info.java resides) is: 'calculator'.

To compile the module in multi-module mode, name of the module directory must match with the name of the module and because in given case, name of module and module directory are different, hence multi-module command cannot be used here. Therefore, below 3 compilation commands cause error:
javac -d out --module-source-path src --module calculator
javac -d out --module-source-path src --module calc
javac -d out --module-source-path src --module calc-module

Let's check single-mode commands:
javac -d out\calc-module src\calculator\com\udayankhattry\ocp1\Calculator.java src\calculator\module-info.java => This will successfully compile the module code and place the generated class files under 'C:\out\calc-module\' directory.
Generated file structure after successful execution of above command will be:

```
C:
├────src
│    └────calculator
│         │ module-info.java
│         │
│         └────com
│              └────udayankhattry
│
```

```
│               └────ocp1
│                       Calculator.java
│
└────out
    └────calc-module
        │   module-info.class
        │
        └────com
            └────udayankhattry
                └────ocp1
                        Calculator.class
```

javac -d out\calc-module src\calculator\com\udayankhattry\ocp1\Calculator.java => This will not compile module-info.java file. Hence not correct .

javac -d out\calc-module src\calculator\module-info.java => This will not compile Calculator.java file. Hence not correct.

Please note: Even for single-module, it is always a good practice to keep the module name and module directory name same, so that to compile the code multi-module command can be used.

So, out of the given 6 options, only 1 option is left, which will successfully compile and execute the given module code:
javac -d out\calc-module src\calculator\com\udayankhattry\ocp1\Calculator.java src\calculator\module-info.java
java --module-path out --module calc/com.udayankhattry.ocp1.Calculator

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
    (to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

Hence below command are invalid:
java -p out -m calculator/com.udayankhattry.ocp1.Calculator [Module name is not 'calculator']
java -p out -m calc-module/com.udayankhattry.ocp1.Calculator [Module name is not 'calc-module']

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries.
For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar' .

Hence for execution below commands will also work:
java -p out -m calc/com.udayankhattry.ocp1.Calculator [Module is available in 'out' directory]
java -p out\calc-module -m calc/com.udayankhattry.ocp1.Calculator [Single exploded module is inside 'out\calc-module']

1.1.68 Answer: B

Reason :
findSecretFile() method declares to throw FileNotFoundException (checked exception) and main(String []) method invokes findSecretFile(); method.
catch-handler in this case can specify FileNotFoundException or IOException or Exception or Throwable. As catch-handler for Throwable is available, hence exception is handled correctly inside main(String []) method.

As, return; statement is available inside catch-block only, therefore compiler doesn't tag `System.out.println("DONE");` as unreachable. If return; statement was present in both try and catch block or finally-block, then unreachable code compilation error would have caused by the last statement in main(String []) method.

Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor
2. public Throwable(String message) {...} : Pass the detail message
3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause
4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)
1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)
2. public String toString() {} :
Returns a short description of this throwable. The result is the concatenation of :
the name of the class of this object
": " (a colon and a space)
the result of invoking this object's getLocalizedMessage() method

If getLocalizedMessage returns null, then just the class name is returned.

On execution, main(String []) method invokes findSecretFile(), which throws an instance of FileNotFoundException.
There is a matching catch-handler available in main(String []) method, therefore `System.out.println(ex.getMessage());` is executed and this prints ACCESS REQUIRED on to the console.
As there is finally-block available, so just before the return; statement takes control out of main(String []) method, code inside finally-block gets executed and this prints LEVEL 1 on to the console.
Control exits the main(String []) method and program terminates successfully.

1.1.69 Answer: A,D

Reason :
If you check the directory structure, you will find that directory 'orders' contains 'items', but 'orders' and 'orders.items' are different packages.
import orders.*; will only import all the classes in 'orders' package but not in 'orders.items' package.

You need to import Order and Item classes.

To import Order class, use either import orders.Order; OR import orders.*; and to import Item class, use either import orders.items.Item; OR import orders.items.*;

1.1.70 Answer: E

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to in t

Given statement:
var festivals = List.of("DIWALI", "CHRISTMAS", "EID"); => festivals refers to a List of String type.

Static List.of() overloaded methods were added in Java 9 and these return an unmodifiable list containing passed elements. So, above list object referred by festivals is unmodifiable.

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.
Interface java.util.function.Predicate<T> declares below non-overriding abstract method:
boolean test(T t);

Lambda expression passed to removeIf(Predicate) method is: `str -> str.length() == 3` and is a valid lambda expression for

Predicate<String> interface.

There is no compilation error in the code but calling removeIf(Predicate) method on unmodifiable list 'festivals' throws an exception(java.lang.UnsupportedOperationException) at runtime.

1.1.71 Answer: A,D

Reason :
Easy question on iterating through 2-dimensional array. Starting index should be 0 and not 1. As for loops contain one statement, hence curly brackets can be ignored.
for-each loop's syntax is correct.

1.1.72 Answer: C

Reason :
Basic/Regular for loop has following form:
for ([ForInit] ; [Expression] ; [ForUpdate]) {...}
[ForInit] can be local variable initialization or the following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpression
PostDecrementExpressio n
MethodInvocation
ClassInstanceCreationExpression

[ForUpdate] can be following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpression
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for [ForInit] and [ForUpdate], `System.out.print(i++);` is used, which is a method invocation statement and hence a valid statement. Given code compiles fine.

Let's check the iterations:
1st iteration: [ForInit] expression is executed, 0 is printed on to the console. i = 1. i < 2 evaluates to true, control goes inside the loop's body and execute `System.out.print(i);` statement. 1 is printed on to the console.
2nd iteration: [ForUpdate] expression is executed, 1 is printed on to the console. i = 2. 2 < 2 evaluates to false, control goes out of the for loop. main method ends and program terminates successfully after printing 011 on to the console.

1.1.73 Answer: E

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to in t

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name. But var type cannot be used as method parameters or method return type. Hence 'operate(var v1, var v2)' causes compilation error.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

1.1.74 Answer: B

Reason :
To know more about join and split methods of String class, please check the URLs:
https://udayankhattry.com/join-string/
https://udayankhattry.com/split-string/

Let's solve the given expression:
Variable 'str' refers to "USER GENERATED".
str = String.join("-", str.split(" "));
str = String.join("-", "USER GENERATED".split(" "));
str = String.join("-", ["USER","GENERATED"]); //Splits on the basis of single space " ".
str = "USER-GENERATED";

`System.out.println(str);` prints USER-GENERATED on to the console.

1.1.75 Answer: B

Reason :
speed method is correctly overloaded in Car class as both the methods have different signature: speed(Byte) and speed(byte...). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Car().speed(b);` tags to speed(Byte) as boxing is preferred over variable arguments. Code as is prints DARK on to the console.

Variable arguments syntax '...' can be used only for method parameters and not for variable type and type-casting. Hence the option of replacing Line n4 and Line n5 are not correct .

If you delete speed(Byte) method, i.e. Line n1, Line n2 and Line n3, then `new Car().speed(b);` would tag to speed(byte...) method and on execution would print LIGHT on to the console.

1.1.76 Answer: C

Reason :
Even though char is compatible with int type but '3' is not equal to 3.
int day = 3; => SORRY!!! NO SALE
Integer day = 3; => day is of Integer type and case contains char '3'. char '3' cannot be compared with Integer and hence compilation error. case '3' can easily be compared with int value but not with Integer type.
int day = '3'; => BUY 2 GET 1 FREE

HINT: There is no need to remember. case '3' value means you are trying to equate or compare day (Integer value) with '3' (char). If assignment operation works then method invocation, switch expression parameter etc. will also work. Integer day = 3; is possible but Integer day = '3'; causes compilation error as char cannot be converted to Integer.

1.1.77 Answer: C

Reason :
There is no compilation error in the given code.
At Line n1, an instance of Counter is created and is referred by variable 'ctr'. ctr.count = 0.

Line n2 just defines the lambda expression, no changes are made to variable 'count' at Line n1.

At Line n3, ctr.count = 100;

At Line n4, code of Lambda expression of Line n2 is executed and it changes ctr.count to 20. ctr.count = 20.

Line n5 prints 20 on to the console.

1.1.78 Answer: B

Reason :
Syntax of module-info.java file is valid, exports directive is also valid. No compilation error in this file.

Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this case, module source path is 'src' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'out' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
It is not used in this case as given module is not dependent upon other compiled/packaged application modules.

Given javac command:
javac -d dir2 --module-source-path dir1 -m shoppin g

It creates module-directory 'shopping' under 'dir2' and then places the generated class file under 'shopping' directory. Hence, correct path of module-info.class will be: C:\dir2\shopping\module-info.class.
It also places other classes based on the package statement in respective source code files.

After compilation, you will get below directory/file structure:

```
C:
+---dir1
|   \---shopping
|       |   module-info.java
|       |
|       \---com
|           \---shopping
|               \---online
|                       Product.java
|
\---dir2
    \---shopping
        |   module-info.class
```

```
|
\---com
    \---shopping
        \---online
                Product.class
```

1.1.79 Answer: D

Reason :

In below statements: student<main> means student inside main method.

On execution of main method: student<main> --> {"James", 25}, marks<main> = 25.

On execution of review method: stud<review> --> {"James", 25} (same object referred by student<main>), marks<review> = 25 (this marks is different from the marks defined in main method).

marks<review> = 35 and stud.marks = 60. So at the end of review method: stud<review> --> {"James", 60}, marks<review> = 35.

Control goes back to main method: student<main> --> {"James", 60}, marks<main> = 25. Changes done to reference variable are visible in main method but changes done to primitive variable are not reflected in main method.

1.1.80 Answer: A

Reason :

Let' suppose:

At Line n1, ArrayList object is stored at [15EE00].

At Line n2, Counter object is stored at [25AF06].

original contains memory address of above counter object. [15EE00] --> {25AF06}

Now original.clone() will create a new array list object, suppose at [45BA12] and then it will copy the contents of the ArrayList object stored at [15EE00].

So, cloned contains memory address of the same counter object. [45BA12] --> {25AF06}

In this case, original != cloned, but original.get(0) == cloned.get(0). This means both the array lists are created at different memory location but refer to same Counter object.

cloned.get(0) returns counter object stored at [25AF06] and .count = 5 means [25AF06] refers to [Counter object (5)].

System.out.println(original); Prints the element of ArrayList original, which is: {25AF06} and toString() method prints: Counter-5 as Counter object referred by [25AF06] is [Counter object (5)].

2 Practice Test-2

2.1 80 Questions covering all topics.

2.1.1 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.function.Predicate;

class Employee {
    private String name ;
    private int age ;
    private double salary ;

    public Employee(String name, int age, double salary) {
        this . name = name;
        this . age = age;
        this . salary = salary;
    }

    public String getName() {
        return name ;
    }

    public int getAge() {
        return age ;
    }

    public double getSalary() {
        return salary ;
    }

    public String toString() {
        return name ;
    }
}

public class Test {
    public static void main(String [] args) {
        List<Employee> list = new ArrayList<>();
        list.add( new Employee( "James" , 25 , 15000 ));
        list.add( new Employee( "Lucy" , 23 , 12000 ));
        list.add( new Employee( "Bill" , 27 , 10000 ));
        list.add( new Employee( "Jack" , 19 , 5000 ));
        list.add( new Employee( "Liya" , 20 , 8000 ));

          process (list, /*INSERT*/ );

          System. out .println(list);
    }

    private static void process(List<Employee> list,
                         Predicate<Employee> predicate) {
        Iterator<Employee> iterator = list.iterator();
        while (iterator.hasNext()) {
           if (predicate.test(iterator.next()))
              iterator.remove();
        }
    }
}
```

Which of the following lambda expressions, if used to replace /*INSERT*/, prints [Jack, Liya] on to the console?

A	. (Employee e) -> { return e.getSalary() >= 10000 ; }
B	. (e) -> { e.getSalary() >= 10000 ; }
C	. e -> { e.getSalary() >= 10000 }
D	. e -> e.getSalary() >= 10000
E	. e - > e.getSalary() >= 10000

2.1.2 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
        switch ( "HELLO" ) {
            case "HELLO" :
                System. out .println( 1 );
            default :
                System. out .println( 2 );
            case "null" :
                System. out .println( 3 );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. 1

C. 2

D. 12

E. 23

F. 123

2.1.3 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        boolean flag1 = true ;
        boolean flag2 = false ;
        boolean flag3 = true ;
        boolean flag4 = false ;

          System. out .println(!flag1 == flag2 !=
              flag3 == !flag4); //Line n1
        System. out .println(flag1 = flag2 !=
              flag3 == ! flag4); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A.	Line n1 causes compilation error	B.	Line n2 causes compilation error
C.	true true	D.	true false
E.	false true	F.	false false

2.1.4 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;


public class Test {
    public static void main(String[] args) {
        List<String> sports = new ArrayList<>();
        sports.add( "Windsurfing" );
        sports.add( "Aerobics" );
        sports.add( "Archery" );
        sports.add( "Diving" );

           Iterator<String> iterator = sports.iterator();
         while (iterator.hasNext()) {
            String sport = iterator.next();
             if (sport.startsWith( "A" )) {
                sports.remove(sport);
            }
        }

           System. out .println(sports);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Windsurfing, Aerobics, Archery, Diving]

B. [Windsurfing, Diving]

C. An exception is thrown at runtime

D. Compilation error

2.1.5 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        System. out .println(sb.append( null ).length());
    }
}
```

What will be the result of compiling and executing Test class?

A. 1

B. 4

C. Compilation error

D. An exception is thrown at runtime

2.1.6 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

interface Sellable {}
abstract class Animal {}
class Mammal extends Animal{}
class Rabbit extends Mammal implements Sellable{}

public class Test {
    {
        List<Animal> list = new ArrayList<>();
        list.add( new Rabbit());
    }
    {
        List<Animal> list = new ArrayList<>();
        list.add( new Mammal());
    }
    {
        List<Mammal> list = new ArrayList<>();
        list.add( new Rabbit());
    }
    {
        List<Sellable> list = new ArrayList<>();
        list.add( new Mammal());
    }
    {
        List<Sellable> list = new ArrayList<>();
        list.add( new Rabbit());
    }
}
```

Which of the following statements is true?

A. Only one initializer block causes compilation error

B. Two initializer blocks cause compilation error

C. Three initializer blocks cause compilation error

D. Four initializer blocks cause compilation error

E. Five initializer blocks cause compilation error

2.1.7 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class MyClass {
    MyClass() {
        System. out .println( 101 );
    }
}

class MySubClass extends MyClass {
     final MySubClass() {
        System. out .println( 202 );
    }
}
public class Test {
    public static void main(String[] args) {
        System. out .println( new MySubClass());
    }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation error
B.	101 202 <Some text containing @ symbol>
C.	202 <Some text containing @ symbol>
D.	202 101 <Some text containing @ symbol>
E.	101 <Some text containing @ symbol>

2.1.8 Consider codes of 3 java files:

```
//Planet.java
package com.udayankhattry.galaxy;

public class Planet {
    String name ;
    public Planet(String name) {
        this . name = name;
    }

    public String toString() {
        return "Planet: " + name ;
    }
}


//Creator.java
```

```
package com.udayankhattry.ocp1;

public class Creator {
    public static Planet create() {
        return new Planet( "Earth" );
    }
}

//TestCreator.java
package com.udayankhattry.ocp1.test;

public class TestCreator {
    public static void main(String[] args) {
        System. out .println(Creator.create());
    }
}
```

And below options:

1.
Add below import statement in Creator.java file:

```
import com.udayankhattry.galaxy.Planet;
```

2.
Add below import statement in Creator.java file:

```
import com.udayankhattry.ocp1.test.TestCreator
```

3.
Add below import statement in TestCreator.java file:

```
import com.udayankhattry.ocp1.Creator;
```

4.
Add below import statement in TestCreator.java file:

```
import com.udayankhattry.galaxy.Planet;
```

Which of the above options needs to be done so that on executing TestCreator class, "Planet: Earth" is printed on to the console? Please note: Unnecessary imports are not allowed.

A. Only 1

B. Only 2

C. Only 3

D. Only 4

E. 1 & 2 only

F. 3 & 4 only

G. 1 & 3 only

H. 1, 3 & 4 only

I. 1, 2, 3 & 4 are needed

2.1.9 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       Boolean [] arr = new Boolean[ 2 ];
       List<Boolean> list = new ArrayList<>();
       list.add(arr[ 0 ]);
       list.add(arr[ 1 ]);

          if (list.remove( 0 )) {
          list.remove( 1 );
       }

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. ArrayIndexOutOfBoundsException is thrown at runtime

C. NullPointerException is thrown at runtime

D. [true]

E. [false]

F. []

2.1.10 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Display {
    void disp(String s);
}

public class Test {
    public static void main(String[] args) {
       method ( /*INSERT*/ );
    }

     private static void method(Display obj, String text) {
       obj.disp(text);
    }
}
```

Which of the following options successfully replace /*INSERT*/ such that on execution, LAMBDA is printed on to the console?

A.	s -> { System. *out* .println(s.toUpperCase()) }, "lambda"
B.	s -> System. *out* .println(s.toUpperCase()), "lambda"
C.	s -> s.toUpperCase(), "lambda"
D.	s -> System. *out* .println(s.toUpperCase())

2.1.11 Range of short data type is from -32768 to 32767

Which of the following code segments, written inside main method will compile successfully?
Select ALL that apply.

A.	short *s1* = 10 ;	B.	short *s2* = 32768 ;
C.	final int *i3* = 10 ; short *s3* = *i3* ;	D.	final int *i4* = 40000 ; short *s4* = *i4* ;
E.	final int *i5* = 10 ; short *s5* = *i5* + 100 ;	F.	final int *m* = 25000 ; final int *n* = 25000 ; short *s6* = *m* + *n* ;
G.	int *i7* = 10 ; short *s7* = *i7* ;		

2.1.12 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int start = 1 ;
        int sum = 0 ;
        do {
            if (start % 2 == 0 ) {
                continue ;
            }
            sum += start; //Line n1
        } while (++start <= 10 );
       System. out .println(sum);
    }
}
```

What will be the result of compiling and executing Test class?

A. 25

B. 55

C. Compilation error

D. 24

2.1.13 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String [] arr = new String[ 1 ];
       System. out .println(arr[ 0 ].isBlank());
    }
}
```

What will be the result of compiling and executing Test class?

A. true

B. false

C. An exception is thrown at runtime

D. Compilation error

2.1.14 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static String s ;
    public static void main(String[] args) {
        try {
            System. out .println( s .length());
        } catch (NullPointerException |
                RuntimeException ex) {
            System. out .println( "DONE" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. DONE

B. Executes successfully but no output

C. Compilation error

D. None of the other options

2.1.15 Consider below codes of 4 java files:

```
//Moveable.java
package com.udayankhattry.ocp1;

public interface Moveable {
    void move();
}


//Animal.java
package com.udayankhattry.ocp1;

public abstract class Animal {
    void move() {
        System. out .println( "ANIMAL MOVING" );
    }
}
```

```
//Dog.java
package com.udayankhattry.ocp1;

public class Dog extends Animal implements Moveable {}
```

```
//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        Moveable moveable = new Dog();
        moveable.move();
    }
}
```

Which of the following statements is correct?

A. There is a compilation error in Animal.java file

B. There is a compilation error in Dog.java file

C. There is a compilation error in Test.java file

D. There is no compilation error and on execution, Test class prints ANIMAL MOVING on to the console

2.1.16 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    static String str = "KEEP IT " ; //Line n1
    public static void main(String[] args) {
        String str = str + "SIMPLE" ; //Line n2
         System. out .println(str);
    }
}
```

What will be the result of compiling and executing Test class?

A. KEEP IT

B. KEEP IT SIMPLE

C. SIMPLE

D. Compilation error

2.1.17 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

boolean test(T t);

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
       List<String> list = new ArrayList<>(
              List. of ( "A" , "E" , "I" , "O" , "U" ));
        if ( verify (list, l -> l.remove( "I" ))) //Line n1
            if ( verify (list, l -> l.add( "I" ))){} //Line n2

          System. out .println(list);
    }

    private static boolean verify(List<String> list,
              Predicate<List<String>> predicate) {
        return predicate.test(list); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. [A, E, I, O, U]

E. [A, E, O, U]

F. [A, E, O, U, I]

2.1.18 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add( "C" );
        list.add( "Z" );
        list.add( "A" );
        list.add( "R" );
        list.subList( 1 , 2 ).clear();
        System. out .println(String. join ( "" , list));
    }
}
```

What will be the result of compiling and executing Test class?

A. CZAR

B. CAR

C. CR

D. Compilation error

E. An exception is thrown at runtime

2.1.19 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String [] ingredients = { "Lettuce" , "Avocado" ,
              "Zucchini" , "Chickpea" };
         for (String ingredient : ingredients) {
            System. out .print(ingredient + " " );
             if (ingredient.equals( "Zucchini" )) {
                continue ;
            }
            System. out .println( "salad!" );
             break ;
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Lettuce salad!
B. Lettuce Avocado salad!
C. Lettuce Avocado Zucchini salad!
D. Lettuce Avocado Chickpea salad!
E. None of the other options

2.1.20 When does a class get the default constructor?

A. If you define parameterized constructor for the class
B. You have to define at least one constructor to get the default constructor
C. If the class does not define any constructors explicitly
D. All classes in Java get a default constructor

2.1.21 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        try {
            main (args);
        } catch (Exception ex) {
            System. out .println( "CATCH-" ); //Line n1
        }
        System. out .println( "OUT" ); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. CATCH-OUT is printed and program terminates successfully

B. OUT is printed and program terminates successfully

C. System.out.println statements at Line n1 and Line n2 are not executed and program ends abruptly

D. Compilation error

2.1.22 Given code of Test.java file:

```
class A {
    public static void main(String [] args) {
      System. out .println( "A" );
    }
}

class B {
    public static void main(String [] args) {
      System. out .println( "B" );
    }
}

class C {
    public static void main(String [] args) {
      System. out .println( "C" );
    }
}

class D {
    public static void main(String [] args) {
      System. out .println( "D" );
    }
}
```

Which of the following options is correct?

A.	To print C on to the console, execute below commands: javac Test.java java Test
B.	To print C on to the console, execute below commands:

	javac C.java java C
C.	To print C on to the console, execute below commands: javac Test.java java C
D.	Test.java file is not a valid java file as it doesn't contain code for class Test
E.	Test.java file will compile successfully but expected output is not possible

2.1.23 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Base {
  public void log() throws NullPointerException {
    System. out .println( "Base: log()" );
  }
}

class Derived extends Base {
  public void log() throws RuntimeException {
    System. out .println( "Derived: log()" );
  }
}

public class Test {
  public static void main(String[] args) {
    Base obj = new Derived();
    obj.log();
  }
}
```

What will be the result of compiling and executing Test class?

A. Base: log()

B. Derived: log()

C. Compilation error in Derived class

D. Compilation error in Test class

2.1.24 On Windows platform below directories/files are available:

1.
C:\codes\printarguments\module-info.java:

```
module printarguments{
}
```

2.
C:\codes\printarguments\com\udayankhattry\ocp1\Test.java:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String... args) {
      System. out .println(String. join ( "-" , args));
    }
}
```

3.
C:\mods\

After compilation below directory/file structure is generated:

```
C:
+---codes
|   \---printarguments
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                       Test.java
|
\---mods
    \---printarguments
        |   module-info.class
        |
        \---com
            \---udayankhattry
                \---ocp1
                        Test.class
```

And the commands executed from C:\

1. java -p mods -m printarguments POWER DRIVEN

2. java -p mods -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN

3. java -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN -p mods

4. java --module-source-path src -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN

5. java --module-source-path mods -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN

6. java --module-path mods --module printarguments POWER DRIVEN

How many of the above commands print POWER-DRIVEN on to the console?

A. Only one

B. Two commands

C. Three commands

D. More than three commands

2.1.25 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;
import java.io.IOException;

public class Test {
    public static void main(String[] args) {
        try {
            find ();
        } catch (Exception ex) {
            System. out .println(ex.getMessage());
        }
    }
    static void find() throws Exception {
      try {
        System. out .print( 1 );
         throw new FileNotFoundException( "FNF" );
      } catch (FileNotFoundException ex) {
        System. out .print( 2 );
         throw new IOException( "IO" );
      } catch (IOException ex) {
        System. out .print( 3 );
         throw new Exception( "EXP" );
      } finally {
        System. out .print( 4 );
         throw new Exception( "FINALLY" );
      }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. 1234FINALLY

C. 124FINALLY

D. 14FINALLY

E. 12IO

F. 124IO

G. 14FNF

2.1.26 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    char c ;
    double d ;
    float f ;

    public static void main(String[] args) {
      Test obj = new Test();
      System. out .println( ">" + obj. c );
```

```
        System. out .println( ">" + obj. d );
        System. out .println( ">" + obj. f );
    }
}
```

What will be the result of compiling and executing Test class?

A.	>null >0.0 >0.0	B.	> >0.0 >0.0
C.	> >0.0 >0.0f	D.	>null >0.0 >0.0f

2.1.27 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var arr = new Double[ 2 ];
        System. out .println(arr[ 0 ] + arr[ 1 ]);
    }
}
```

What will be the result of compiling and executing Test class?

A. An exception is thrown at runtime

B. 0.0

C. 0

D. 0.00

E. Compilation error

2.1.28 Consider below code of Circle. java file:

```
package com.udayankhattry.ocp1;

public class Circle {
    private double radius ;

    public Circle( double radius) {
        this . radius = radius;
    }

    public double getArea() {
        return Math. PI * radius * radius ;
    }
}
```

User must be allowed to read and change the value of radius field. What needs to be done so that all the classes can read/change the value of radius field and Circle class is well encapsulated as well?

A.	Nothing needs to be done
B.	Change radius declaration from `private double radius;` to `double radius;`
C.	Change radius declaration from `private double radius;` to `protected double radius;`
D.	Change radius declaration from `private double radius;` to `public double radius;`

E.	Add below 2 methods in Circle class:
	public double getRadius() {
	    return radius ;
	}
	public void setRadius(**double** radius) {
	    this . **radius** = radius;
	}

F.	Add below 2 methods in Circle class:
	protected double getRadius() {
	    return radius ;
	}
	protected void setRadius(**double** radius) {
	    this . **radius** = radius;
	}

2.1.29 Consider three independent modules with the same name 'messages' in three different directories:

```
C:
├──────out1
│   └──────messages
│        │ module-info.class
│        │
│        └──────com
│            └──────udayankhattry
│                └──────ocp1
│                      Main.class
│
├──────out2
│   └──────messages
│        │ module-info.class
│        │
│        └──────com
│            └──────udayankhattry
│                └──────ocp1
│                      Main.class
│
└──────out3
    └──────messages
         │ module-info.class
         │
         └──────com
             └──────udayankhattry
                 └──────ocp1
                       Main.class
```

main(String...) method of Main class defined in 'out1' contains:

```
System. out .println( "KEEP IT SIMPLE" );
```

main(String...) method of Main class defined in 'out2' contains:

```
System. out .println( "NEVER GIVE UP" );
```

main(String...) method of Main class defined in 'out3' contains:

```
System. out .println( "YES YOU CAN" );
```

And the command:

```
java -p out3;out2;out1\messages -m
messages/com.udayankhattry.ocp1.Main
```

What is the result?

A. It causes error because of the duplicate modules found

B. KEEP IT SIMPLE

C. NEVER GIVE UP

D. YES YOU CAN

2.1.30 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String fruit = "mango" ;
        switch (fruit) {
           case "Apple" :
              System. out .println( "APPLE" );
           case "Mango" :
              System. out .println( "MANGO" );
           case "Banana" :
              System. out .println( "BANANA" );
               break ;
           default :
              System. out .println( "ANY FRUIT WILL DO" );
       }
    }
}
```

What will be the result of compiling and executing Test class?

A.	MANGO	B.	ANY FRUIT WILL DO
C.	MANGO BANANA	D.	MANGO ANY FRUIT WILL DO

E.	MANGO BANANA ANY FRUIT WILL DO		

2.1.31 Consider the code of TestStudent.java file:

```
package com.udayankhattry.ocp1;

class Student {
    String name ;
     int age ;

      void Student() {
        Student( "James" , 25 ); //Line n1
     }

      void Student(String name, int age) {
         this . name = name;
         this . age = age;
    }
}

public class TestStudent {
     public static void main(String[] args) {
        Student s = new Student(); //Line n2
         System. out .println(s. name + ":" + s. age ); //Line n3
     }
}
```

What will be the result of compiling and executing TestStudent class?

A. null:0

B. James:25

C. Line n1 causes compilation error

D. Line n2 causes compilation error

E. An exception is thrown at runtime

2.1.32 Which of the following operators is used in lambda

expressions?

A. =>

B. = >

C. ->

D. - >

2.1.33 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       System. out .println( new RuntimeException()); //Line n1
        System. out .println( new RuntimeException(
                    "HELLO" )); //Line n2
        System. out .println( new RuntimeException(
                   new RuntimeException( "HELLO" ))); //Line n3
    }
}
```

Does above code compile successfully?

A. Yes

B. No

2.1.34 Consider below codes of 3 java files:

```
//Super.java
package com.udayankhattry.ocp1;

public interface Super {
   String name = "SUPER" ; //Line n1
}


//Sub.java
package com.udayankhattry.ocp1;

public interface Sub extends Super { //Line n2
}

//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      Sub sub = null ;
      System. out .println(sub. name ); //Line n3
    }
}
```

Which of the following statements is correct?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Line n3 throws an exception at runtime

E. Test class compiles successfully and on execution prints SUPER on to the console

2.1.35 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var score = 30 ; // Line n1
        var grade = 'F' ; // Line n2
        if ( 50 <= score < 60 ) // Line n3
            grade = 'D' ;
        else if ( 60 <= score < 70 ) // Line n4
            grade = 'C' ;
        else if ( 70 <= score < 80 ) // Line n5
            grade = 'B' ;
        else if (score >= 80 )
           grade = 'A' ;
      System. out .println(grade);
   }
}
```

What is the result of compiling and executing Test class?

A. Compilation error
B. A
C. B
D. C
E. D
F. F

2.1.36 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String str = "BEVERAGE" ;
        String [] arr = str.split( "E" , 3 );
        System. out .println(String. join ( "." , arr));
    }
}
```

What is the result of compiling and executing Test class?

A. BEVERAGE
B. B.VERAGE
C. B.V.RAGE
D. B.V.RAG.
E. B.V.RAG..
F. Compilation error

2.1.37 Consider below code snippet:

```
interface Workable {
    void work();
}

/*INSERT*/ {
    public void work() {} //Line n1
}
```

And the statements:

1.	**abstract class** Work **implements** Workable
2.	**class** Work **implements** Workable
3.	**interface** Work **extends** Workable
4.	**abstract interface** Work **extends** Workable
5.	**abstract class** Work

How many statements can replace /*INSERT*/ such that there is no compilation error?

A. One statement

B. Two statements

C. Three statements

D. Four statements

E. Five statements

2.1.38 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    int i1 = 10 ;
    static int i2 = 20 ;

    private void change1( int val) {
        i1 = ++val; //Line n1
        i2 = val++; //Line n2
    }

    private static void change2( int val) {
        i1 = --val; //Line n3
        i2 = val--; //Line n4
    }

    public static void main(String[] args) {
        change1( 5 ); //Line n5
        change2 ( 5 ); //Line n6
        System. out .println( i1 + i2 ); //Line n7
    }
}
```

Which of the following statements are correct regarding above code?
Select ALL that apply.

A. Line n1 causes compilation error
B. Line n2 causes compilation error
C. Line n3 causes compilation error
D. Line n4 causes compilation error
E. Line n5 causes compilation error
F. Line n6 causes compilation error
G. Line n7 causes compilation error
H. Above code compiles successfully
I. Above code prints 8 on execution
J. Above code prints 30 on execution

2.1.39 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface I1 {
    void m1();
}

interface I2 extends I1 {
    void m2();
}

interface I3 extends I2 {
    void m3();
}

interface I4 {
   String toString();
}
```

```
public class Test {
    public static void main(String[] args) {
       /*INSERT*/
    }
}
```

And the statements:

1.	I1 i1 = () -> System. *out* .println(1);
2.	I2 i2 = () -> System. *out* .println(2);
3.	I3 i3 = () -> System. *out* .println(3);
4.	I4 i4 = () -> System. *out* .println(4);
5.	I4 i5 = () -> "" ;

Which of the above statements can replace /*INSERT*/ such that there is no compilation error?

A. Only one statement

B. Two statements

C. Three statements

D. Four statements

E. All Five statements

F. None of the statements

2.1.40 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   \---tester
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                   \---test
|                           TestStringUtil.java
|
+---out
```

```
\---bin
    \---stringutility
        |   module-info.class
        |
        \---com
            \---util
                    StringUtil.class
```

There is an exploded module 'stringutility' available under bin directory and its module descriptor file exports the package 'com.util'.

Below are the codes of Java files:-

C:\src\tester\module-info.java:

```
module tester {
    requires stringutility;
}
```

C:\src\tester\com\udayankhattry\ocp1\test\TestStringUtil.java:

```
package com.udayankhattry.ocp1.test;

import com.util.StringUtil;

public class TestStringUtil {
    public static void main(String[] args) {
       System. out .println(StringUtil.repeat( "Tic" , 5 ));
    }
}
```

Which of the following sets of javac and java command execute successfully, if executed from C:\?

A.	javac -d out --module-source-path src;bin -m tester java --module-path bin -m tester/com.udayankhattry.ocp1.test.TestStringUtil
B.	javac -d out --module-source-path src,bin -m tester java --module-path out -m tester/com.udayankhattry.ocp1.test.TestStringUtil

C.	javac -d out --module-source-path src --module-class-path bin -m tester java --module-path bin;out -m tester
D.	javac -d out --module-source-path src --module-path bin -m tester java --module-path bin;out -m tester/com.udayankhattry.ocp1.test.TestStringUtil

2.1.41 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        boolean flag = false ;
        do {
            if (flag = !flag) { //Line n1
                System. out .print( 1 ); //Line n2
                continue ; //Line n3
            }
            System. out .print( 2 ); //Line n4
        } while (flag); //Line n5
    }
}
```

What will be the result of compiling and executing Test class?

A. 1

B. 2

C. 12

D. 21

E. 212

F. 121

G. 112

H. 221

I. Compilation error

2.1.42 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       StringBuilder sb =
               new StringBuilder( "Breathe Deeply" );
       String str1 = sb.toString();
       String str2 = "Breathe Deeply" ;

          System. out .println(str1 == str2);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. true

C. false

D. An exception is thrown at runtime

2.1.43 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Super {
   public String num = "10" ; //Line n1
}

class Sub extends Super {
   protected int num = 20 ; //Line n2
}

public class Test {
    public static void main(String[] args) {
       Super obj = new Sub();
       System. out .println(obj. num += 2 ); //Line n3
    }
}
```

What will be the result of compiling and executing above code?

A. Compilation error at Line n2
B. Compilation error at Line n3
C. It executes successfully and prints 12 on to the console
D. It executes successfully and prints 22 on to the console
E. It executes successfully and prints 102 on to the console
F. It executes successfully and prints 202 on to the console

2.1.44 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void change( int num) {
       num++;
       System. out .println(num);
   }

    public static void main(String[] args) {
       int i1 = 1 ;
       Test. change (i1);
       System. out .println(i1);
   }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation Error	B.	2 1
C.	2 2	D.	1 1
E.	None of the other options		

2.1.45 A bank's swift code is generally of 11 characters and used in international money transfers.

An example of swift code: ICICINBBRT4
ICIC: First 4 letters for bank code
IN: Next 2 letters for Country code

BB: Next 2 letters for Location code
RT4: Next 3 letters for Branch code

Which of the following code correctly extracts country code from the swift code referred by String reference variable swiftCode?

A.	swiftCode.substring(4 , 6);
B.	swiftCode.substring(5 , 6);
C.	swiftCode.substring(5 , 7);
D.	swiftCode.substring(4 , 5);

2.1.46 How many of the below code statements, written inside main method will compile successfully?

1.	var arr1 = new int []{ 10 };
2.	var arr2 = new String[][] {};
3.	var arr3 = new char [][] {{}};
4.	var arr4 = { 10 , 20 , 30 };
5.	var arr5 = new String[][] { new String[]{ "LOOK" }, new String[] "UP" }};
6.	var [] arr6 = new int [] { 2 , 3 , 4 };

A. None of the statements compile successfully

B. Only 1 statement compiles successfully

C. 2 statements compile successfully

D. 3 statements compile successfully

E. 4 statements compile successfully

F. 5 statements compile successfully

G. All 6 statements compile successfully

2.1.47 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String fName = "Joshua" ;
        String lName = "Bloch" ;
        System. out .println(fName = lName);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. false

C. true

D. None of the other options

2.1.48 Consider below code of A.java file:

```
public class A {
    public static void main(String [] args) {
        System. out .print( "A" );
    }
}

public class B {
    public static void main(String... args) {
        System. out .print( "B" );
    }
}

public class C {
    public static void main(Object... args) {
        System. out .print( "C" );
    }
}

public class D {
    public static void main(String a) {
        System. out .print( "D" );
    }
}
```

And the command:

```
java A.java
```

What is the result?

A. A
B. AB
C. ABC
D. ABCD
E. CD
F. None of the other options

2.1.49 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Parent {
   String quote = "MONEY DOESN'T GROW ON TREES" ;
}

class Child extends Parent {
   String quote = "LIVE LIFE KING SIZE" ;
}

class GrandChild extends Child {
   String quote = "PLAY PLAY PLAY" ;
}

public class Test {
    public static void main(String[] args) {
       GrandChild gc = new GrandChild();
       System. out .println( /*INSERT*/ );
   }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print MONEY DOESN'T GROW ON TREES on to the console?
Select ALL that apply.

A	. gc. **quote**
B	. (Parent)gc. **quote**
C	. ((Parent)gc). **quote**
D	. ((Parent)(Child)gc). **quote**
	(Parent)(Child)gc. **quote**

E.	

2.1.50 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        var a = 3 ; //Line n1
        var b = 5 ; //Line n2
        var c = 7 ; //Line n3
        var d = 9 ; //Line n4
        boolean res = --a + --b < 1 && c++ + d++ > 1 ;
       System. out .printf( "a = %d, b = %d, c = %d,
              d = %d, res = %b" , a, b, c, d, res);
    }
}
```

What will be the result of compiling and executing Test class?

A. a = 2, b = 4, c = 7, d = 9, res = false

B. a = 2, b = 4, c = 8, d = 10, res = false

C. a = 2, b = 4, c = 7, d = 9, res = true

D. a = 2, b = 4, c = 8, d = 10, res = true

E. a = 3, b = 5, c = 8, d = 10, res = false

F. a = 3, b = 5, c = 8, d = 10, res = true

2.1.51 Which of these access modifiers can be used for a top level interface?

A. private

B. protected

C. public

D. All of the other options

2.1.52 Given code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
import java.util.Arrays;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       StringBuilder [] arr = { new StringBuilder( "A" ),
              new StringBuilder( "A" )};
       List<StringBuilder> list = Arrays. asList (arr);
        for ( int i = 0 ; i < 2 ; i++)
           if (i == 0 )
              list.forEach(sb -> sb.append( "B" ));
           else
               list.forEach(sb -> sb.append( "C" ));

          list.forEach(sb -> System. out .println(sb));
    }
}
```

What will be the result of compiling and executing Test class?

A.	A A	B.	ABC ABC
C.	ABB ACC	D.	BC BC
E.	AB AB	F.	AC AC
G.	Runtime exception is thrown		

2.1.53 Which of the following keywords is used to manually throw an exception?

A. throw

B. throws

C. thrown

D. catch

2.1.54 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String [] arr = new String[ 7 ];
       System. out .println(arr);
    }
}
```

What will be the result of compiling and executing Test class?

A. An exception is thrown at runtime

B. Compilation Error

C. It prints null

D. It prints some text containing @ symbol

2.1.55 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   +---servicing
|   |   |   module-info.java
|   |   |
|   |   \---com
|   |       \---store
|   |           \---support
|   |                   Car.java
|   |                   Service.java
|   |
|   \---testing
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                       Test.java
```

```
|
\---cls
```

Below are the file details:-

C:\src\servicing\module-info.java:

```
module servicing {
    exports com.store.support;
}
```

C:\src\servicing\com\store\support\Car.java:

```
package com.store.support;

class Vehicle {
    public String toString() {
        return "VEHICLE" ;
    }
}

public class Car extends Vehicle {
    public String toString() {
        return "CAR" ;
    }
}
```

C:\src\servicing\com\store\support\Service.java:

```
package com.store.support;

public class Service {
    public static void doService(Vehicle vehicle) {
        System. out .println( "SERVICING " + vehicle);
    }
}
```

C:\src\testing\module-info.java:

```
module testing {
    requires servicing;
}
```

C:\src\testing\com\udayankhattry\ocp1\Test.java:

```
package com.udayankhattry.ocp1;

import com.store.support.*;

public class Test {
    public static void main(String[] args) {
       Vehicle obj = new Car();
       Service.doService(obj);
    }
}
```

And the below commands to be executed from C:\

```
javac -d cls --module-source-path src -m testing
java -p cls -m testing/com.udayankhattry.ocp1.Test
```

Which of the following statements is correct?

A. Given code compiles successfully and output is: SERVICING CAR

B. Given code compiles successfully and output is: SERVICING VEHICLE

C. javac command causes some error

D. java command throws an exception at runtime

2.1.56 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class M { }
class N extends M { }
class O extends N { }
class P extends O { }

public class Test {
    public static void main(String args []) {
       M obj = new O();
        if (obj instanceof M)
           System. out .print( "M" );
        if (obj instanceof N)
           System. out .print( "N" );
        if (obj instanceof O)
            System. out .print( "O" );
        if (obj instanceof P)
            System. out .print( "P" );
    }
}
```

What will be the result of compiling and executing Test class?

A. MNO

B. MNP

C. NOP

D. MOP

E. MNOP

F. O

G. M

2.1.57 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        m1 ( null );
    }

    static void m1(CharSequence s) {
        System. out .println( "CharSequence" );
    }

    static void m1(String s) {
        System. out .println( "String" );
    }

    static void m1(Object s) {
        System. out .println( "Object" );
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation Error

B. CharSequence

C. String

D. Object

2.1.58 Consider below code of "super.java" file:

```
package com.udayankhattry.ocp1;

class _________ { //multiple underscores after class keyword
    public static void main(String [] args) {
      System. out .println( "HELP!!!" );
    }
}
```

And the command:

```
java super.java
```

What is the result? Please note super is a keyword in java.

A. Above command causes error

B. HELP!!!

2.1.59 In JDK 11, is it allowed to create a class in default package (with missing package statement)?

A. Yes

B. No

2.1.60 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
      String text = null ;
       /*INSERT*/
    }
}
```

Following options are available to replace /*INSERT*/:

1.	System. *out* .println(text.repeat(3));
2.	System. *out* .println(**null** + **null** + **null**);
3.	System. *out* .println(**null** + **"null"** + **null**);
4.	System. *out* .println(text *= 3);
5.	System. *out* .println(text += **"null"** .repeat(2));
6.	System. *out* .println(text + text + text);

7.	text += **null** ;
	System. *out* .println((text.concat(**null**)));

How many of the above options can be used to replace /*INSERT*/ (separately and not together) such that nullnullnull is printed on to the console?

A. One option only

B. Two options only

C. Three options only

D. Four options only

E. Five options only

F. Six options only

G. All seven options

2.1.61 _________ modifier is most restrictive and __________ modifier is least restrictive.

Which of the following options (in below specified order) can be filled in above blank spaces?

A. public, private

B. private, public

C. default (with no access modifier specified), public

D. protected, public

E. default (with no access modifier specified), protected

2.1.62 Consider below code fragment:

```
package com.udayankhattry.ocp1;

abstract class Food {
   protected abstract double getCalories();
}

class JunkFood extends Food {
    double getCalories() {
        return 200.0 ;
    }
}
```

Which 3 modifications, done independently, enable the code to compile?

A. Make the getCalories() method of Food class public

B. Remove the protected access modifier from the getCalories() method of Food class

C. Make the getCalories() method of Food class private

D. Make the getCalories() method of JunkFood class protected

E. Make the getCalories() method of JunkFood class public

F. Make the getCalories() method of JunkFood class private

2.1.63 Given below the directory/file structure on Windows platform:

```
C:
+---src
|  \---com.display
|          module-info.java
|
\---out
```

Contents of C:\src\com.display\module-info.java file:

```
module com.display {
    /*INSERT*/ java.se;
}
```

And the command to be executed from C:\

```
javac -d out --module-source-path src --module com.display
```

Which of the below options can replace /*INSERT*/ such that above javac command executes successfully (no compilation error in the code)?

Select ALL that apply.

A. exports

B. export

C. import

D. imports

E. requires

F. requires transitive

2.1.64 Consider below code fragment:

```
String place = "MISSS" ;
System. out .println(place.replace( "SS" , "T" ));
```

What is the output?

A. MIST

B. MITS

C. MISSS

D. MIT

2.1.65 Consider below code of Player.java:

```
class Greet {
    static void main(String [] args) {
       System. out .println( "Welcome! " + args[ 1 ]);
    }
}

public class Player {
    public static void main(String [] args) {
       Greet. main (args);
    }
}
```

And the commands:

```
javac Player.java
java Player Cristiano Ronaldo
```

What is the result?

A. Welcome! Cristiano

B. Welcome! Ronaldo

C. An Exception is thrown at runtime

D. Compilation error in Greet class as main method is not public

E. Greet.main(args); causes compilation error in Player class

2.1.66 In which of the Java versions, JPMS (Java Platform Module System) was introduced?

A. Java 7

B. Java 8

C. Java 9

D. Java 10

E. Java 11

2.1.67 Given below the directory/file structure on Windows platform:

```
C:
\---src
    \---library
        |   module-info.java
        |
        \---com
            \---udayankhattry
                +---books
                |       Book.java
                |
                \---members
                        Member.java
```

Below are the file contents:-

C:\src\library\com\udayankhattry\books\Book.java:

```java
package com.udayankhattry.books;

public class Book {
    //Lots of valid codes
}
```

C:\src\library\com\udayankhattry\members\Member.java:

```java
package com.udayankhattry.members;

public class Member {
    //Lots of valid codes
}
```

C:\src\library\module-info.java:

```java
module library {
     //INSERT CODE HERE
}
```

The author of this module wants to export the package 'com.udayankhattry.books' to 'bookhouse' and 'onlinestore' modules only and the package 'com.udayankhattry.members' to 'test' module only.

Which of the following options, when inserted in module-info.java file, will fulfill his requirements?

A.	exports com.udayankhattry.books to bookhouse;
	exports com.udayankhattry.books to onlinestore;
	exports com.udayankhattry.members to test;

B.	exports com.udayankhattry.books to bookhouse, onlinestore; exports com.udayankhattry.members to test;
C.	export com.udayankhattry.books to bookhouse, onlinestore; exports com.udayankhattry.members to test;
D.	export com.udayankhattry.books to bookhouse, onlinestore; export com.udayankhattry.members to test;

E.	export com.udayankhattry.books to bookhouse;
	export com.udayankhattry.books to onlinestore;
	export com.udayankhattry.members to test;

F.	exports com.udayankhattry.books to bookhouse & onlinestore;
	exports com.udayankhattry.members to test;

G.	exports com.udayankhattry.books;
	exports com.udayankhattry.members to test;

2.1.68 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static void div( int i, int j) {
        try {
            System. out .println(i / j);
        } catch (ArithmeticException e) {
            throw (RuntimeException)e;
        }
    }

    public static void main(String[] args) {
        try {
            div ( 5 , 0 );
        } catch (ArithmeticException e) {
            System. out .println( "AE" );
        } catch (RuntimeException e) {
            System. out .println( "RE" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Program ends abruptly

C. AE is printed on to the console and program terminates successfully

D. RE is printed on to the console and program terminates successfully

2.1.69 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface ITester {
    void test();
}

public class Test {
    public static void main(String[] args) {
       ITester obj = () -> System. out .println( "KEEP CALM" );
   }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. KEEP CALM

C. Program compiles and executes successfully but nothing is printed on to the console

D. An exception is thrown at runtime

2.1.70 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Profitable {
    double profitPercentage = 42.0 ;
}

class Business implements Profitable {
   double profitPercentage = 50.0 ; //Line n1
}

public class Test {
    public static void main(String[] args) {
       Profitable obj = new Business(); //Line n2
        System. out .println(obj. profitPercentage ); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Test class compiles successfully and on execution prints 42.0 on to the console

E. Test class compiles successfully and on execution prints 50.0 on to the console

2.1.71 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util. ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add( "apple" );
        fruits.add( "orange" );
        fruits.add( "grape" );
        fruits.add( "mango" );
        fruits.add( "banana" );
        fruits.add( "grape" );

           if (fruits.remove( "grape" ))
            fruits.remove( "papaya" );

           System. out .println(fruits);
    }
}
```

What will be the result of compiling and executing Test class?

A. An exception is thrown at runtime

B. Compilation error

C. [apple, orange, mango, banana]

D. [apple, orange, mango, banana, grape]

E. [apple, orange, grape, mango, banana, grape]

2.1.72 Consider below code fragment:

```
import java.util.*;

class A{}
class B extends A{}

abstract class Super {
   abstract List<A> get() throws IndexOutOfBoundsException;
}

abstract class Sub extends Super {
   /*INSERT*/
}
```

Which of the following options replaces /*INSERT*/ such that there is no compilation error?

A.	abstract List<A> get() throws ArrayIndexOutOfBoundsException
B.	abstract List<B> get();
C.	abstract ArrayList<A> get() throws Exception;
D.	abstract ArrayList<B> get();

2.1.73 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int i = 0 ;
       String res = null ;
        for (String [] s = { "A" , "B" , "C" , "D" };;
              res = String. join ( "." , s)) { //Line n1
           if (i++ == 0 )
              continue ;
           else
             break ;
       }
       System. out .println(res); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error at Line n1
B. Compilation error at Line n2
C. A.B.C.D
D. A.B
E. A.
F. A
G. null

2.1.74 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int elements = 0 ;
       Object [] arr = { "A" , "E" , "I" ,
             new Object(), "O" , "U" }; //Line n1
        for ( var obj : arr) { //Line n2
            if (obj instanceof String) {
                continue ;
           } else {
                break ;
           }
           elements++; //Line n3
        }
       System. out .println(elements); //Line n4
    }
}
```

What will be the result of compiling and executing Test class?

A. 0
B. 1
C. 3
D. 5
E. 6

F. Compilation error at Line n1

G. Compilation error at Line n2

H. Compilation error at Line n3

I. Compilation error at Line n4

2.1.75 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       System. out .println( "Password" + 1 + 2 + 3 + 4 );
   }
}
```

What will be the result of compiling and executing Test class?

A. Password10

B. Password19

C. Password1234

D. Password 10

2.1.76 Given codes of A.java file:

```
//A.java
package com.udayankhattry.ocp1;

public class A {
   public int i1 = 1 ;
   protected int i2 = 2 ;
}
```

and B.java file:

```
//B.java
package com.udayankhattry.ocp1.test;

import com.udayankhattry.ocp1.A;

public class B extends A {
    public void print() {
        A obj = new A();
        System. out .println(obj.i1); //Line 8
         System. out .println(obj.i2); //Line 9
         System. out .println( this .i2); //Line 10
         System. out .println( super .i2); //Line 11
    }

     public static void main(String [] args) {
         new B().print();
    }
}
```

One of the statements inside print() method causes compilation error.
Which of the below solutions will help to resolve compilation error?

A. Comment the statement at Line 8

B. Comment the statement at Line 9

C. Comment the statement at Line 10

D. Comment the statement at Line 11

2.1.77 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        System. out .println( add ( 90 , 7 ));
        System. out .println( add ( 'a' , 1 ));
              //ASCII code for 'a' is 97 and 'b' is 98
    }

     public static var add( int v1, int v2) {
         return v1 + v2;
    }
}
```

What is the result of compiling and executing above code?

A.	97 98	B.	a b
C.	a 98	D.	97 b
E.	Compilation error	F.	Runtime error

2.1.78 Given below the directory/file structure on Windows platform:

```
C:
+---codes
|   \---mymodule
|           module-info.java
|           Test.java
|
\---mods
```

Below is the code of Test.java file:

```
public class Test {
    public static void main(String [] args) {
       System. out .println(String. join ( "-" , args)); //Line n1
    }
}
```

And below is the code of module-info.java file:

```
module mymodule{
}
```

And the command executed from C:\

```
javac -d mods --module-source-path codes -m mymodule
```

What is the result?

A. Compilation error in Test class because of Line n1

B. Compilation error in Test class, for reasons other than Line n1

C. Given javac command options are not correct

D. Given command compiles successfully

2.1.79 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;

public class Test {
    public static void main(String[] args) {
       ArrayList<Integer> original =
              new ArrayList<>(); //Line n1
        original.add( 10 ); //Line n2

          ArrayList<Integer> cloned =
                (ArrayList<Integer>) original.clone();
       Integer i1 = cloned.get( 0 );
       ++i1;

          System. out .println(cloned);
    }
}
```

What will be the result of compiling and executing Test class?

A. [11]

B. [10]

C. Compilation error

D. An exception is thrown at runtime

2.1.80 Consider below code of TestSquare.java file:

```
package com.udayankhattry.ocp1;

class Square {
    int length ;
   Square sq ;

    Square( int length) {
       this . length = length;
   }
```

```
    void setInner(Square sq) {
       this . sq = sq;
   }

    int getLength() {
       return this . length ;
   }
}

public class TestSquare {
    public static void main(String[] args) {
      Square sq1 = new Square( 10 ); //Line n1
       Square sq2 = new Square( 5 ); //Line n2
       sq1.setInner(sq2); //Line n3
       System. out .println(sq1. sq . length ); //Line n4
    }
}
```

What will be the result of compiling and executing TestSquare class?

A. It prints 0 on to the console

B. It prints 5 on to the console

C. It prints 10 on to the console

D. It prints null on to the console

E. Compilation error

F. An exception is thrown at runtime

2.2 Answers of Practice Test - 2 with Explanation

2.1.1 Answer: A,D

Reason :

Jack's salary is 5000 and Liya's salary is 8000. If Employee's salary is >= 10000 then that Employee object is removed from the list.

Lambda expression for Predicate<Employee> must accept Employee type as a parameter and must return boolean value.
Allowed lambda expression is:
(Employee e) -> { return e.getSalary() >= 10000; },
Can be simplified to: (e) -> { return e.getSalary() >= 10000; } => type can be removed from left side of the expression.
Further simplified to: e -> { return e.getSalary() >= 10000; } => if there is only one parameter in left part, then round brackets (parenthesis) can be removed.
Further simplified to: e -> e.getSalary() >= 10000 => if there is only one statement in the right side then semicolon inside the body, curly brackets and return keyword can be removed. But all 3 [return, {}, ;] must be removed together.

NOTE: there should not be any space between - and > of arrow operator.

2.1.2 Answer: F

Reason :

case values must evaluate to the same/compatible type as the switch expression can use.
switch expression can accept following:
char or Character,
byte or Byte,
short or Short,
int or Integer,
An enum only from Java 6,
A String expression only from Java 7.
Compatible literal value or constant can be used as the switch expression .

In this case, switch expression "HELLO" is of String type.
All the 3 case values are of String type, so no issues in the given code.

1st case is the matching case, 1 is printed on to the console. No break statement inside 'case "HELLO":', hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends. So in this case, it prints 2 and after that 3 and finally control goes out of the switch block.
main method ends and program terminates successfully after printing 123 on to the console.

2.1.3 Answer: E

Reason :
Let's solve the expression at Line n1:
!flag1 == flag2 != flag3 == !flag4
(!flag1) == flag2 != flag3 == (!flag4) //Logical NOT has got highest precedence among given operators
((!flag1) == flag2) != flag3 == (!flag4) //== and != have same precedence and left to right associative, grouping == first
(((!flag1) == flag2) != flag3) == (!flag4) //grouping != next
Above expression is left with single operator ==, whose left side is: (((!flag1) == flag2) != flag3) and right side is: (!flag4). As == is a binary operator, so left side is evaluated first.
((false == flag2) != flag3) == (!flag4) //!flag1 is false
((false == false) != flag3) == (!flag4) //flag2 is false
(true != flag3) == (!flag4) //(false == false) evaluates to true
(true != true) == (!flag4) //flag3 is true
false == (!flag4) //(true != true) evaluates to false
false == true //!flag4 is true
false //(false == true) evaluates to false
Hence, false is printed on to the console.

Let's solve the expression at Line n2:
flag1 = flag2 != flag3 == !flag4
flag1 = flag2 != flag3 == (!flag4) //Logical NOT has got highest precedence among given operators
flag1 = (flag2 != flag3) == (!flag4) //== and != have same precedence and left to right associative, grouping == first
flag1 = ((flag2 != flag3) == (!flag4)) //grouping == nex t
Above expression is left with single assignment operator =, whose right side needs to be evaluated first

flag1 = ((false != flag3) == (!flag4)) //flag2 is false
flag1 = ((false != true) == (!flag4)) //flag3 is true
flag1 = (true == (!flag4)) //(false != true) evaluates to true
flag1 = (true == true) //!flag4 is true
flag1 = true //(true == true) evaluates to true
true is assigned to flag1 and true is also printed on to the console

One suggestion: In the real exam, if you find a question containing multiple expressions, then first check if there is any compilation error or not. If there is no compilation error in all the expressions, then only solve the expressions.

2.1.4 Answer: C

Reason :
ConcurrentModificationException exception may be thrown for following condition:
1. Collection is being iterated using Iterator/ListIterator or by using for-each loop.

And

2. Execution of Iterator.next(), Iterator.remove(), ListIterator.previous(), ListIterator.set(E) & ListIterator.add(E) methods. These methods may throw java.util.ConcurrentModificationException in case Collection had been modified by means other than the iterator itself, such as Collection.add(E) or Collection.remove(Object) or List.remove(int) etc.

For the given code, 'sports' list is being iterated using the Iterator<String>.
hasNext() method of Iterator has following implementation:

```
public boolean hasNext() {
    return cursor != size;
}
```

Where cursor is the index of next element to return and initially it is 0.

1st Iteration: cursor = 0, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns "Windsurfing".
2nd Iteration: cursor = 1, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns "Aerobics". As "Aerobics" starts with "A", hence sports.remove(dryFruit) removes "Aerobics" from the list and hence reducing the list's size by 1, size becomes 3 .
3rd Iteration: cursor = 2, size = 3, hasNext() returns true. iterator.next()

method throws java.util.ConcurrentModificationException.

If you want to remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove(...) method. Using List.remove(...) method while iterating the list (using the Iterator/ListIterator or for-each) may throw java.util.ConcurrentModificationException.

2.1.5 Answer: C

Reason :
'append' method is overloaded in StringBuilder class: append(String), append(StringBuffer) and append(char[]) etc. In this case compiler gets confused as to which method `append(null)` can be tagged because String, StringBuffer and char[] are not related to each other in multilevel inheritance. Hence `sb.append(null)` causes compilation error.

2.1.6 Answer: A

Reason :
Even though code seems to be checking the knowledge of ArrayList but it actually checks the knowledge of Polymorphism.
List<Sellable> list = new ArrayList<>(); is valid statement and list can accept any object passing instanceof check for Sellable type.
Rabbit implements Sellable hence new Rabbit() can be added to list.
But as Mammal doesn't implement Sellable hence new Mammal() can't be added to list.

Other initializer blocks can be verified on similar lines. So there is only one initializer block, which causes compilation error.

2.1.7 Answer: A

Reason :
Constructors cannot use final, abstract or static modifiers. As no-argument constructor of MySubClass uses final modifier, therefore it causes compilation error.

2.1.8 Answer: G

Reason :
Planet is defined in 'com.udayankhattry.galaxy' package, Creator is defined in 'com.udayankhattry.ocp1' package and TestCreator is defined in 'com.udayankhattry.ocp1.test' package.

Planet class doesn't mention 'Creator' or 'TestCreator' and hence no import statements are needed in Planet class.
Creator class uses the name 'Planet' in its code and hence Creator class needs to import Planet class using 'import com.udayankhattry.galaxy.Planet;' statement or 'import com.udayankhattry.galaxy.*;' statement.
TestCreator class uses the name 'Creator' in its code and hence TestCreator class needs to import Creator class using 'import com.udayankhattry.ocp1.Creator;' statement or 'import com.udayankhattry.ocp1.*;' statement.

Please note, even though in TestCreator class, `Creator.create()` returns an instance of Planet class but as name 'Planet' is not used, hence Planet class is not needed to be imported.

Planet class correctly overrides toString() method, hence when an instance of Planet class is passed to println(...) method, as in the below statement:
System.out.println(Creator.create());
toString() method defined in the Planet class is invoked, which print "Planet: Earth" on to the console.

2.1.9 Answer: C

Reason :
Default values are assigned to all array elements. As Boolean is of reference type, hence arr[0] = null and arr[1] = null. After addition list contains [null, null].

list.remove(0) removes the item at index 0 and returns the removed Boolean object referring to null. If expression can specify Boolean type, so no compilation error over here.

For the boolean expression of if-block, Java runtime tries to extract the stored boolean value using booleanValue() method, and this throws an instance of NullPointerException as booleanValue() method is invoked on null reference.

2.1.10 Answer: B

Reason :
Method method's 1st parameter is Display, hence a lambda expression can

be passed as 1st argument and 2nd parameter is String, so String object needs to be passed as 2nd argument.

Let's check all the options one by one:
s -> { System.out.println(s.toUpperCase()) }, "lambda": ✗ Semicolon before closing curly bracket is missing.

s -> System.out.println(s.toUpperCase()), "lambda": ✓ It is a correct lambda expression for target type Display. 2nd argument is also of String type. Output will be LAMBDA.

s -> s.toUpperCase(), "lambda": ✗ It is a legal syntax but nothing is printed on to the console.

s -> System.out.println(s.toUpperCase()): ✗ 2nd argument of String type is missing.

2.1.11 Answer: A,C,E

Reason :
Let's check all the statements one by one:

short s1 = 10;
Above statement compiles successfully, even though 10 is an int literal (32 bits) and s1 is of short primitive type which can store only 16 bits of data.
Here java does some background task, if value of int literal can be easily fit to short primitive type (-32768 to 32767), then int literal is implicitly casted to short type.
So above statement is internally converted to:
short s1 = (short)10;

short s2 = 32768;
It causes compilation failure as 32768 is out of range value.

final int i3 = 10;
short s3 = i3 ;
Above code compiles successfully. If you are working with final variable and the value is within the range, then final variable is implicitly casted to target type, as in this case i3 is implicitly casted to short.

final int i4 = 40000;

```
short s3 = i4;
```

It causes compilation failure as 40000 is out of range value.

```
final int i5 = 10;
short s5 = i5 + 100;
```

Above code compiles successfully. If you are working with constant expression and the resultant value of the constant expression is within the range, then resultant value is implicitly casted. In this case, resultant value 110 is implicitly casted.

```
final int m = 25000;
final int n = 25000;
short s6 = m + n;
```

m + n is a constant expression but resultant value 50000 is out of range for short type, hence it causes compilation failure.

```
int i7 = 10;
short s7 = i7;
```

Compilation error as i7 is non-final variable and hence cannot be implicitly casted to short type.

2.1.12 Answer: A

Reason :

When start is divisible by 2 [2, 4, 6, 8, 10], continue; statement takes the control to boolean expression and hence Line n1 is not executed.

Line n1 is executed only in the case of odd numbers which are less than or equals to 10. Therefore, result is the sum of numbers 1,3,5,7,9; which is 25.

2.1.13 Answer: C

Reason :

new String[1] creates a String [] object of one element and as all the elements of array are initialized to respective zeros (in case of primitive type) or null (in case of reference type), arr[0] refers to null .

isBlank() method of String class (available since Java 11) returns true if the string is empty or contains only white space codepoints, otherwise false. As isBlank() is available, hence `arr[0].isBlank()` compiles successfully.

But as method 'isBlank()' is invoked on null reference (arr[0] refers to null), so NullPointerException is thrown at runtime.

2.1.14 Answer: C

Reason :

NullPointerException extends RuntimeException and in multi-catch syntax we can't specify multiple Exceptions related to each other in multilevel inheritance.

2.1.15 Answer: B

Reason :

Method move() declared in Moveable interface is implicitly public and abstract.

Abstract class Animal has non-abstract method move() and it is declared with no modifier (package scope). Abstract class in java can have 0 or more abstract methods. Hence Animal class compiles successfully.

class Dog extends Animal and as both the classes Animal and Dog are within the same package 'com.udayankhattry.ocp1', Dog inherits the move() method defined in Animal class.

Dog class implements Moveable interface as well, therefore it must implement public move() method as well. But as inherited move() method from Animal class is not public, therefore Dog class fails to compile.

2.1.16 Answer: D

Reason :

At Line n2, local variable 'str' shadows the static variable 'str' created at Line n1. Hence, for the expression `str + "SIMPLE"`, Java compiler complains as local variable 'str' is not initialized.

2.1.17 Answer: F

Reason :

verify(...) method accepts 2 parameters: List<String> type and Predicate<List<String>> type.

For Predicate<List<String>>, test method should have the signature: `boolean test(List<String> t);`

Line n3 correctly invokes the test(...) method of Predicate<List<String>> interface.

`List<String> list = new ArrayList<>(List.of("A", "E", "I", "O", "U"));` It constructs the ArrayList instance containing the elements of the specified collection (list object returned by List.of(...)). This ArrayList object is not backed by the passed collection and hence can be modified without any issues. list --> {A, E, I, O, U}

remove(Object) method of List interface returns true if removal was successful otherwise false.
add(E) method of List interface returns true if addition was successful otherwise false.

As, both the methods, remove(Object) and add(E) return boolean value, hence these methods can be used as the body of lambda expression of Predicate interface.

Line n1 is a valid call to verify(...) method and it removes matching element "I" from the list and returns true. So, list --> {A, E, O, U}. As true is returned, hence control goes inside if-block to execute Line n2.

Line n2 is also a valid call to verify(...) method and it adds "I" at the end of the list and returns true. list --> {A, E, O, U, I}. But as there is nothing inside this if-block, hence control goes to the last statement inside main(String[]) method, `System.out.println(list);` is executed and [A, E, O, U, I] is printed on to the console.

2.1.18 Answer: B

Reason :
list --> [C, Z, A, R]

sublist method is declared in List interface:
List<E> subList(int fromIndex, int toIndex)
fromIndex is inclusive and toIndex is exclusiv e
It returns a view of the portion of this list between the specified fromIndex and toIndex. The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list and vice-versa.
If returned list (or view) is structurally modified, then modification are reflected in this list as well but if this list is structurally modified, then the semantics of the list returned by this method become undefined.
If fromIndex == toIndex, then returned list is empty.
If fromIndex < 0 OR toIndex > size of the list OR fromIndex > toIndex, then IndexOutOfBoundsException is thrown.

list.subList(1, 2) --> [Z] (fromIndex is inclusive and endIndex is exclusive, so start index is 1 and end index is also 1)
list.subList(1, 2).clear(); => It removes element "Z" from the view and also from the ArrayList object referred by list. After this statement, list --> [C, A, R]

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO "

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g., List<String> list = new ArrayList<>(); list.add("A"); list.add(null); String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

`System.out.println(String.join("", list));` prints CAR on to the console.

2.1.19 Answer: A

Reason :
break; and continue; are used inside for-loop, hence no compilation error. In first iteration "Lettuce " is printed on to the console. Cursor remains on the same line as 'print' method is used and not 'println'. boolean expression of if-block returns false, control goes just after if-block and appends "salad!" on to the console.
break; statement takes the control out of for loop, main method ends and program terminates successfully.

So "Lettuce salad!" is printed on to the console.

2.1.20 Answer: C

Reason :
Default constructor (which is a no-argument constructor) is added by Java compiler, only if there are no constructors in the class.

2.1.21 Answer: C

Reason :
main(args) method is invoked recursively without specifying any exit condition, so this code ultimately throws java.lang.StackOverflowError. StackOverflowError is a subclass of Error type and not Exception type, hence it is not handled. Stack trace is printed to the console and program ends abruptly. Statements at Line n1 and Line n2 are not executed .

Java doesn't allow to catch specific checked exceptions if these are not thrown by the statements inside try block.
catch(java.io.FileNotFoundException ex) {} will cause compilation error in this case as main(args); will never throw FileNotFoundException. But Java allows to catch Exception type, hence catch (Exception ex) {} doesn't cause any compilation error.

2.1.22 Answer: C

Reason :
Test.java is a valid java file. As none of the classes in Test.java file are public, hence file name can use any valid Java identifier.
As file name is Test.java, hence to compile the code below command is used:
javac Test.java

Execution of above command creates 4 class files: A.class, B.class, C.class & D.class.
To print C on to the console, class C must be executed. To execute C class, command is:
java C

2.1.23 Answer: B

Reason :
NullPointerException extends RuntimeException. Overriding method may or may not throw any RuntimeException. Only thing to remember is that if overridden method throws any unchecked exception or Error, then overriding method must not throw any checked exceptions.
So, method log() in Derived class correctly overrides Base class's method.
Rest is simple polymorphism. 'obj' refers to an instance of Derived class and hence obj.log(); invokes method log() of Derived class, which prints "Derived: log()" on to the console.

2.1.24 Answer: A

Reason :
Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C "
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

If command-line arguments: POWER and DRIVEN are passed, then System.out.println(String.join("-", args)); will print POWER-DRIVEN.

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
    (to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

Let's check all the commands:
java -p mods -m printarguments POWER DRIVEN ✗ As it is an exploded module hence mainclass is not optional.
java -p mods -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN ✓ It prints POWER-DRIVEN on to the console.
java -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN -p mods ✗ -m option must be the last option.
java --module-source-path src -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN ✗ --module-source-path is not a valid option of java command
java --module-source-path mods -m printarguments/com.udayankhattry.ocp1.Test POWER DRIVEN ✗ --module-source-path is not a valid option of java command
java --module-path mods --module printarguments POWER DRIVEN ✗ As it is an exploded module hence mainclass is not optional.

Hence, only one command produces expected output.

2.1.25 Answer: C

Reason :
Method find() declares to throw Exception and the throw statements inside this method is throwing the Subclasses of Exception, hence no issues in find() method.
main(String []) method also provide legal try-catch block, hence the given code compiles successfully .

Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor
2. public Throwable(String message) {...} : Pass the detail message
3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause
4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)
1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)
2. public String toString() {} :
Returns a short description of this throwable. The result is the concatenation of:
the name of the class of this object
": " (a colon and a space)
the result of invoking this object's getLocalizedMessage() method

If getLocalizedMessage returns null, then just the class name is returned.

Let's check the execution:
main(String []) method invokes find() and first statement inside try-block gets executed, 1 is printed on to the console.
Next statement is executed and an instance of FileNotFoundException is thrown.
Matching catch-handler is available, so control goes inside the first catch-handler and 2 is printed on to the console. An instance of IOException is thrown by the catch-block but another catch-block at the same level are not executed.
As finally-block always executes, hence, 4 is printed on to the console and an Exception instance is thrown by the finally-block.

Control goes back to the calling method main(String []), catch-handler is executed and it prints FINALLY on to the console.

Hence output is: 124FINALLY

2.1.26 Answer: B

Reason :
Primitive type instance variables are initialized to respective zeros (byte: 0, short: 0, int: 0, long: 0L, float: 0.0f, double: 0.0, boolean: false, char: \u0000). When printed on the console; byte, short, int & long prints 0, float & double print 0.0, boolean prints false and char prints nothing or non-printable character (whitespace).
Reference type instance variables are initialized to null.

2.1.27 Answer: A

Reason :
`var arr = new Double[2];`: Right side expression creates an Double[] object of 2 elements. Target-type (Double[]) is specified on the right side, hence this statement compiles successfully. arr infers to Double[] type.

Array elements are initialized to their default values. arr is referring to an array of Double type, which is reference type and hence both the array elements are initialized to null.

To calculate arr[0] + arr[1], java runtime converts the expression to arr[0].doubleValue() + arr[1].doubleValue(). As arr[0] and arr[1] are null hence calling doubleValue() method throws NullPointerException.

2.1.28 Answer: E

Reason :
Circle class needs to be well encapsulated, this means that instance variable radius must be declared with private access modifier and getter/setter methods must be public, so that value in radius variable can be read/changed by other classes.
Out of the given options, below option is correct:
Add below 2 methods in Circle class:

```
public double getRadius() {
    return radius;
}

public void setRadius(double radius) {
    this.radius = radius;
}
```

2.1.29 Answer: D

Reason :

Format of the java command related to module is:

```
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
       (to execute the main class in a module)
```

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

When multiple modules with the same name are in different directories on the module path, first module is selected and rest of the modules with same name are ignored. As first module directory in the module path is 'out3', hence YES YOU CAN is the output.

2.1.30 Answer: B

Reason :
"mango" is different from "Mango", so there is no matching case available. default block is executed and as it is the last block inside switch hence after printing "ANY FRUIT WILL DO" control goes out of switch block, main method ends and program terminates successfully.

2.1.31 Answer: A

Reason :
Methods can have same name as the class. Student() and Student(String, int) are methods and not constructors of the class, note the void return type of these methods .
Line n1 is a valid instance method call from Student() method, hence Line n1 doesn't cause any compilation error.

As no constructors are provided in the Student class, java compiler adds default no-argument constructor. That is why the Line n2 doesn't cause any compilation error.

Default values are assigned to instance variables, hence null is assigned to name and 0 is assigned to age.

Line n3 prints null:0 on to the console.

2.1.32 Answer: C

Reason :
Arrow operator (->) was added in JDK 8 for lambda expressions. NOTE: there should not be any space between - and >.

2.1.33 Answer: A

Reason :
Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor
2. public Throwable(String message) {...} : Pass the detail message
3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause
4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Hence all 3 statements Line n1, Line n2 and Line n3 compile successfully.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)
1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)
2. public String toString() {} :
Returns a short description of this throwable. The result is the concatenation of :
the name of the class of this object
": " (a colon and a space)

the result of invoking this object's getLocalizedMessage() method

If getLocalizedMessage returns null, then just the class name is returned.

Because of the toString() method,
Line n1 prints "java.lang.RuntimeException".
Line n2 prints "java.lang.RuntimeException: HELLO"
Line n3 prints "java.lang.Exception: java.lang.RuntimeException: HELLO"

2.1.34 Answer: E

Reason :
Variable 'name' declared inside interface Super is implicitly public, static and final. Line n1 compiles successfully.
In Java a class can extend from only one class but an interface can extend from multiple interfaces. Line n2 compiles successfully.
Variable 'name' can be accessed in 2 ways: Super.name and Sub.name.
Though correct way to refer static variable is by using the type name, such as Sub.name but it can also be invoked by using Sub reference variable. Hence, sub.name at Line n3 correctly points to the name variable at Line n1.
For invoking static fields, object is not needed, therefore even if sub refers to null, sub.name doesn't throw NullPoionterException.
Test class compiles successfully and on execution prints SUPER on to the console.

2.1.35 Answer: A

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name .

At Line n1, score infers to int type.

At Line n2, grade infers to char type.

Both Line n1 and Line n2 compile successfully.

Let's check the boolean expression of Line n3:
50 <= score < 60
As multiple operators are available, so lets group the operators first on the basis of precedence and associativity.
Relational operators (<, >, <= and >=) are at same level and left to right associative, hence given expression can be grouped as:
(50 <= score) < 60
< is a binary operators with two operands: (50 <= score) on the left is of boolean type and 60 on the right is of int type. But < operator is not defined for boolean, int type and hence Line n3 causes compilation error.
Line n4 and Line n5 cause compilation error for the same reason.

2.1.36 Answer: C

Reason :
To know more about join and split methods of String class, please check the URLs:
https://udayankhattry.com/join-string/
https://udayankhattry.com/split-string/

Let's solve the given expression:
Variable 'str' refers to "BEVERAGE".
str.split("E", 3); returns ["B","V","RAGE"] as pattern is applied 3 - 1 = 2 times.
String.join(".", arr); = String.join(".", ["B","V","RAGE"]); returns "B.V.RAGE" and last statement prints B.V.RAGE on to the console.

2.1.37 Answer: C

Reason :
/*INSERT*/ cannot be replaced with interface as work() method at Line n1 is neither abstract nor default. Hence, statements 3 and 4 will not work.
Let's check other statements:
1. abstract class Work implements Workable: abstract class in java can have 0 or more abstract methods. It compiles successfully .
2. class Work implements Workable: It correctly implements the work()

method of Workable interface, hence it compiles successfully.
5. abstract class Work: abstract class in java can have 0 or more abstract methods. It compiles successfully.

Hence, out of 5 statements, 3 will compile successfully.

2.1.38 Answer: C,E,G

Reason :
i1 is an instance variable and i2 is a static variable.
Instance method can access both instance and static members. Hence, Line n1 and Line n2 compile successfully.
Static method can access only static members. Hence, Line n3 [accessing instance variable i1], Line n5 [accessing instance method change1(int)] and Line n7 [accessing instance variable i1] cause compilation error.

2.1.39 Answer: A

Reason :
Target type of lambda expression must be a Functional interface.
Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].

Let's check which of the given interfaces are Functional interfaces...

Interface I1 contains only one non-overriding abstract method m1() and hence it is a Functional interface.
Interface I2 extends I1 and therefore it has two non-overriding abstract methods m1() and m2(). I2 is NOT a Functional interface.
Interface I3 extends I2 and therefore it has three non-overriding abstract methods m1(), m2() and m3(). I3 is NOT a Functional interface.
Interface I4 contains only one overriding abstract method toString() and therefore it is NOT a Functional interface.

Out of the given 4 interfaces, only I1 is the Functional interface and hence can be used as the target type of the lambda expression .

Let's check the validity of the lambda expression:
`() -> System.out.println(1);` is the correct implementation of `void m1();` method and therefore can be assigned to target type I1.

Out of given 5 statements only statement number 1 will successfully replace the /*INSERT*/.

2.1.40 Answer: D

Reason :
Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this case, module source path is 'src' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'out' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
In this case, it is 'bin' directory .

It is clear that module 'tester' requires the module 'stringutility' to access the 'com.udayankhattry.ocp1.test.TestStringUtil' class. Compiled code of module 'stringutility' is available under 'bin' directory.
To access compiled module code of other modules at compile time use --module-path or -p option.
Hence, correct command to compile 'tester' module is:
javac -d out --module-source-path src --module-path bin -m tester

After compilation, you will get below directory/file structure:

```
C:.
+---src
|   \---tester
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                   \---test
|                           TestStringUtil.java
|
+---out
|   \---tester
|       |   module-info.class
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                   \---test
|                           TestStringUtil.class
|
\---bin
    \---stringutility
        |   module-info.class
        |
        \---com
            \---util
                    StringUtil.class
```

Format of the java command related to module is:

```
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
       (to execute the main class in a module )
```

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
Hence, correct command to execute TestStringUtil class is:
java --module-path bin;out -m tester/com.udayankhattry.ocp1.test.TestStringUtil

2.1.41 Answer: C

Reason :
Body of do-while loop is executed first and then condition is checked for the next iteration.
Initially, flag = false;
1st iteration: Boolean expression of if-block `flag = !flag` = `flag = !false` = `flag = true`: it assigns true to variable 'flag' and evaluates to true as well. Line n2 is executed and 1 is printed on to the console. Line n3 takes the control to the boolean expression of Line n5.
2nd iteration: As flag is true, boolean expression at Line n5 evaluates to true and control enters the loop's body. Boolean expression of if-block `flag = !flag` = `flag = !true` = `flag = false`: it assigns false to variable 'flag' and evaluates to false as well. Line n2 and Line n3 are not executed. Line n4 is executed, which prints 2 on to the console. Control goes to the boolean expression of Line n5.
3rd iteration: As flag is false, boolean expression at Line n5 evaluates to false and control exits the loop.

Program terminates successfully after printing 12 on to the console.

2.1.42 Answer: C

Reason :
toString() method defined in StringBuilder class doesn't use String literal rather uses the constructor of String class to create the instance of String class .

So both 'str1' and 'str2' refer to different String instances even though their contents are same. str1 == str2 returns false.

2.1.43 Answer: E

Reason :
Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

'obj' is of Super type, hence obj.num refers to num variable at Line n1, which is of String type.
Expression at Line n3:
obj.num += 2
=> obj.num = obj.num + 2
=> obj.num = "10" + 2
=> obj.num = "102"

obj.num refers to "102" and same is printed on to the console.

2.1.44 Answer: B

Reason :
There are no compilation errors and main(String[]) method is invoked on executing Test class.
i1 = 1.
`Test.change(i1)` is executed next, contents of i1 (which is 1) is copied to the variable 'num' and method change(int) starts executing.
`num++;` increments the value of num by 1, num = 2. There are no changes to the value of variable 'i1' of main(String[]) method, which still contains 1.
`System.out.println(num);` prints 2 on to the console.
change(int) method finishes its execution and control goes back to the main(String[]) method.
`System.out.println(i1);` prints 1 on to the console.

2.1.45 Answer: A

Reason :
substring(int beginIndex, int endIndex) is used to extract the substring. The substring begins at "beginIndex" and extends till "endIndex - 1".

Country code information is stored at index 4 and 5, so the correct substring method to extract country code is: swiftCode.substring(4, 6);

2.1.46 Answer: E

Reason :

`var arr1 = new int[]{10};`: Right side expression creates an int[] object of one element and 10 is assigned to that element. Target-type (int[]) is specified on the right side, hence this statement compiles successfully. arr1 infers to int[] type.

`var arr2 = new String[][] {};`: Right side expression creates a String[][] object, whose first dimension is 0 (similar to `var arr2 = new String[0] [];`), so it's a valid array object. Target-type (String[][]) is specified on the right side, hence this statement compiles successfully. arr2 infers to String[][] type.

`var arr3 = new char[][] {{}};`: Right side expression creates a char[][] object, whose first dimension is 1 (similar to `var arr3 = new char[1][];`), so it's a valid array object. Target-type (char[][]) is specified on the right side, hence this statement compiles successfully. arr3 infers to char[][] type.

`var arr4 = {10, 20, 30};`: Explicit target-type is needed for the array initializer, if you use var type. Over here as explicit target-type is missing, hence this statement causes compilation error.

`var arr5 = new String[][] {new String[]{"LOOK"}, new String[] {"UP"}};`: Right side expression creates String[][] object whose first dimension is 2 and second dimension is 1 (same as `var arr5 = new String[2][1]; arr5[0][0] = "LOOK"; arr5[1][0] = "UP";`). Target-type (String[][]) is specified on the right side, hence this statement compiles successfully. arr5 infers to String[][] type.

`var [] arr6 = new int[] {2, 3, 4};`: var is not allowed as an element type of an array, hence `var [] arr6` causes compilation error.

Hence, out of 6 statements, 4 statements compile successfully.

2.1.47 Answer: D

Reason :

Both fName and lName are of String reference type. fName refers to "Joshua" and lName refers to "Bloch".
In System.out.println() statement, we have used assignment operator (=) and not equality operator (==). So result is never boolean.
fName = lName means copy the contents of lName to fName.
As lName is referring to "Bloch" and so after the assignment, fName starts referring to "Bloch" as well.
System.out.println() finally prints the String referred by fName, which is "Bloch".
As this option is is not available, hence correct answer is "None of the other options"

2.1.48 Answer: A

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
   (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java A.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java A.java` is equivalent to (but not exactly same as):

```
javac -d <memory> A.java
java -cp <memory> A
```

Hence in this case, output is: A.
Please note that source-file can contain multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

There are no changes in the java file naming rules, if you compile the program using javac command. It will still complain about public classes not being defined in their respective files .

```
F:\>javac A.java
A.java:7: error: class B is public, should be declared in a file named
B.java
public class B {
       ^
A.java:13: error: class C is public, should be declared in a file named
C.java
public class C {
       ^
A.java:19: error: class D is public, should be declared in a file named
D.java
public class D {
       ^
3 errors
```

But Single-file program executed using java command, ignores this rule.

Also note that file name can be any name supported by underlying operating system, such as if you rename 'A.java' to '123.java'.
java 123.java will be equivalent to:

```
javac -d <memory> 123.java
java -cp <memory> A
```

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

2.1.49 Answer: C,D

Reason :
As instance variables are hidden by subclasses and not overridden, therefore instance variable can be accessed by using explicit casting.
Let's check all the options one by one:
gc.quote => It refers to "PLAY PLAY PLAY" as gc is of GrandChild class.

(Parent)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Parent type. This would cause compilation error.

((Parent)gc).quote => Variable 'gc' is casted to Parent type, so this expression refers to "MONEY DOESN'T GROW ON TREES". It is one of the correct options .

((Parent)(Child)gc).quote => 'gc' is of GrandChild type, it is first casted to Child and then to Parent type and finally quote variable is accessed, so this expression refers to "MONEY DOESN'T GROW ON TREES". It is also one of the correct options.

(Parent)(Child)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Child type. This would cause compilation error.

2.1.50 Answer: A

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

At Line n1, a infers to int type.
At Line n2, b infers to int type.
At Line n3, c infers to int type.
At Line n4, d infers to int type.

Given expression:
--a + --b < 1 && c++ + d++ > 1;
--a + --b < 1 && (c++) + (d++) > 1; //postfix has got highest precedence
(--a) + (--b) < 1 && (c++) + (d++) > 1; //prefix comes after postfix
{(--a) + (--b)} < 1 && {(c++) + (d++)} > 1; //Then comes binary +.
Though parentheses are used but I used curly brackets, just to explain.
[{(--a) + (--b)} < 1] && [{(c++) + (d++)} > 1]; //Then comes relational operator (<,>). I used square brackets instead of parentheses.
This expression is left with just one operator, && and this operator is a binary operator so works with 2 operands, left operand [{(--a) + (--b)} < 1] and right operand [{(c++) + (d++)} > 1]

Left operand of && must be evaluated first, which means [{(--a) + (--b)} < 1] must be evaluated first .

```
[{2 + (--b)} < 1] && [{(c++) + (d++)} > 1]; //a=2, b=5, c=7, d=9
[{2 + 4} < 1] && [{(c++) + (d++)} > 1]; //a=2, b=4, c=7, d=9
[6 < 1] && [{(c++) + (d++)} > 1];
false && [{(c++) + (d++)} > 1];
```

&& is short circuit operator, hence right operand is not evaluated and false is returned.

Output of the given program is: a = 2, b = 4, c = 7, d = 9, res = false

2.1.51 Answer: C

Reason :
A top level interface can be declared with either public or default modifiers.
public interface is accessible across all packages but interface declared with default modifier and be accessed in the defining package only.

2.1.52 Answer: B

Reason :
Arrays.asList(...) method returns a fixed-size list backed by the specified array and as list is backed by the specified array therefore, you cannot add or remove elements from this list. Using add/remove methods cause an exception at runtime. But you can invoke the set(int index, E element) method on the returned list.
This behavior is bit different from the List.of(...) method, which returns unmodifiable list, hence calling add/remove/set methods on the unmodifiable list throws an exception at runtime.

Given code, does not try to add new elements to the list or does not try to remove elements from the list, hence no question of any runtime exception.

Iterable<T> interface has forEach(Consumer) method. List<E> extends Collection<E> & Collection<E> extends Iterable<E>, therefore forEach(Consumer) can easily be invoked on reference variable of List<E> type.
As Consumer is a Functional Interface, hence a lambda expression can be

passed as argument to forEach() method.
forEach(Consumer) method performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception .

Initially,
list --> {["A"], ["A"]}
1st iteration: i = 0, i == 0 evaluates to true, code inside if-block gets executed. "B" is appended to both the list elements. list --> {["AB"], ["AB"]}
2nd iteration: i = 1, i == 0 evaluates to false, code inside else-block gets executed. "C" is appended to both the list elements. list --> {["ABC"], ["ABC"]}

list.forEach(sb -> System.out.println(sb)); prints below on to the console:
ABC
ABC

2.1.53 Answer: A

Reason :
catch is for catching the exception and not throwing it.
thrown is not a java keyword.
throws is used to declare the exceptions, a method can throw.
To manually throw an exception, throw keyword is used. e.g., throw new Exception();

2.1.54 Answer: D

Reason :
Variable 'arr' refers to an array object of String of 7 elements and it contains the memory address of String array object.
'arr' is of reference type, therefore when `System.out.println(arr);` is executed, toString() method defined in Object class is invoked, which returns <fully qualified name of internal array class>@<hexadecimal representation of hashcode>. That is why some text containing @ symbol is printed on to the console.

2.1.55 Answer: C

Reason :
module-info.java file of 'testing' module requires 'servicing' module and

module-info.java file of 'servicing' module exports 'com.store.support' package. Hence, 'testing' module can access all the public classes of 'com.store.support' package and their public members.

Please note, class 'Vehicle' is not public and therefore java compiler complains about `Vehicle obj = new Car();` statement .

```
C:\>javac -d cls --module-source-path src -m testing
src\testing\com\udayankhattry\ocp1\Test.java:7: error: cannot find symbol
        Vehicle obj = new Car();
        ^
  symbol:   class Vehicle
  location: class Test
1 error
```

If you replace, 'Vehicle' with 'Car' [Car obj = new Car();], then code would compile successfully and would print SERVICING CAR on to the console.

2.1.56 Answer: A

Reason :

```
M
^
N
^
O [obj refers to instance of O class]
^
P
```

obj instanceof M => true, hence "M" is printed on to the console.
obj instanceof N => true, hence "N" is printed on to the console.
obj instanceof O => true, hence "O" is printed on to the console.
but
obj instanceof P => false, "P" is not printed.

Output is: MNO

2.1.57 Answer: C

Reason :
Method m1 is overloaded to accept 3 different parameters: String, CharSequence and Object and as these are related in multilevel

inheritance, hence m1(null) is mapped to the class lowest in hierarchy, which is String class. Hence, output will be "String" .

Now if you add one more overloaded method, `static void m1(StringBuilder s) {...}` in the Test class, then `m1(null);` would cause compilation error as StringBuilder and String classes are not related to each other in multilevel inheritance.
m1(null); would match to both m1(String) and m1(StringBuilder), so m1(null) in that case would be ambiguous call and would cause compilation error.

For the same reason, System.out.println(null); causes compilation error as println method is overloaded to accept 3 reference types Object, String and char [] along with primitive types.
System.out.println(null); matches to both println(char[]) and println(String), so it is an ambiguous call and hence the compilation error.

2.1.58 Answer: B

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
    (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

Please note that file name can be any name supported by underlying operating system, hence super.java is a valid file name.
Class names can be any name following Java identifier naming rules:
- It should with a letter, the dollar sign "$", or the underscore character "_".
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- It must not be a java keyword.

Hence multiple underscores '_________' is a valid class name.

The effect of `java super.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java super.java` is equivalent to (but not exactly same as):
javac -d <memory> super.java
java -cp <memory> ________ _

Hence, the output is: HELP!!!

Please note that source-file can contain package statement and multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

2.1.59 Answer: A

Reason :
A class defined in unnamed module can have missing package statement but a class defined in named module cannot have missing package statement. So, it is allowed to have a class in default package but it is not recommended.

2.1.60 Answer: C

Reason :
You need to keep in mind an important point related to String Concatenation:
If only one operand expression is of type String, then string conversion is performed on the other operand to produce a string at run time.
If one of the operand is null, it is converted to the string "null".
If operand is not null, then the conversion is performed as if by an invocation of the toString method of the referenced object with no arguments; but if the result of invoking the toString method is null, then the string "null" is used instead.

Let's check all the options:

System.out.println(text.repeat(3)); --> text refers to null, hence text.repeat(3) throws NullPointerException at runtime.

System.out.println(null + null + null); --> Compilation error as operator + is undefined for the arguments: null, null

System.out.println(null + "null" + null); --> If either one of the operand is of String type, then operator + acts as concatenation operator.
As multiple operators are used in above expression, so let's add parenthesis first .
= System.out.println((null + "null") + null); //Operator + is left to right associative
= System.out.println(("null" + "null") + null); //null is converted to "null" as it is used in String concatenation
= System.out.println("nullnull" + null);
= System.out.println("nullnull" + "null"); //null is converted to "null" as it is used in String concatenation
= System.out.println("nullnullnull");
It prints the expected output on to the console

System.out.println(text *= 3); --> Compilation error as *= operator is undefined for the arguments: String, int

System.out.println(text += "null".repeat(2));
= System.out.println(text = text + "null".repeat(2));
= System.out.println(text = text + "nullnull"); //"null".repeat(2) returns "nullnull"
[Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".]
= System.out.println(text = "null" + "nullnull"); //As text is null and used in String concatenation, so it is converted to "null"
= System.out.println(text = "nullnullnull");
"nullnullnull" is assigned to variable text and "nullnullnull" is printed on to the console as well.

System.out.println(text + text + text); --> Even though text refers to null but + operator is defined for the arguments: String, String. So, no compilation error.
Also as text is null and used in String concatenation, so it is converted to

"null"
= System.out.println("null" + "null" + "null");
= System.out.println("nullnullnull");
It prints the expected output on to the console

text += null; System.out.println((text.concat(null)));
First statement, `text += null;` is equals to `text = text + null`
As text is of String type, hence given operator behaves as String Concatenation operator and that is why null reference is converted to "null" .
So, first statement translates to: `text = "null" + "null"` and that is why text refers to "nullnull".
Second statement, `System.out.println((text.concat(null)));`
= `System.out.println(("nullnull".concat(null)));`
If you see the source code of concat(String str) it uses str.length(); so invoking length() method on null reference causes NullPointerException.

Out of the 7 given options, only 3 will print expected output on to the console.

2.1.61 Answer: B

Reason :
'private' is most restrictive, then comes 'default (with no access modifier specified)', after that 'protected' and finally 'public' is least restrictive.

2.1.62 Answer: B,D,E

Reason :
abstract methods cannot be declared with private modifier as abstract methods need to be overridden in child classes.
abstract methods can be declared with either public, protected and package (no access modifier) modifier and hence overriding method cannot be declared which private modifier in the child class. That is why getCalories() method in Food and JunkFood classes cannot be declared private.

Access modifier of overriding method should either be same as the access modifier of overridden method or it should be less restrictive than the access modifier of overridden method. Hence below solutions will work:
1. Remove the protected access modifier from the getCalories() method of

Food class: By doing this, both the overridden and overriding methods will have same access modifier (no access modifier)
or
2. Make the getCalories() method of JunkFood class protected: By doing this, both the overridden and overriding methods will have same access modifier (protected)
or
3. Make the getCalories() method of JunkFood class public: By doing this, access modifier of overriding method (which is public) is less restrictive than the access modifier of overridden method (which is protected)

2.1.63 Answer: E,F

Reason :
exports: ✗ A module exports package and requires another module. 'java.se' is a java standard module and not a package. Hence, java compiler complains.

export: ✗ 'export' is not a valid module directive.

import: ✗ 'import' is not a valid module directive.

imports: ✗ 'imports' is not a valid module directive.

requires: ✓ Directive requires expect module name and 'java.se' is a java standard module. Module 'com.display' can read it. `requires java.se;` is a valid directive.

requires transitive: ✓ `requires transitive java.se;` is also a valid directive and it is used for implied readability.
Implied readability is to specify a transitive dependency on another module such that other modules reading your module also read that dependency.
For example, if module C has `requires B;` (C reads B) and module B has `requires A;` (B reads A), then C doesn't read A.
To have the transitive readability, 'requires transitive' directive is used. If module C has `requires B;` (C reads B) and module B has `requires transitive A;` (B reads A [transitive dependency]), then C will also read A.

2.1.64 Answer: B

Reason :
According to Javadoc, replace(CharSequence target, CharSequence replacement) method of String class returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The replacement proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaa" will result in "ba" rather than "ab".

"MISSS".replace("SS", "T"); returns "MITS".

2.1.65 Answer: B

Reason :
Special main method (called by JVM on execution) should be static and should have public access modifier. It also takes argument of String [] type (Varargs syntax String... can also be used).
String [] or String... argument can use any identifier name, even though in most of the cases you will see "args" is used.

File is correctly named as Player.java (name of public class).
Class Player has special main method but main method defined in Greet class is not public and hence it can't be called by JVM. But there is no issue with the syntax hence no compilation error.
java Player Cristiano Ronaldo passes new String [] {"Cristiano", "Ronaldo"} to args of Player.main method. Player.main method invokes Greet.main method with the same argument: new String [] {"Cristiano", "Gosling"}. args[1] is "Ronaldo" hence "Welcome! Ronaldo" gets printed on to the console.

2.1.66 Answer: C

Reason :
Modules were introduced in Java 9 under the project Jigsaw.
Access below link to know more about project Jigsaw:
http://openjdk.java.net/projects/jigsaw/

2.1.67 Answer: B

Reason :
If a particular package needs to be exported to specific module(s), then

qualified exports are used. Syntax of qualified exports is:
exports <package_name> to <comma_separated_list_of_modules>;

Hence to export the package 'com.udayankhattry.members' to 'test' module only, export directive will be:
exports com.udayankhattry.members to test;

and to export the package 'com.udayankhattry.books' to 'bookhouse' and 'onlinestore' modules only, export directive will be:
exports com.udayankhattry.books to bookhouse, onlinestore ;

Note, the keyword is 'exports' and not 'export'.

exports com.udayankhattry.books; => Package 'com.udayankhattry.books' will be exported to all the modules and not only to 'bookhouse' and 'onlinestore' modules.

Also note that having multiple exports directive for same package is note allowed, below code causes compilation error:
exports com.udayankhattry.books to bookhouse;
exports com.udayankhattry.books to onlinestore;

Therefore correct option is:
exports com.udayankhattry.books to bookhouse, onlinestore;
exports com.udayankhattry.members to test;

2.1.68 Answer: C

Reason :
Any RuntimeException can be thrown without any need it to be declared in throws clause of surrounding method.

`throw (RuntimeException)e;` doesn't cause any compilation error.

Even though variable 'e' is type casted to RuntimeException but exception object is still of ArithmeticException, which is caught in main method and 'AE' is printed to the console.

2.1.69 Answer: C

Reason :
Lambda expression is defined correctly but test() method is not invoked on obj reference. So, no output.

2.1.70 Answer: D

Reason :
'profitPercentage' variable of Profitable interface is implicitly public, static and final.
Line n1 defines the instance variable 'profitPercentage' of Business class. There is no error at Line n1 .
Super type reference variable can refer to an instance of Sub type, therefore no issues at Line n2 as well.

Even though correct syntax for accessing interface variable is by using Interface name, such as Profitable.profitPercentage but reference variable also works. obj.profitPercentage doesn't cause any compilation error.
As, obj is of Profitable type, hence obj.profitPercentage points to the 'profitPercentage' variable of Profitable type. Given code compiles successfully and on execution prints 42.0 on to the console.

2.1.71 Answer: D

Reason:
remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false.

Initially list has: [apple, orange, grape, mango, banana, grape].
fruits.remove("grape") removes the first occurrence of "grape" and after the successful removal, list has: [apple, orange, mango, banana, grape].
fruits.remove("grape") returns true, control goes inside if block and executes fruits.remove("papaya");

fruits list doesn't have "papaya", so the list remain unchanged. In the console, you get: [apple, orange, mango, banana, grape].

2.1.72 Answer: A

Reason :
Few things to keep in mind:
1.
There are 2 rules related to return types of overriding method:
A. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

B. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

2 .
In case of overriding, if overridden method declares to throw any RuntimeException or Error, overriding method may or may not throw any RuntimeException but overriding method must not throw any checked exceptions.

3.
In generics syntax, Parameterized types are not polymorphic, this means even if B is subtype of A, List<B> is not subtype of List<A>. Remember this point. So below syntaxes are NOT allowed:
List<A> list = new ArrayList<B>(); OR ArrayList<A> list = new ArrayList<B>();

Let's check all the options one by one:
abstract List<A> get() throws ArrayIndexOutOfBoundsException; => ✓
It returns the same return type 'List<A>' and it is allowed to throw any RuntimeException (ArrayIndexOutOfBoundsException is RuntimeException)

abstract List<B> get(); => ✗ List<B> is not subtype of List<A>, it is not covariant return type.

abstract ArrayList<A> get() throws Exception; => ✗ As overridden method declares to throw IndexOutOfBoundsException, which is a Runtime Exception, overriding method is not allowed to declare to throw any checked Exception. Class Exception and its subclasses are checked exceptions.

abstract ArrayList<B> get(); => ✗ ArrayList<B> is not subtype of List<A>, it is not covariant return type.

2.1.73 Answer: C

Reason :
Basic/Regular for loop has following form:
for ([ForInit] ; [Expression] ; [ForUpdate]) {...}

[ForInit] can be local variable initialization or the following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpressio n
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

[ForUpdate] can be following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpression
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for [ForInit], `String [] s = {"A", "B", "C", "D"};` is used, which is a local variable initialization statement and hence a valid statement. For [ForUpdate], `res = String.join(".", s)` is used, which is a assignment statement and hence valid as well. Line n1 compiles fine.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g. ,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

As compiler is aware that given infinite loop has an exit point, hence it doesn't mark Line n2 as unreachable. Line n2 doesn't cause any compilation error as well.

Let's check the iterations:
Initially, i = 0 and res = null.
1st iteration: s refers to a String array of 4 elements: {"A", "B", "C", "D"}. Boolean expression is missing, hence it is considered as true. Control enters loop's body.
`i++ == 0` evaluates to true and i is incremented by 1, i = 1. continue;

statement takes the control to [ForUpdate] expression.
2nd iteration: res = String.join(".", s) is evaluated as res = "A.B.C.D".
Boolean expression is missing, hence it is considered as true. Control enters loop's body.
`i++ == 0` evaluates to true and i is incremented by 1, i = 2. break; statement takes the control out of the for loop.
Line n2 is executed and prints A.B.C.D on to the console.

2.1.74 Answer: H

Reason :
Line n1 correctly declares an Object array. Line n1 doesn't cause any compilation error.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name, package name or loop's label but it cannot be used as a class or interface name.
At Line n2, variable 'obj' infers to Object type, so no issues at Line n2.

if-else block uses break; and continue; statements. break; will exit the loop and will take the control to Line n4 on the other hand continue; will take the control to Line n2. In both the cases Line n3 will never be executed.
As Compiler knows about it, hence it tags Line n3 as unreachable, which causes compilation error.

2.1.75 Answer: C

Reason :
As expression contains + operator only, which is left to right associative.
Let us group the expression.
"Password" + 1 + 2 + 3 + 4
= ("Password" + 1) + 2 + 3 + 4
= (("Password" + 1) + 2) + 3 + 4

= ((("Password" + 1) + 2) + 3) + 4
[Let us solve it now, + operator with String behaves as concatenation operator.]
= (("Password1" + 2) + 3) + 4
= ("Password12" + 3) + 4
= "Password123" + 4
= "Password1234"

2.1.76 Answer: B

Reason :
class A is declared public and defined in com.udayankhattry.ocp1 package, there is no problem in accessing class A outside com.udayankhattry.ocp1 package.
class B is defined in com.udayankhattry.ocp1.test package, to extend from class A either use import statement `import com.udayankhattry.ocp1.A;` or fully qualified name of the class 'com.udayankhattry.ocp1.A'. No issues with this class definition as well.

Variable i1 is declared public in class A, so Line 8 doesn't cause any compilation error. Variable i2 is declared protected so it can only be accessed in subclass using using inheritance but not using object reference variable. obj.i2 causes compilation error.

class B inherits variable i2 from class A, so inside class B it can be accessed by using either this or super. Line 10 and Line 11 don't cause any compilation error.

2.1.77 Answer: E

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name. But var type cannot be used as method parameters or method return type. Hence 'var add(int v1, int v2)' causes compilation error.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

2.1.78 Answer: B

Reason :
Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Hence, there is no compilation error at Line n1 .

A module is a set of packages and all the classes should be part of some package (not the default package). Given code causes compilation error as Test class is not defined in the package.

```
C:\>javac -d mods --module-source-path codes -m mymodule
codes\mymodule\Test.java:1: error: unnamed package is not allowed in named modules
public class Test {
^
1 error
```

2.1.79 Answer: B

Reason :
Let' suppose:
At Line n1, ArrayList object is stored at [15EE00].
At Line n2, Integer object created through auto-boxing is stored at [25AF06].

original contains memory address of above Integer object. [15EE00] --> {25AF06}

Now original.clone() will create a new ArrayList object, suppose at [45BA12] and then it will copy the contents of the ArrayList object stored at [15EE00].
So, cloned contains memory address of the same Integer object.
[45BA12] --> {25AF06}

In this case, original != cloned, but original.get(0) == cloned.get(0). This means both the array lists are created at different memory location but refer to same Counter object.

Integer i1 = cloned.get(0) means, i1 also stores [25AF06]. ++i1; means i1 now refers to another Integer object at different memory location, say [38AB00] and [38AB00] refers to [Integer(11)].

i1 is not assigned back to the list so cloned list stays intact and still contains {25AF06} and in the output you get [10].

2.1.80 Answer: B

Reason :
As both the classes: Square and TestSquare are in the same file, hence variables 'length' and 'sq' can be accessed using dot operator. Given code compiles successfully.

Line n1 creates an instance of Square class and 'sq1' refers to it.
sq1.length = 10 and sq1.sq = null.
Line n2 creates an instance of Square class and 'sq2' refers to it.
sq2.length = 5 and sq2.sq = null.

On execution of Line n3, sq1.sq = sq2.

Line n4: System.out.println(sq1.sq.length); =>
System.out.println(sq2.length); => Prints 5 on to the console.

3 Practice Test-3

3.1 80 Questions covering all topics.

3.1.1 Consider below code of Child.java file:

```
class Parent {
    public static void main(String [] args) {
       System. out .println( "Parent" );
    }
}

class Child extends Parent {
    public static void main(String [] args) {
       System. out .println( "Child" );
    }
}
```

And the command:

```
java Child.java
```

What is the result?

A.	Parent	B.	Child
C.	Parent Child	D.	Child Parent
E.	Above command causes error		

3.1.2 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       Error obj = new Error();
        boolean flag1 =
              obj instanceof RuntimeException; //Line n1
        boolean flag2 =
              obj instanceof Exception; //Line n2
        boolean flag3 =
            obj instanceof Error; //Line n3
        boolean flag4 =
            obj instanceof Throwable; //Line n4
        System. out .println(flag1 + ":" + flag2 + ":"
            + flag3 + ":" + flag4);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. false:false:true:true

C. false:true:true:true

D. true:true:true:true

E. false:false:true:false

3.1.3 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var arr = new int [x][y]; //Line n1
        arr[ 1 ][ 4 ] = 100 ;
       arr[ 6 ][ 6 ] = 200 ;
       arr[ 3 ][ 6 ] = 300 ;
    }
}
```

And below combination of x and y values:

1. x = 6, y = 6
2. x = 2, y = 5
3. x = 4, y = 7
4. x = 7, y = 7
5. x = 8, y = 8
6. x = 0, y = 0
7. x = -1, y = -1

How many of above x,y pair(s) can replace x and y at Line n1

such that Test.java file compiles successfully?

A. All 7 pairs
B. 6 pairs
C. 5 pairs
D. 4 pairs
E. 3 pairs
F. 2 pairs
G. 1 pair

3.1.4 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        boolean b1 = 0 ;
        boolean b2 = 1 ;
        System. out .println(b1 + b2);
    }
}
```

What is the result of compiling and executing Test class?

A. 0
B. 1
C. true
D. false
E. Compilation error

3.1.5 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

abstract class Animal {}
class Dog extends Animal{}
```

```
public class Test {
    public static void main(String [] args) {
      List<Animal> list = new ArrayList<Dog>();
      list.add( 0 , new Dog());
      System. out .println(list.size() > 0 );
   }
}
```

What will be the result of compiling and executing Test class?

A. true

B. false

C. Compilation error

D. Runtime exception

3.1.6 Consider below code of Test. java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String [][] arr = { { "%" , "$$" },
            { "***" , "@@@@" , "#####" }};
       for (String [] str : arr) {
          for (String s : str) {
            System. out .println(s);
             if (s.length() == 4 ) //Line n1
                break ; //Line n2
          }
          break ; //Line n3
       }
    }
}
```

What will be the result of compiling and executing Test class?

A.	%	B.	% $$
C.	%	D.	%

	$$ ***		$$ *** @@@@
E.	% $$ *** @@@@ #####		

3.1.7 Which of the following are Java Exception classes? Select 3 options.

A. ClassCastException

B. NullException

C. NumberFormatException

D. IllegalArgumentException

E. ArrayIndexException

3.1.8 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        final boolean flag = false ; //Line n1
        while (flag) {
            System. out .println( "Good Morning!" ); //Line n2
        }
    }
}
```

Which of the following statements is correct for above code?

A. Program compiles and executes successfully but produces no output

B. Compilation error

C. Infinite loop

D. It will print "Good Morning!" once

3.1.9 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Formatter {
   public abstract String format(String s1, String s2);
}

public class Test {
    public static void main(String[] args) {
       Formatter f1 = (str1, str2) ->
             str1 + "_" + str2.toUpperCase();
       System. out .println(f1.format( "Udayan" , "Khattry" ));
    }
}
```

What will be the result of compiling and executing Test class?

A. Udayan_Khattry

B. UDAYAN_Khattry

C. Udayan_KHATTRY

D. UDAYAN_KHATTRY

3.1.10 Given code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
       args[ 1 ] = "Day!" ;
       System. out .println(args[ 0 ] + " " + args[ 1 ]);
    }
}
```

And the commands:

```
javac Test.java
java Test Good
```

What is the result?

A. Good
B. Good Day!
C. Compilation Error
D. An exception is thrown at runtime

3.1.11 Consider below code of Test.java file:

```
//Test.java
package com.udayankhattry.ocp1;

class Parent {
    public String toString() {
        return "Inner " ;
    }
}

class Child extends Parent {
    public String toString() {
        return super .toString().concat( "Peace!" );
    }
}

public class Test {
    public static void main(String[] args) {
        System. out .println( new Child());
    }
}
```

What will be the result of compiling and executing Test class?

A. Inner
B. Peace!
C. Inner Peace!
D. Compilation error

3.1.12 super keyword in java is used to:

A. refer to the static variable of the class
B. refer to the static method of the class

C. refer to the object of current class

D. refer to the object of parent class

3.1.13 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        for ( var var = 0 ; var <= 2 ; var++){} //Line n1
        System. out .println(var); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. 0

B. 2

C. 3

D. Compilation error at Line n1

E. Compilation error at Line n2

3.1.14 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String sport = "swimming" ;
        switch (sport) {
            default :
                System. out .println( "RUNNING" );
            case "Tennis" :
                System. out .println( "TENNIS" );
            case "Swimming" :
                System. out .println( "SWIMMING" );
            case "Football" :
                System. out .println( "FOOTBALL" );
                 break ;
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	RUNNING	B.	SWIMMING
C.	SWIMMING FOOTBALL	D.	RUNNING TENNIS SWIMMING FOOTBALL

3.1.15 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util. ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<Integer> list = new ArrayList<>();
       list.add( 100 );
       list.add( 7 );
       list.add( 50 );
       list.add( 17 );
       list.add( 10 );
       list.add( 5 );

          list.removeIf(a -> a % 10 == 0 );

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [100, 7, 50, 17, 10, 5]

B. [100, 50, 10]

C. [7, 17, 5]

D. Compilation error

E. Runtime Exception

3.1.16 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String... args) {
      System. out .println( "Java 11 is awesome!!!" );
    }
}
```

Location of Test.java file:

```
F:
└───codes
    ├───src
    │   └───com
    │       └───udayankhattry
    │           └───ocp1
    │                   Test.java
    │
    └───classes
```

You are currently at 'codes' folder.

F:\codes>

Which of the following javac commands, typed from above location, will generate Test.class file structure under 'classes' directory?

```
F:
└───codes
    ├───src
    │   └───com
    │       └───udayankhattry
    │           └───ocp1
    │                   Test.java
    │
    └───classes
```

```
└───────com
    └───────udayankhattry
        └───────ocp1
             Test.class
```

A.	javac classes\ src\com\udayankhattry\ocp1\Test.java
B.	javac -d classes\ src\com\udayankhattry\ocp1\Test.java
C.	javac -d classes\ Test.java
D.	javac -d classes\ com.udayankhattry.ocp1.Test.java
E.	javac -d classes\ src\com.udayankhattry.cop1.Test.java
F.	javac -d classes\ src\Test.java

3.1.17 Consider below code of Counter. java file:

```
package com.udayankhattry.ocp1;

public class Counter {
    int count ;

     private static void increment(Counter counter) {
       counter. count ++;
    }

     public static void main(String [] args) {
       Counter c1 = new Counter();
       Counter c2 = c1;
       Counter c3 = null ;
       c2. count = 1000 ;
        increment (c2);
    }
}
```

On executing Counter class, how many Counter objects are created in the memory?

A. 1

B. 2

C. 3

D. 4

3.1.18 Platform module ___________, is the most important module and every Java module requires it implicitly.

A. java.se
B. java.base
C. java.lang
D. java.compiler
E. java.logging

3.1.19 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int m = 20 ;
        int var = --m * m++ + m-- - --m;
        System. out .println( "m = " + m);
        System. out .println( "var = " + var);
    }
}
```

What will be the result of compiling and executing Test class?

A.	m = 25
	var = 363
		B.	m = 363
			var = 363

C.	m = 18
	var = 363
		D.	Compilation error

3.1.20 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class A {
    public String toString() {
        return null ;
    }
}

public class Test {
    public static void main(String [] args) {
        String text = null ;
        text = text + new A(); //Line n1
         System. out .println(text.length()); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n1 causes Runtime error

C. Line n2 causes Runtime error

D. 0

E. 4

F. 8

3.1.21 On Windows platform below directories/files are available:

```
C:
+---source
|   \---com.udayankhattry.modularity
|       |  module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---modularity
|                       Main.java
|
+---classes
|   \---com.udayankhattry.modularity
|       |  module-info.class
|       |
|       \---com
|           \---udayankhattry
|           \---ocp1
|                   Main.class
|
\---jars
```

Contents of module-info.java file:

```
module com.udayankhattry.modularity {
}
```

Contents of Main.java file:

```
package com.udayankhattry.ocp1;

public class Main {
    public static void main(String... args) {
      System. out .println( "MODULAR APPLICATION" );
    }
}
```

Which of the following jar commands executed from C:\, will package the above module such that the execution of the command:

```
java -p jars -m com.udayankhattry.modularity
```

prints MODULAR APPLICATION on to the console?

A.	jar jars\test.jar com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
B.	jar -cf jars\test.jar -C classes\com.udayankhattry.modularity .
C.	jar -cfe jars\test.jar com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
D.	jar jars\test.jar -C classes\com.udayankhattry.modularity .

3.1.22 Your Vendor has provided you a modular jar to test the database connection API.

Below are the Jar details:

Jar File name: dbconnection.jar
Module name: dbconnection
Module-descriptor of this jar contains:

```
module dbconnection {
   exports com.database.util to dbtester;
}
```

You have to write the module and test code, which of the following is correct module descriptor?
Please note that commented line in each option represents the module descriptor file name.

A.	*//module-info.java*
	module dbtester {
	    requires dbconnection;
	}

B.	*//module-info.java*
	module dbtester {
	    requires com.database.util;
	}

C.	*//module-info.java*
	module mymodule {
	    requires dbconnection;
	}

D.	*//module-info.java*
	module mymodule {
	    requires com.database.util;
	}

E.	*//module-desc.java*
	module dbtester {
	    requires dbconnection;
	}

F.	*//module-info.java*
	module dbtester {
	}

3.1.23 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java. util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add( "P" );
        list.add( "O" );
        list.add( "T" );

          List<String> subList =
               list.subList( 1 , 2 ); //Line n1
         subList.set( 0 , "E" ); //Line n2
         System. out .println(String. join ( "" , list));
    }
}
```

What will be the result of compiling and executing Test class?

A. PET

B. POT

C. Compilation error

D. An exception is thrown by Line n2

3.1.24 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    static String var = "FRIENDS" ; //Line n1
    public static void main(String[] args) {
        int var = (var = Test. var .length()); //Line n2
        System. out .println(var); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. It compiles successfully and on execution prints FRIENDS on to the console

E. It compiles successfully and on execution prints 7 on to the console

3.1.25 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      StringBuilder[] sb =
            new StringBuilder[ 2 ]; //Line n1
       sb[ 0 ] = new StringBuilder( "PLAY" ); //Line n2

         boolean [] flag = new boolean [ 1 ]; //Line n3

         if (flag[ 0 ]) { //Line n4
          sb[ 1 ] = new StringBuilder( "GROUP" ); //Line n5
        }

         System. out .println(String. join ( "-" , sb)); //Line n6
    }
}
```

What will be the result of compiling and executing Test class?

A. PLAY

B. GROUP

C. PLAY-GROUP

D. PLAY-null

E. -

F. Compilation error

G. An exception is thrown at runtime

3.1.26 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> places = new ArrayList<>();
        places.add( "Austin" );
        places.add( "Okinawa" );
        places.add( "Giza" );
        places.add( "Manila" );
        places.add( "Batam" );
        places.add( "Giza" );

           if (places.remove( "Giza" ))
            places.remove( "Austin" );

           System. out .println(places);
    }
}
```

What will be the result of compiling and executing Test class?

A. An exception is thrown at runtime

B. Compilation error

C. [Okinawa, Manila, Batam]

D. [Austin, Okinawa, Giza, Manila, Batam, Giza]

E. [Austin, Okinawa, Manila, Batam, Giza]

F. [Austin, Okinawa, Manila, Batam]

G. [Okinawa, Manila, Batam, Giza]

H. [Okinawa, Giza, Manila, Batam]

3.1.27 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Parent {
    int var = 1000 ; // Line n1

     int getVar() {
         return var ;
    }
}

class Child extends Parent {
    private int var = 2000 ; // Line n2
    int getVar() {
       return super . var ; //Line n3
    }
}

public class Test {
    public static void main(String[] args) {
       Child obj = new Child(); // Line n4
        System. out .println(obj. var ); // Line n5
    }
}
```

There is a compilation error in the code.
Which three modifications, done independently, print 1000 on to the console?

A. Change Line n1 to private int var = 1000;

B. Delete the Line n2

C. Change Line n3 to return var;

D. Change Line n4 to Parent obj = new Child();

E. Delete the method getVar() from the Child class

F. Change Line n5 to System.out.println(obj.getVar());

3.1.28 Consider below codes of 3 java files:

```
//Animal.java
package a;

public class Animal {
   Animal() {
      System. out .print( "ANIMAL-" );
   }
}


//Dog.java
package d;

import a.Animal;

public class Dog extends Animal {
    public Dog() {
        System. out .print( "DOG" );
    }
}

//Test.java
package com.udayankhattry.ocp1;

import d.Dog;

public class Test {
    public static void main(String[] args) {
        new Dog();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in Animal.java file

B. Compilation error in Dog.java file

C. Compilation error in Test.java file

D. It executes successfully and prints ANIMAL-DOG on to the console

E. It executes successfully and prints DOG on to the console

F. It executes successfully but nothing is printed on to the console

3.1.29 Given the code fragment:

```
String txt = "an" ;
System. out .println(txt.split( "an" ). length );
```

What is the result?

A. 0

B. 1

C. 2

D. 3

E. Compilation error

3.1.30 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       System. out .println( 1 + 2 + 3 + 4 + "Running" );
    }
}
```

What will be the result of compiling and executing Test class?

A. 10Running

B. 1234Running

C. 64Running

D. 10 Running

3.1.31 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        try {
            check ();
        } catch (RuntimeException e) {
            System. out .println(e.getClass()
                .getName()); //Line n1
        }
    }

    private static void check() {
        try {
            RuntimeException re
                = new RuntimeException(); //Line n2
            throw re; //Line n3
        } catch (RuntimeException e) {
            System. out .println( 1 );
            ArithmeticException ex
                = (ArithmeticException)e; //Line n4
            System. out .println( 2 );
            throw ex;
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	1 2 java.lang.RuntimeException
B.	1 2 java.lang.ArithmeticException
C.	1 java.lang.ArithmeticException
D.	1 java.lang.RuntimeException
E.	1 java.lang.ClassCastException

3.1.32 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class SpecialString {
    String str ;
    SpecialString(String str) {
        this . str = str;
    }
}

public class Test {
     public static void main(String[] args) {
        System. out .println( new String(
                "Imperfectly Perfect" ));
        System. out .println( new StringBuilder(
                "Imperfectly Perfect" ));
        System. out .println( new SpecialString(
                "Imperfectly Perfect" ));
    }
}
```

What will be the result of compiling and executing Test class?

A.	Imperfectly Perfect Imperfectly Perfect Imperfectly Perfect
B.	Imperfectly Perfect Imperfectly Perfect <Some text containing @ symbol>
C.	Imperfectly Perfect <Some text containing @ symbol> <Some text containing @ symbol>
D.	Compilation error

3.1.33 Consider below code of TestMessage.java file:

```
package com.udayankhattry.ocp1;

class Message {
    String msg = "LET IT GO!" ;

     public void print() {
        System. out .println( msg );
    }
}

public class TestMessage {
     public static void change(Message m) { //Line n5
        m. msg = "NEVER LOOK BACK!" ; //Line n6
     }
     public static void main(String[] args) {
        Message obj = new Message(); //Line n1
        obj.print(); //Line n2
        change (obj); //Line n3
        obj.print(); //Line n4
     }
}
```

What will be the result of compiling and executing TestMessage class?

A.	null NEVER LOOK BACK!
B.	NEVER LOOK BACK! NEVER LOOK BACK!
C.	null null
D.	LET IT GO! NEVER LOOK BACK!
E.	LET IT GO! LET IT GO!
F.	Compilation error

3.1.34 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

interface I1 {
    public static void print(String str) {
      System. out .println( "I1:" + str.toUpperCase());
    }
}

class C1 implements I1 {
    void print(String str) {
       System. out .println( "C1:" + str.toLowerCase());
    }
}

public class Test {
    public static void main(String[] args) {
       I1 obj = new C1();
       obj. print ( "Java" );
    }
}
```

Which of the following statements is correct?

A. Class C1 causes compilation error

B. Class Test causes compilation error

C. Interface I1 causes compilation error

D. Given code compiles successfully and on execution prints I1:JAVA on to the console

E. Given code compiles successfully and on execution prints C1:java on to the console

3.1.35 Consider below code of Test.java file:

```
class Test {
    public static void main(String [] args) {
        final var x = 10 ; //Line n1
        byte b = x + 10 ; //Line n2
        System. out .println(b);
    }
}
```

Which of the following commands will print 20 on to the console? Select ALL that apply.

A.	Code causes compilation error and hence it will not execute
B.	java Test.java
C.	java --source 11 Test.java

D.	java --source 10 Test.java
E.	java --source 9 Test.java
F.	java --source 8 Test.java

3.1.36 Given below the directory/file structure on Windows platform:

```
C:
+---codes
|   \---currency
|       |  module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                       Currency.java
|
\---classes
```

Below are the file contents:-

C:\codes\currency\module-info.java:

```
module currency {
}
```

C:\codes\currency\com\udayankhattry\ocp1\Currency.java:

```
package com.udayankhattry.ocp1;

public class Currency {
    //Lots of valid codes
}
```

And consider the command to be executed from C:\

```
javac -d classes _________________ codes -m currency
```

Which of the options can be used to fill above blank such that there is no error on executing above command?

A. --module-path

B. --path

C. --source-code-path

D. --module-source-path

E. --source

F. -p

3.1.37 Given the following definitions of the class Insect and the interface Flyable, the task is to declare a class Mosquito that inherits from the class Insect and implements the interface Flyable.

```
class Insect {}
interface Flyable {}
```

Select the correct option to accomplish this task:

A	. **class** Mosquito **implements** Insect **extends** Flyable{}
B	. **class** Mosquito **implements** Insect, Flyable{}
C	. **class** Mosquito **extends** Insect, Flyable{}
D	. **class** Mosquito **extends** Insect **implements** Flyable{}

3.1.38 Consider below code of TestBook.java file:

```
package com.udayankhattry.ocp1;

class Book {
    private String name ;
    private String author ;

     Book() {}

     Book(String name, String author) {
       name = name;
       author = author;
    }

     String getName() {
        return name ;
    }
    String getAuthor() {
       return author ;
    }
}

public class TestBook {
    public static void main(String[] args) {
       private Book book = new Book(
            "Head First Java" , "Kathy Sierra" );
       System. out .println(book.getName());
       System. out .println(book.getAuthor());
    }
}
```

What will be the result of compiling and executing above code?

A.	Compilation error in Book class
B.	Compilation error in TestBook class
C.	null null
D.	Head First Java Kathy Sierra

3.1.39 Which of the following is the correct package declaration to declare Exam class in com.university.sem1 package?

A.	package com.university.sem1.Test;
B.	package com.university.sem1;
C.	package com.university.sem1.*;
D.	Package com.university.sem1;

3.1.40 Consider the code of TestEmployee.java file:

```
package com.udayankhattry.ocp1;

class Employee {
    String name ;
    int age ;

    Employee() {
      Employee( "Michael" , 22 ); //Line n1
    }

    Employee(String name, int age) {
        this . name = name;
        this . age = age;
    }
}

public class TestEmployee {
    public static void main(String[] args) {
      Employee emp = new Employee();
      System. out .println(emp. name + ":" + emp. age ); //Line n2
    }
}
```

What will be the result of compiling and executing TestEmployee class?

A. Compilation error at Line n1

B. Compilation error at Line n2

C. null:0

D. Michael:22

E. An exception is thrown at runtime

3.1.41 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.sql.SQLException;

public class Test {
    private static void m() throws SQLException {
        try {
            throw new SQLException();
        } catch (Exception e) {
            throw e;
        }
    }

    public static void main(String[] args) {
        try {
            m ();
        } catch (SQLException e) {
            System. out .println( "CAUGHT SUCCESSFULLY" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Method m() causes compilation error

B. Method main(String []) causes compilation error

C. CAUGHT SUCCESSFULLY is printed on to the console and program terminates successfully

D. Program ends abruptly

3.1.42 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String str = "Think" ; //Line n3
         change (str); //Line n4
         System. out .println(str); //Line n5
    }

    private static void change(String s) {
        s.concat( "_Twice" ); //Line n9
    }
}
```

What will be the result of compiling and executing Test class?

A. Think

B. _Twice

C. Think_Twice

D. None of the other options

3.1.43 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      StringBuilder sb = new StringBuilder( "TOMATO" );
      System. out .println(sb.reverse()
              .replace( "O" , "A" )); //Line n1
    }
}
```

What will be the result of compiling and executing Test class?

A. TOMATO

B. TAMATO

C. TAMATA

D. OTAMOT

E. OTAMAT

F. ATAMAT

G. Compilation error

3.1.44 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

/*INSERT*/

public class Test {
    public static void main(String[] args) {
      MyInterface mi = () -> "" ;
   }
}
```

Which of the following interface definitions can replace

/*INSERT*/ such that there is no compilation error?
Select ALL that apply .

A.

```
interface MyInterface {
    String toStr();
}
```

B.

```
interface MyInterface {
    String toStr();
    String toString();
}
```

C.

```
interface MyInterface {
    Object toStr();
    String toString();
     static void print(String str) {
        System. out .println(str);
    }
}
```

D.

```
interface MyInterface {
    Object toStr();
     boolean equals(MyInterface);
}
```

E.

```
interface MyInterface {
    String toString();
}
```

F.

```
interface MyInterface {
    CharSequence test();
}
```

G.

```
interface MyInterface {
    StringBuilder m();
}
```

3.1.45 For the given code:

```
package com. udayankhattry.ocp1;

interface Printable {
    void print(String msg);
}

public class Test {
    public static void main(String[] args) {
        /*INSERT*/
        obj.print( "WINNERS NEVER QUIT" );
    }
}
```

Which of the following options successfully replace /*INSERT*/ such that on execution, WINNERS NEVER QUIT is printed on to the console?
Select ALL that apply.

A.	Printable obj = (String msg) -> {System. *out* .println(msg);};
B.	Printable obj = (String msg) -> {System. *out* .println(msg); **return** ;};
C.	Printable obj = String msg -> {System. *out* .println(msg);};
D.	Printable obj = (msg) -> {System. *out* .println(msg);};
E.	Printable obj = (msg) -> System. *out* .println(msg);
F.	Printable obj = msg -> {System. *out* .println(msg)};
G.	Printable obj = msg -> System. *out* .println(msg);
H.	Printable obj = x -> System. *out* .println(x);
I.	Printable obj = y - > System. *out* .println(y);

3.1.46 Given below the directory/file structure on Windows platform:

```
C:
\---src
    \---com.udayankhattry.utility
        |   module-info.java
```

```
|
\---com
    \---udayankhattry
        \---utility
            |   FileUtility.java
            |   StringUtility.java
            |
            \---internal
                    DateUtility.java
```

Name of the module: com.udayankhattry.utility

and there are three public classes available:
com.udayankhattry.utility.FileUtility
com.udayankhattry.utility.StringUtility
com.udayankhattry.utility.internal.DateUtility

Currently module-info.java file is blank, which of the following options correctly describes the content of module-info.java file such that FileUtility and StringUtility classes are available for public use but not the DateUtility class ?

A.

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility.*;
}
```

B.

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility;
}
```

C.

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility.FileUtility;
    exports com.udayankhattry.utility.StringUtility;
}
```

D.

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility.internal;
}
```

E.

```
module com.udayankhattry.utility {
}
```

F.

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility.FileUtility;
    exports com.udayankhattry.utility.StringUtility;
    not exports com.udayankhattry.utility.internal.DateUtility;
}
```

G.

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility;
    not exports com.udayankhattry.utility.internal;
}
```

3.1.47 What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        byte b1 = ( byte ) ( 127 + 21 );
        System. out .println(b1);
    }
}
```

A. 148

B. Compilation error

C. -108

D. 128

3.1.48 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.*;

public class Test {
    public static void main(String[] args) {
      String [] arr = { "EARTH" , "MOON" , "SUN" , "PLUTO" };
       var list = /*INSERT*/ ;
      list.set( 3 , "JUPITER" ); //Line n1
       list.forEach(str -> System. out .println(str)); //Line n2
    }
}
```

Which of the following options can replace /*INSERT*/ such that output is:

```
EARTH
MOON
SUN
JUPITER
```

Select ALL that apply.

A.	List.of(arr)
B.	Arrays.asList(arr)
C.	new ArrayList<>(List.of(arr))
D.	ArrayList.of(arr)
E.	new ArrayList<>(ArrayList.of(arr))

3.1.49 In a modular graph, [POSITION-1] are represented by nodes and [POSITION-2] between the modules are represented by arrows/edges.

Which of the following provide correct replacement for [POSITION-1] and [POSITION-2]?

A. Replace [POSITION-1] with 'classes' and [POSITION-2] with 'dependencies'

B. Replace [POSITION-1] with 'dependencies' and [POSITION-2] with 'classes'

C. Replace [POSITION-1] with 'dependencies' and [POSITION-2] with 'packages'

D. Replace [POSITION-1] with 'packages' and [POSITION-2] with 'dependencies'

E. Replace [POSITION-1] with 'modules' and [POSITION-2] with 'dependencies'

F. Replace [POSITION-1] with 'dependencies' and [POSITION-2] with 'modules'

3.1.50 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface M {
    public static void log() {
       System. out .println( "M" );
    }
}

abstract class A {
    public static void log() {
       System. out .println( "N" );
    }
}

class MyClass extends A implements M {}

public class Test {
    public static void main(String[] args) {
       M obj1 = new MyClass();
       obj1. log (); //Line n1

          A obj2 = new MyClass();
       obj2. log (); //Line n2

          MyClass obj3 = new MyClass();
       obj3. log (); //Line n3
     }
}
```

Which of the following statements is correct?

A. There is a compilation error in interface M

B. There is a compilation error in class A

C. Line n1 causes compilation error

D. Line n2 causes compilation error

E. Line n3 causes compilation error

F. Given code compiles successfully

3.1.51 Consider below codes of 4 java files:

```
//I1.java
package com.udayankhattry.ocp1;

public interface I1 {
    int i = 10 ;
}


//I2.java
package com.udayankhattry.ocp1;

public interface I2 {
    int i = 20 ;
}


//I3.java
package com.udayankhattry.ocp1;

public interface I3 extends I1, I2 { //Line n1

}


//Test.java
package com.udayankhattry.ocp1;

public class Test {
     public static void main(String[] args) {
        System. out .println(I1.i); //Line n2
         System. out .println(I2.i); //Line n3
         System. out .println(I3.i); //Line n4
     }
}
```

Which of the following statements is correct?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Line n4 causes compilation error

E. There is no compilation error

3.1.52 Which one of these top level classes cannot be sub-classed?

A. class Dog {}

B. abstract class Cat {}

C. final class Electronics {}

D. private class Car {}

3.1.53 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        var val = 9 ;
       System. out .println(val += 10 - -val-- - --val);
    }
}
```

What will be the result of compiling and executing Test class?

A. 21

B. 22

C. 23

D. 24

E. 25

F. 26

G. Compilation error

3.1.54 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
        var x = 10 ; //Line n1
        if ( false )
            System. out .println(x); //Line n2
        System. out .println( "HELLO" ); //Line n3
    }
}
```

What is the result of compiling and executing Test class?

A.	Compilation error at Line n1	B.	Compilation error at Line n2
C.	Compilation error at Line n3	D.	HELLO
E.	10 HELLO		

3.1.55 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    static int i1 = 10 ;
    int i2 = 20 ;

    int add() {
        return this . i1 + this . i2 ; //Line n1
    }

    public static void main(String[] args) {
        System. out .println( new Test().add()); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. It executes successfully and prints 30 on to the console

B. It executes successfully and prints 20 on to the console

C. It executes successfully and prints 10 on to the console

D. Line n1 causes compilation error

E. Line n2 causes compilation error

3.1.56 Consider the following interface declaration:

```
public interface I1 {
   void m1() throws java.io.IOException;
}
```

Which of the following incorrectly implements interface I1?

A.

```
public class C1 implements I1 {
   public void m1() {}
}
```

B.

```
public class C2 implements I1 {
   public void m1() throws java.io.FileNotFoundException{}
}
```

C.

```
public class C3 implements I1 {
   public void m1() throws java.io.IOException{}
}
```

D.

```
public class C4 implements I1 {
   public void m1() throws Exception{}
}
```

3.1.57 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       int [] arr1 = { 1 , 2 , 3 }; //Line n1
       int [] arr2 = { 'A' , 'B' }; //Line n2
       arr1 = arr2; //Line n3
```

```
        for ( int i = 0 ; i < arr1. length ; i++) {
          System. out .print(arr1[i] + " " ); //Line n4
        }
    }
}
```

ASCII code of 'A' is 65 and 'B' is 66. What will be the result of compiling and executing Test class?

A. 1 2 3

B. A B

C. 65 66

D. 0 0

E. Compilation error

3.1.58 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.function.Predicate;

class Furniture {
    private String name ;
    private double weight ;
    private double price ;

    public Furniture(String name, double weight,
               double salary) {
        this . name = name;
        this . weight = weight;
        this . price = price ;
    }

    public String getName() {
        return name ;
    }

     public double getWeight() {
        return weight ;
    }

     public double getPrice() {
        return price ;
    }

     public String toString() {
        return name ;
    }
}

public class Test {
    public static void main(String [] args) {
       List<Furniture> list = new ArrayList<>();
       list.add( new Furniture( "Chair" , 10.2 , 40 ));
       list.add( new Furniture( "Table" , 23.7 , 300 ));
       list.add( new Furniture( "Sofa" , 41 , 900 ));
       list.add( new Furniture( "Bed" , 45 , 1500 ));
       list.add( new Furniture( "Cabinet" , 51 , 2000 ));

         process (list, f -> f.getWeight() < 45 );
    }

     private static void process(List<Furniture> list,
                 Predicate<Furniture> predicate) {
       Iterator<Furniture> iterator = list.iterator();
        while (iterator.hasNext()) {
           Furniture f = iterator.next();
            if (predicate.test(f))
               System. out .print(f + " " );
       }
    }
}
```

What will be the result of compiling and executing Test class?

A. Chair Table Sofa Bed Cabinet

B. Chair Table Sofa

C. Bed Cabinet

D. Compilation error

3.1.59 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.IOException;
import java.sql.SQLException;

public class Test {
    public static void main(String[] args) {
        /*INSERT*/
    }

    private static void save() throws IOException {}

    private static void log() throws SQLException {}
}
```

Which of the block of codes can be used to replace /*INSERT*/ such that there is no compilation error? Select ALL that apply .

A.

```
try {
    save ();
    log ();
} catch (IOException | SQLException ex) {}
```

B.

```
try {
    save ();
    log ();
} catch (SQLException | IOException ex) {}
```

C.

```
try {
    save ();
    log ();
} catch (IOException | Exception ex) {}
```

D.

```
try {
    save ();
    log ();
} catch (SQLException | Exception ex) {}
```

E.

```
try {
    save ();
    log ();
} catch (Exception | RuntimeException ex) {}
```

F.

```
try {
    save ();
    log ();
} catch (Exception ex) {}
```

3.1.60 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class X {
    void greet() {
      System. out .println( "Good Morning!" );
   }
}

class Y extends X {
    void greet() {
      System. out .println( "Good Afternoon!" );
   }
}

class Z extends Y {
    void greet() {
      System. out .println( "Good Night!" );
    }
}

public class Test {
    public static void main(String[] args) {
      X x = new Z();
      x.greet(); //Line n1
       ((Y)x).greet(); //Line n2
       ((Z)x).greet(); //Line n3
    }
}
```

What will be the result of compiling and executing above code?

A.	Compilation error
B.	An exception is thrown at runtime
C.	It compiles successfully and on execution prints below: Good Night! Good Afternoon! Good Morning!
D.	It compiles successfully and on execution prints below: Good Night! Good Night! Good Night!
E.	It compiles successfully and on execution prints below: Good Morning! Good Morning! Good Morning!

3.1.61 Given below the directory/file structure on Windows platform:

```
C:
+---src
```

```
|   +---com.udayankhattry.gifts
|   |   |   module-info.java
|   |   |
|   |   \---com
|   |       \---udayankhattry
|   |           \---gifts
|   |                   Gift.java
|   |
|   +---com.udayankhattry.game
|   |   |   module-info.java
|   |   |
|   |   \---com
|   |       \---udayankhattry
|   |           \---game
|   |                   Game.java
|   |
|   \---com.udayankhattry.test
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---test
|                       Test.java
|
\---cls
```

File contents below:-

C:\src\com.udayankhattry.gifts\com\udayankhattry\gifts\Gift.java:

```java
package com.udayankhattry.gifts;

import java.util.List;

public class Gift {
    static List<String> gifts = List. of ( "Car" , "Laptop" ,
                            "Smartphone" , "Smartwatch" , "EBook Reader" );

      private String name ;

      Gift(String name) {
         this . name = name;
    }

    public String toString() {
        return name ;
    }

    public static Gift get( int num) {
        return new Gift( gifts .get(num));
    }
}
```

C:\src\com.udayankhattry.gifts\module-info.java:

```
module com.udayankhattry.gifts {
    exports com.udayankhattry.gifts;
}
```

C:\src\com.udayankhattry.game\com\udayankhattry\game\Game.jav

```
package com.udayankhattry.game;

import com.udayankhattry.gifts.Gift;

public class Game {
    public Gift play( int num) {
        if (num < 0 || num > 4 )
            return null ;
        return Gift.get(num);
    }
}
```

C:\src\com.udayankhattry.game\module-info.java:

```
module com.udayankhattry.game {
    /*INSERT*/ com.udayankhattry.gifts;
    exports com.udayankhattry.game;
}
```

C:\src\com.udayankhattry.test\com\udayankhattry\test\Test.java:

```
package com.udayankhattry.test;

import com.udayankhattry.game.Game;
import com.udayankhattry.gifts.Gift;
```

```
public class Test {
    public static void main(String[] args) {
       Gift myGift = new Game().play( 2 );
       System. out .println(myGift);
    }
}
```

C:\src\com.udayankhattry.test\module-info.java:

```
module com.udayankhattry.test {
    requires com.udayankhattry.game;
}
```

And the below command to be executed from C:\

```
javac -d cls --module-source-path src -m com.udayankhattry.test
```

File C:\src\com.udayankhattry.game\module-info.java is incomplete, which of the following options replaces /*INSERT*/ such that given javac command successfully compiles the module code?

A. require
B. requires
C. requires transitive
D. export
E. exports

3.1.62 For the given code snippet:

```
List< String > list = new /*INSERT*/ ();
```

Which of the following options, if used to replace /*INSERT*/, compiles successfully?
Select ALL that apply.

A. List<String>
B. List<>
C. ArrayList<String>
D. ArrayList<>

3.1.63 Consider below code fragment:

```
private void emp() {}
```

And the statements:

1. Given code compiles successfully if it is used inside the class named 'emp'
2. Given code compiles successfully if it is used inside the class named 'Emp'
3. Given code compiles successfully if it is used inside the class named 'employee'
4. Given code compiles successfully if it is used inside the class named 'Employee'
5. Given code compiles successfully if it is used inside the class named 'Student'
6. Given code compiles successfully if it is used inside the class named '_emp_'

How many statements are true?

A. Only one statement

B. Two statements

C. Three statements

D. Four statements

E. Five statements

F. All six statements

3.1.64 Below is the code of Test.java file.

```
package com.udayankhattry.ocp1;

public class Test {
    /* INSERT */
}
```

Below are the definitions of main method:

1.

```
public static final void main(String... a) {
    System. out .println( "Java Rocks!" );
}
```

2.

```
public void main(String... args) {
    System. out .println( "Java Rocks!" );
}
```

3.

```
static void main(String [] args) {
    System. out .println( "Java Rocks!" );
}
```

4.

```
public static void main(String [] args) {
    System. out .println( "Java Rocks!" );
}
```

5.

```
public static void main(String args) {
    System. out .println( "Java Rocks!" );
}
```

How many definitions of main method can replace /* INSERT */ such that on executing Test class, "Java Rocks!" is printed on to the console?

A. Only one definition

B. Only two definitions

C. Only three definitions

D. Only four definitions

E. All 5 definitions

3.1.65 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class Test {
    public static void main(String[] args) {
      List<String> objects = new ArrayList<>();
      objects.add( "Watch" );
      objects.add( "Arrow" );
      objects.add( "Anchor" );
      objects.add( "Drum" );

         ListIterator<String> iterator = objects.listIterator();
       while (iterator.hasNext()) {
          if (iterator.next().startsWith( "A" )) {
            iterator.remove();
          }
      }

         System. out .println(objects);
   }
}
```

What will be the result of compiling and executing Test class?

A. [Watch, Arrow, Anchor, Drum]

B. [Watch, Drum]

C. An exception is thrown at runtime

D. Compilation error

3.1.66 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String str = "ALASKA" ;
      System. out .println(str.charAt(
            str.indexOf( "A" ) + 1 ));
   }
}
```

What will be the result of compiling and executing Test class?

A. A

B. L

C. S

D. K

E. Runtime error

3.1.67 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        /*INSERT*/ x = 7 , y = 200 ;
        System. out .println(String.valueOf(x + y).length());
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print 3 on to the console? Select ALL that apply.

A. byte

B. short

C. int

D. long

E. float

F. double

G. var

3.1.68 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
      List<String> list1 = new ArrayList<>();
      list1.add( "A" );
      list1.add( "D" );

         List<String> list2 = new ArrayList<>();
      list2.add( "B" );
      list2.add( "C" );

         list1.addAll( 1 , list2);

         System. out .println(list1);
    }
}
```

What will be the result of compiling and executing Test class?

A. [A, B, C, D]

B. [A, D, B, C]

C. [A, D]

D. [A, B, C]

3.1.69 Consider below code of Apple.java file:

```
package com.udayankhattry.ocp1;

public class Apple {
    public String color ;

     public Apple(String color) {
        /*INSERT*/
     }

     public static void main(String [] args) {
       Apple apple = new Apple( "GREEN" );
       System. out .println(apple. color );
    }
}
```

For the class Apple, which option, if used to replace /*INSERT*/, will print GREEN on to the console?

A. color = color;
B. this.color = color;
C. color = GREEN;
D. this.color = GREEN;

3.1.70 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String str = " " ; //single space
        boolean b1 = str.isEmpty();
        boolean b2 = str.isBlank();
       System. out .println(b1 + " : " + b2); //Line n1

          str.strip(); //Line n2
        b1 = str.isEmpty();
       b2 = str.isBlank();
       System. out .println(b1 + " : " + b2); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A.	false : true true : true	B.	false : false true : true
C.	true : false true : false	D.	false : true false : true
E.	false : true true : false		

3.1.71 Consider below codes of 4 java files:

```
//A.java
package com.udayankhattry.ocp1;
```

```
public class A {
    public void print() {
        System. out .println( "A" );
    }
}

//B.java
package com.udayankhattry.ocp1;

public class B extends A {
    public void print() {
        System. out .println( "B" );
    }
}

//C.java
package com.udayankhattry.ocp1;

public class C extends A {
    public void print() {
        System. out .println( "C" );
    }
}

//Test.java
package com.udayankhattry.ocp1.test;

import com.udayankhattry.ocp1.*;

public class Test {
    public static void main(String[] args) {
        A obj1 = new C();
        A obj2 = new B();
        C obj3 = (C)obj1;
        C obj4 = (C)obj2;
        obj3.print();
    }
}
```

What will be the result of compiling and executing Test class?

A. It executes successfully and prints A on to the console

B. It executes successfully and prints B on to the console

C. It executes successfully and prints C on to the console

D. Compilation error

E. An exception is thrown at runtime

3.1.72 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Equality {
    boolean equals(Object obj);
}

public class Test {
    public static void main(String[] args) {
        Equality eq = x -> true ;
        System. out .println(eq.equals( null ));
    }
}
```

What will be the result of compiling and executing Test class ?

A. Compilation error

B. An exception is thrown at runtime

C. true

D. null

E. false

3.1.73 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int [] arr = { 2 , 1 , 0 };
        for ( int i : arr) {
            System. out .println(arr[i]);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	2 1 0	B.	0 1 2
C.	Compilation error	D.	An Exception is thrown at runtime

3.1.74 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String [] arr = { "A" , "B" , "C" , "D" , "E" };
        for ( /*INSERT*/ ) {
            System. out .print(arr[n]);
        }
    }
}
```

Which option, if used to replace /*INSERT*/, on execution will print ACE on to the console?

A. int n = 0; n < arr.length; n += 1

B. int n = 0; n < arr.length; n += 2

C. int n = 1; n < arr.length; n += 1

D. int n = 1; n < arr.length; n += 2

3.1.75 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var res = "" ; //Line n1
        String [] arr = { "1" , "2" , "3" };
        for ( var s : arr) { //Line n2
            res += String. join ( "." , s); //Line n3
        }
        System. out .println(res); //Line n4
    }
}
```

What will be the result of compiling and executing Test class?

A. 1.2.3

B. 123

C. 3

D. Compilation error

E. An exception is thrown at runtime

3.1.76 On Windows platform below directories/files are available:

1.
C:\codes\com.testing\module-info.java:

```
module com.testing {
    /*INSERT*/
}
```

2.
C:\codes\com.testing\com\udayankhattry\ocp1\Test.java:

```
package com.udayankhattry.ocp1;

import java.io.IOException;
import java.sql.SQLException;

public class Test {
    public static void main(String[] args) {
        try {
            save ();
            log ();
        } catch (IOException | SQLException ex) {}
    }

    private static void save() throws IOException {}
    private static void log() throws SQLException {}
}
```

3.
C:\modules\

And the command executed from C:

```
javac -d modules --module-source-path codes --module com.testing
```

Which of the following changes enable above command to execute successfully (i.e. modular code to compile successfully)?

A. No changes are necessary, code as is compiles successfully

B. Replace /*INSERT*/ with requires java.io;

C. Replace /*INSERT*/ with requires java.sql;

D. Replace /*INSERT*/ with requires java.*;

E. Replace /*INSERT*/ with requires java.base; java.sql;

3.1.77 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<String> trafficLight = new ArrayList<>();
        trafficLight.add( 1 , "RED" );
        trafficLight.add( 2 , "ORANGE" );
        trafficLight.add( 3 , "GREEN" );

            trafficLight.remove(Integer. valueOf ( 2 ));

            System. out .println(trafficLight);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. [RED, GREEN]

D. [RED, ORANGE]

E. [RED, ORANGE, GREEN]

3.1.78 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       check (); //Line n1
    }

    private static void check() throws Exception { //Line n2
       System. out
               .println( "NOT THROWING ANY EXCEPTION" ); //Line
n3
    }
}
```

What will be the result of compiling and executing Test class ?

A. Compilation error at Line n1

B. Compilation error at Line n2

C. Compilation error at Line n3

D. NOT THROWING ANY EXCEPTION

3.1.79 Which of the following is not a valid array declaration?

A. int [] arr1 = new int[8];

B. int [][] arr2 = new int[8][8];

C. int [] arr3 [] = new int[8][];

D. int arr4[][] = new int[][8];

3.1.80 Consider below code snippet available in the same package:

```
abstract class Traveller {
    void travel(String place){}
}

abstract class BeachTraveller extends Traveller {
   /*INSERT*/
}
```

Which of the following declarations/definitions can replace /*INSERT*/ such that there is no compilation error? Select ALL that apply.

A. abstract void travel();

B. abstract void travel(String beach);

C. public abstract void travel();

D. public void travel() throws RuntimeException {}

E. public void travel(String beach) throws Exception {}

F. void travel(String beach) throws java.io.IOException {}

G. public void travel(Object obj) {}

3.2 Answers of Practice Test - 3 with Explanation

3.1.1 Answer: A

Reason :

Starting with JDK 11, it is possible to launch single-file source-code Programs.

If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
     (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Child.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java Child.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Child.java
java -cp <memory> Parent
```

Hence in this case, output is: Parent.

Please note that source-file can contain multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

Also note that file name can be any name supported by underlying operating system, such as if you rename 'Child.java' to '123.java'.

java 123.java will be equivalent to:

```
javac -d <memory> 123.java
java -cp <memory> Parent
```

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

3.1.2 Answer: A

Reason :

class Error extends Throwable, so `obj instanceof Error;` and `obj

instanceof Throwable;` return true.
But Error class is not related to Exception and RuntimeException classes in multilevel inheritance and that is why Line n1 and Line n2 cause compilation error.

3.1.3 Answer: A

Reason :
Given question expects you to solve the compilation error and not care about runtime error. For array indexes, any int values can be used, hence all the 7 pairs are allowed in this case.

If question were expecting to compile and execute the program successfully, then any combination greater than the max indexes values would have worked. For example, in the given code, as max 1st dimension value = 6 and max 2nd dimension value = 6, so any int value > 6 can be used for x and any int value > 6 can be used for y.
Out of the given seven options, only two options (x = 7, y = 7) and (x = 8, y = 8) would have worked.

Also note that the statement at Line n1 after replacing the values of x and y compiles successfully. E.g.
`var arr = new int[1][1];`: Right side expression creates an int[][] object. arr infers to int[][] type.

3.1.4 Answer: E

Reason :
In Java language, boolean type can store only two values: true and false and these values are not compatible with int type.
Also + operator is not defined for boolean types. Hence, all the 3 statements inside main method causes compilation error.

3.1.5 Answer: C

Reason :
List is super type and ArrayList is sub type, hence List l = new ArrayList(); is valid syntax.
Animal is super type and Dog is sub type, hence Animal a = new Dog(); is valid syntax. Both depicts Polymorphism.

But in generics syntax, Parameterized types are not polymorphic, this means ArrayList<Animal> is not super type of ArrayList<Dog>.

Remember this point. So below syntaxes are not allowed:
ArrayList<Animal> list = new ArrayList<Dog>(); OR List<Animal> list = new ArrayList<Dog>();

3.1.6 Answer: B

Reason :

Variable 'arr' refers to a two-dimensional array. for-each loops are used to iterate the given array.
In 1st iteration of outer loop, str refers to one-dimensional String array {"%", "$$"}.
In 1st iteration of inner loop, s refers to "%" and "%" will be printed on to the console. Boolean expression of Line n1 evaluates to false so Line n2 is not executed.
In 2nd iteration of inner loop, s refers to "$$" and "$$" will be printed on to the console. Boolean expression of Line n1 evaluates to false so Line n2 is not executed.
Iteration of inner for-each loop is over and control executes Line n3. break; statement at Line n3 terminates the outer loop and program ends successfully.

So, output is:

```
%
$$
```

3.1.7 Answer: A,C,D

Reason :

ClassCastException, NumberFormatException and IllegalArgumentException are Runtime exceptions. There are no exception classes in java with the names: NullException and ArrayIndexException.

3.1.8 Answer: B

Reason :

`final boolean flag = false;` statement makes flag a compile time constant.
Compiler knows the value of flag, which is false at compile time and hence it gives "Unreachable Code" error for Line n2.

3.1.9 Answer: C

Reason :

Dot (.) operator has higher precedence than concatenation operator, hence toUpperCase() is invoked on str2 and not on the result of (str1 + "_" + str2).

3.1.10 Answer: D

Reason :
public static void main(String[] args) method is invoked by JVM.
Variable args is initialized and assigned with Program arguments. For example,
java Test: args refers to String [] of size 0.
java Test Hello: args refers to String [] of size 1 and 1st array element refers to "Hello"
java Test 1 2 3: args refers to String [] of size 3 and 1st array element refers to "1", 2nd array element refers to "2" and 3rd array element refers to "3".

Command used in this question: java Test Good, so args refers to String[] of size 1 and element at 0th index is "Good".
args[1] = "Day!"; is trying to access 2nd array element at index 1, which is not available and hence an exception is thrown at runtime.

3.1.11 Answer: C

Reason :
`System.out.println(new Child());` invokes the toString() method on Child's instance.

Parent class's method can be invoked by super keyword. super.toString() method returns "Inner " and "Inner ".concat("Peace!") returns "Inner Peace!".

3.1.12 Answer: D

Reason :
'super' refers to parent class object and 'this' refers to currently executing object.

3.1.13 Answer: E

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index

variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier 'var' is not a keyword, hence 'var' can still be used as variable name, method name or package name but it cannot be used as a class or interface name.
At Line n1, initialization expression of for loop is: `var var = 0;`. Java compiler automatically infers 'var' to be of int type as right side expression is int literal.

Variable 'var' is declared inside for loop, hence it is not accessible beyond loop's body. Line n2 causes compilation error.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

3.1.14 Answer: D

Reason :
"swimming" is different from "Swimming", so there is no matching case available. default block is executed, "RUNNING" is printed on to the console. No break statement inside default, hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends. So in this case, it prints TENNIS, SWIMMING, FOOTBALL one after another and break; statement takes control out of switch block. main method ends and program terminates successfully.

3.1.15 Answer: C

Reason :
removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.
Interface java.util.function.Predicate<T> declares below non-overriding abstract method:
boolean test(T t);

Lambda expression passed to removeIf(Predicate) method is: `a -> a % 10 == 0` and is a valid lambda expression for Predicate<Integer> interface.

Predicate's test method returns true for all the Integers divisible by 10, so all the list elements which are divisible by 10 (100, 50 and 10] are successfully removed by removeIf(Predicate) method.

`System.out.println(list);` prints remaining elements of the list and hence output is: [7, 17, 5]

3.1.16 Answer: B

Reason :
Use -d option with javac command. -d option is used to specify where to place generated class files (which is under the 'classes' directory).
As you are typing javac command from within 'codes' directory, hence path of 'classes' directory and Test.java file relative to 'codes' directory can be given.
So, correct command is: javac -d classes\ src\com\udayankhattry\ocp1\Test.java

Please note file name is 'Test.java' but not 'com.udayankhattry.ocp1.Test.java'.

3.1.17 Answer: A

Reason :
new Counter(); is invoked only once, hence only one Counter object is created in the memory. 'c1', 'c2', 'c3' and 'counter' are reference variables of Counter type and not Counter objects.

3.1.18 Answer: B

Reason :
By default, all the Java modules are dependent on java.base module. There is no need to have 'requires' directive in the module descriptor file (module-info.java) for java.base module.

3.1.19 Answer: C

Reason :
Given statement:
int var = --m * m++ + m-- - --m;
int var = --m * (m++) + (m--) - --m; // Postfix operator has higher precedence than other available operators
int var = (--m) * (m++) + (m--) - (--m); //Then comes prefix operators
int var = ((--m) * (m++)) + (m--) - (--m); //* comes next
int var = (((--m) * (m++)) + (m--)) - (--m); //+ is next

Right hand side is left with just one operator '-', it is a binary operator, hence let's solve the left hand side first.
int var = ((19 * (m++)) + (m--)) - (--m); //m = 19
int var = ((19 * 19) + (m--)) - (--m); //m = 20
int var = (361 + 20) - (--m); //m = 19
int var = 381 - (--m); //m = 19
int var = 381 - 18; //m = 18
int var = 363 // m = 18
So,
m = 18
var = 363

3.1.20 Answer: F

Reason :
You need to keep in mind an important point related to String Concatenation:
If only one operand expression is of type String, then string conversion is performed on the other operand to produce a string at run time.
If one of the operand is null, it is converted to the string "null".
If operand is not null, then the conversion is performed as if by an invocation of the toString method of the referenced object with no arguments; but if the result of invoking the toString method is null, then the string "null" is used instead .

Let's check the expression of Line n1:
text = text + new A(); --> As text is of String type, hence + operator behaves as concatenation operator.
As text is null, so "null" is used in the Expression.
new A() represents the object of A class, so toString() method of A class is invoked, but as toString() method of A class returns null, hence "null" is used in the given expression.
So, given expression is written as:
text = "null" + "null";
text = "nullnull";

Hence, Line n2 prints 8 on to the console.

3.1.21 Answer: C

Reason :

Expected output from the given Java command C:\>java -p jars -m com.udayankhattry.modularity is: MODULAR APPLICATION

Please note that in the above command class name to execute (com.udayankhattry.ocp1.Main) is not specified in the -m option, which means META-INF\MANIFEST.MF file must have below entry for the main class:
Main-Class: com.udayankhattry.ocp1.Main

To create this entry in jar file -e or --main-class option of jar command is used. Out of the given 4 options, only one command has -e option:

```
jar -cfe jars\test.jar com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
```

And hence it is the correct option.

Let's see how above jar command works:
-c or --create: Create the archive
-f or --file=FILE: The archive file name
-e or --main-class=CLASSNAME: The application entry point for stand-alone applications bundled into a modular, or executable, jar archive
-C DIR: Change to the specified directory and include the following file

Hence,

```
jar --create --file=jars\test.jar --main-class=com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
```

OR

```
jar --create --file jars\test.jar --main-class com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
```

OR

```
jar -c -f jars\test.jar -e com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
```

OR

```
jar -cfe jars\test.jar com.udayankhattry.ocp1.Main -C classes\com.udayankhattry.modularity .
```

are exactly same commands and works as below:
Create a new archive 'jars\test.jar' and set the entry point to 'com.udayankhattry.ocp1.Main' class.
-C classes\com.udayankhattry.modularity . instructs jar tool to change to

the 'classes\com.udayankhattry.modularity' directory and put all compiled files from this directory in the JAR file.

3.1.22 Answer: A

Reason :

1. Module description file name is: module-info.java

2. Module descriptor of the jar has below entry:

exports com.database.util to dbtester; => This is known as qualified export, in which package 'com.database.util' is made available to only named module 'dbtester'. Hence your module name must be: 'dbtester'.

3. Your module descriptor must have below requires directive:

requires dbconnection;

Please note after the requires directive module name should come and not the package name.

Based on above 3 points, correct option is:

```
//module-info.java
module dbtester {
    requires dbconnection;
}
```

3.1.23 Answer: A

Reason :

list --> [P, O, T]

sublist method is declared in List interface:

List<E> subList(int fromIndex, int toIndex)

fromIndex is inclusive and toIndex is exclusive

It returns a view of the portion of this list between the specified fromIndex and toIndex. The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list and vice-versa.

If returned list (or view) is structurally modified, then modification are reflected in this list as well but if this list is structurally modified, then the semantics of the list returned by this method become undefined.

If fromIndex == toIndex, then returned list is empty.

If fromIndex < 0 OR toIndex > size of the list OR fromIndex > toIndex, then IndexOutOfBoundsException is thrown.

list.subList(1, 2) --> [O] (fromIndex is inclusive and endIndex is exclusive, so start index is 1 and end index is also 1). subList --> [O]. At Line n2, `subList.set(0, "E");` => sublist --> [E]. This change is also reflected in the backed list, therefore after this statement, list --> [P, E, T]

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null "
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

`System.out.println(String.join("", list));` prints PET on to the console.

3.1.24 Answer: E

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name. Line n1 compiles successfully as static variable 'var' is of String type and refers to "FRIENDS".

At Line n2:
int var = (var = Test.var.length()); //Local variable 'var' is declared of int type and shadows static String variable 'var'
=> int var = (var = 7); //Test.var refers to "FRIENDS" and "FRIENDS".length() = 7
=> int var = 7; //7 is assigned to var
=> 7 is again assigned to var

Line n3 prints 7 on to the console.

3.1.25 Answer: D

Reason :
Line n1 declares a StringBuilder [] object of 2 elements and both the elements are initialized to default value null. So, sb --> {null, null}.

Line n2 initializes 1st array element to ["PLAY"]. So, sb --> {["PLAY"], null}.

Line n3 creates a boolean array object of 1 element and this element is initialized to default value of boolean, which is false. flag --> {false}.

At Line n4, boolean expression of if-block is 'flag[0]', which evaluates to false, control doesn't enter if block and Line n5 is not executed.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter .
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

As StringBuilder implements CharSequence, hence `String.join("-", sb)` is valid as 'sb' refers to StringBuilder[] object.
As per above explanation, `String.join("-", sb)` will return PLAY-null and Line n6 will print PLAY-null on to the console.

3.1.26 Answer: G

Reason :
remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false.

Initially list has: [Austin, Okinawa, Giza, Manila, Batam, Giza].
places.remove("Giza") removes the first occurrence of "Giza" and after the successful removal, list has: [Austin, Okinawa, Manila, Batam, Giza].
places.remove("Giza") returns true, control goes inside if block and executes places.remove("Austin");

places list contains "Austin", so after the removal list has: [Okinawa, Manila, Batam, Giza].

3.1.27 Answer: B,D,F

Reason :
Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

Please note:
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable

declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name. There is no compilation error in the code for using identifier 'var'.

Line n5 causes compilation error as obj is of Child type and 'var' is declared private in Child class. Variable 'var' of Child class cannot be accessed outside the Child class.

Let's check all the options one by one:
'Change Line n1 to private int var = 1000;' => It will not rectify the existing error of Line n5, in fact after this change, Line n3 will also cause compilation error.

'Delete the Line n2' => After deleting this line, obj.var at Line n5 will refer to variable 'var' of Parent class. Hence, output will be 1000 in this case.

'Change Line n3 to return var;' => This will have no effect to the output of the code, as getVar() method has not been invoked.

'Change Line n4 to Parent obj = new Child();' => After this modification, obj becomes Parent type, hence obj.var will refer to variable 'var' of Parent class. Hence, output will be 1000 in this case.

'Delete the method getVar() from the Child class' => This will have no effect to the output of the code, as getVar() method has not been invoked.

'Change Line n5 to System.out.println(obj.getVar());' => obj.getVar() will invoke the getVar() method of Child class and this method returns the variable value from Parent class (super.var). Hence, output will be 1000 in this case.

3.1.28 Answer: B

Reason :
super(); is added by the compiler as the first statement in both the

constructors:

```
Animal() {
    super();
    System.out.print("ANIMAL-");
}
```

and

```
public Dog() {
    super();
    System.out.print("DOG");
}
```

Class Animal extends from Object class and Object class has no-argument constructor, hence no issues with the constructor of Animal class.

Animal class's constructor has package scope, which means it is accessible to all the classes declared in package 'a'. But Dog class is declared in package 'b' and hence `super();` statement inside Dog class's constructor causes compilation error as no-argument constructor of Animal class is not visible.

There is no compilation error in Test.java file as Dog class's constructor is public and therefore `new Dog();` compiles successfully.

3.1.29 Answer: A

Reason :
Instance overloaded method split(...) has below declarations:
1. public String[] split(String regex, int limit) {...}: It returns the String[] after splitting this string around matches of the given regex.
The limit parameter controls the number of times the pattern is applied and therefore affects the length of the resulting array.
A. If the limit is positive then the pattern will be applied at most (limit - 1) times, the array's length will be no greater than limit, and the array's last entry will contain all input beyond the last matched delimiter. For example:
"BAZAAR".split("A", 1); returns ["BAZAAR"] as pattern is applied 1 - 1 = 0 time.
"BAZAAR".split("A", 2); returns ["B","ZAAR"] as pattern is applied 2 -

1 = 1 time.
"BAZAAR".split("A", 3); returns ["B","Z","AR"] as pattern is applied 3 - 1 = 2 times.
"BAZAAR".split("A", 4); returns ["B","Z","","R"] as pattern is applied 4 - 1 = 3 times.
"BAZAAR".split("A", 4); returns ["B","Z","","R"] as pattern needs to be applied 5 - 1 = 4 times but it is applied 3 times (which is max).
":".split(":", 2); returns ["",""] as pattern is applied (2 - 1) time and there is an empty space ("") before and after ":", so resulting array contains two empty strings.

B. If the limit is zero then the pattern will be applied as many times as possible, the array can have any length, and trailing empty strings will be discarded. For example,
"BAZAAR".split("A", 0); returns ["B","Z","","R"].
":".split(":", 0); returns blank array as both the empty strings are discarded.

C. If the limit is negative then the pattern will be applied as many times as possible and the array can have any length. For example,
"BAZAAR".split("A", -10); returns ["B","Z","","R"] as pattern is applied max times, which is 3.
":".split(":", -2); returns ["",""] as pattern is applied max times, which is 3.

2. public String[] split(String regex) {...}: It returns the String[] after splitting this string around matches of the given regular expression.
This method works as if by invoking the two-argument split method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.
"BAZAAR".split("A"); returns ["B","Z","","R"] as this statement is equivalent to "BAZAAR".split("A", 0); .
":".split(":"); returns blank array as this statement is equivalent to ":".split(":", 0);.

Let's solve the given expression based on above points:
txt.split("an").length
="an".split("an").length
=[].length //"an".split("an") returns blank array as both the empty strings are discarded.

=0

Hence, 0 is printed on to the console.

3.1.30 Answer: A

Reason :
As expression contains + operator only, which is left to right associative.
Let us group the expression.
1 + 2 + 3 + 4 + "Running"
= (1 + 2) + 3 + 4 + "Running"
= ((1 + 2) + 3) + 4 + "Running"
= (((1 + 2) + 3) + 4) + "Running"
Let us solve it now:
= ((3 + 3) + 4) + "Running"
= (6 + 4) + "Running"
= 10 + "Running"
[+ operator with String behaves as concatenation operator.]
= "10Running"

3.1.31 Answer: E

Reason :
Line n3 throws an instance of RuntimeException. As catch(RuntimeException e) is available, hence control starts executing catch-block inside check() method.
1 is printed on to the console.
At Line n4, instance of super-class (RuntimeException) is type-casted to sub-class (ArithmeticException), hence Line n4 throws an instance of ClassCastException.
ClassCastException is a sub-class of RuntimeException, so catch-block of main method is executed and Line n1 prints the fully qualified name of ClassCastException. java.lang.ClassCastException is printed on to the console.

3.1.32 Answer: B

Reason :
String and StringBuilder classes override toString() method, which prints the text stored in these classes.
SpecialString class doesn't override toString() method, therefore when instance of SpecialString is printed on to the console, toString() method

defined in Object class is invoked and that is why you get: <fully qualified name of SpecialString class>@<hexadecimal representation of hashcode>.

3.1.33 Answer: D

Reason :
Message class doesn't specify any constructor, hence Java compiler adds below default constructor:
Message() {super();}

Line n1 creates an instance of Message class and initializes instance variable 'msg' to "LET IT GO!". Variable 'obj' refers to this instance.
Line n2 prints LET IT GO! on to the console.
Line n3 invokes change(Message) method, as it is a static method defined in TestMessage class, hence `change(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 'm' copies the content of variable 'obj' (which stores the address to Message instance created at Line n1). 'm' also refers to the same instance referred by 'obj'.

Line n6, assigns "NEVER LOOK BACK!" to the 'msg' variable of the instance referred by 'm'. As 'obj' and 'm' refer to the same instance, hence obj.msg also refers to "NEVER LOOK BACK!". change(Message) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg refers to "NEVER LOOK BACK!", so this statement prints NEVER LOOK BACK! on to the console.

Hence in the output, you get:
LET IT GO!
NEVER LOOK BACK!

3.1.34 Answer: B

Reason :
As per Java 8, default and static methods were added in the interface.
Interface I1 defines static method print(String), there is no compilation error in interface I1.
Also the scope of print(String) method of I1 is limited to interface I1 and

it can be invoked by using Interface name only, I1.print("").

class C1 implements I1 and it also defines print(String) instance method. Even though class C1 implements I1, it doesn't have static print(String) method in its scope, therefore class C1 compiles successfully.

Super type reference variable can refer to an instance of Sub type, therefore the statement `I1 obj = new C1();` compiles successfully.
obj is of I1 type, hence `obj.print("Java");` tries to tag the static method of I1 but static print(String) method of I1 can only be invoked by using I1.print("Java");.
Therefore, `obj.print("Java");` causes compilation error.

3.1.35 Answer: B,C,D

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.
Hence at Line n1, java compiler automatically infers x to be of int type as right side expression is int literal.

Line n2 compiles successfully. As x is a final variable, so x + 10 is a constant expression. If you are working with constant expression and the resultant value of the constant expression is within the range, then resultant value is implicitly casted. In this case, resultant value 20 is implicitly casted.

Starting with JDK 11, it is possible to launch single-file source-code Programs .
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
      (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java Test.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Test.java
java -cp <memory> Test
```

Please note that source-file can contain multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

`java Test.java` prints 20 on to the console.
`java --source 11 Test.java` instructs the java command to process the Test.java file as Java 11 source code. As var reserved type works JDK 10 onwards, hence this command also prints 20 on to the console.
`java --source 10 Test.java` instructs the java command to process the Test.java file as Java 10 source code. As var reserved type works JDK 10 onwards, hence this command also prints 20 on to the console.
`java --source 9 Test.java` instructs the java command to process the Test.java file as Java 9 source code. As var reserved type works JDK 10 onwards, hence this command causes error.
`java --source 8 Test.java` instructs the java command to process the Test.java file as Java 8 source code. As var reserved type works JDK 10 onwards, hence this command causes error.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286
For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

3.1.36 Answer: D

Reason :
Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:

Specify where to find input source files for multiple modules. In this case, module source path is 'codes' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'classes' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

Let's check all the options one by one:
--module-path ✗ --module-source-path is missing. --module-path is same as -p and it expects path of compiled classes or jar files containing compiled classes.

--module-source-path ✓ Correct.

--path ✗ --path is not a valid option of javac command.

--source-code-path ✗ --source-code-path is not a valid option of javac command .

--source ✗ --source is not a valid option of javac command.

-p ✗ --module-source-path is missing. -p is same as --module-path and it expects path of compiled classes or jar files containing compiled classes.

3.1.37 Answer: D

Reason :
A class in Java extends from another class and implements interface(s).

Hence correct syntax is:
class Mosquito extends Insect implements Flyable{}

3.1.38 Answer: B

Reason :
Variable 'book' in main(String[]) method of TestBook class cannot be declared private as it is a local variable. Hence, there is a compilation error in TestBook class.
Only final modifier can be used with local variables.

3.1.39 Answer: B

Reason :
To declare Test class in com.university.sem1 package, use following declaration:
package com.university.sem1;
No wild-card (*) allowed in package declaration. Don't include class name in package declaration. NOTE: all small case letters in package keyword.

3.1.40 Answer: A

Reason :
Employee class has overloaded constructors: Employee() is a no-argument constructor and Employee(String, int) is a parameterized constructor.
A constructor can call another constructor by using this(...) and not the constructor name. Hence Line n1 causes compilation error.

As both Employee and TestEmployee classes are defined in the same package, hence emp.name and emp.age are valid syntaxes. Line n2 compiles successfully.

3.1.41 Answer: C

Reason :
Even though it seems like method m() will not compile successfully, but starting with JDK 7, it is allowed to use super class reference variable in throw statement referring to sub class Exception object.

In this case, method m() throws SQLException and compiler knows that variable e (Exception type) refers to an instance of SQLException only and hence allows it.

Program executes successfully and prints CAUGHT SUCCESSFULLY on to the console.

3.1.42 Answer: A

Reason :

At Line n3, String reference variable 'str' refers to String object "Think".
When change(String) method is invoked by Line n4, both variables 's' and 'str' refer to the same String object "Think".
As String is immutable, so Line 9 doesn't modify the passed object, instead it creates a new String object "Think_Twice".
But this newly created object is not referred and hence is a candidate for GC.
When control goes back to calling method main(String[]), 'str' still refers to "Think".
Line n5 prints "Think" on to the console.

3.1.43 Answer: G

Reason :

sb --> {"TOMATO"}
sb.reverse() --> {"OTAMOT"}. reverse() method returns a StringBuilder object.
replace method of StringBuilder class accepts 3 arguments: `replace(int start, int end, String str)`. At Line n1, replace("O", "A") method accepts 2 arguments and hence it causes compilation error.

3.1.44 Answer: A,B,C,F

Reason :

Target type of lambda expression must be a Functional interface.
Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].

Let's check which of the given options are Functional interfaces:

```
interface MyInterface {
    String toStr();
} ✓
```

MyInterface has only one non-overriding abstract method [toStr()] and hence a Functional interface.

Given lambda expression `() -> ""` means method accepts no argument and returns a String object. It is a correct implementation of toStr() method.

```
interface MyInterface {
    String toStr();
    String toString();
} ✓
```

MyInterface has only one non-overriding abstract method [toStr()] and toString() method is overriding abstract method, therefore it is also a valid Functional interface.
`() -> "";` is the correct implementation of toStr() method.

```
interface MyInterface {
    Object toStr();
    String toString();
    static void print(String str) {
        System.out.println(str);
    }
} ✓
```

MyInterface has only one non-overriding abstract method [toStr()] and toString() method is overriding abstract method & print(String) is the static method, therefore it is also a valid Functional interface.
`() -> "";` is the correct implementation of toStr() method.

```
interface MyInterface {
    Object toStr();
    boolean equals(MyInterface);
} ✗
```

My interface has two non-overriding abstract methods. Please note that `boolean equals(Object);` is overriding abstract method but `boolean equals(MyInterface);` is non- overriding abstract method.

```
interface MyInterface {
    String toString();
} ✗
```

`String toString();` is overriding abstract method and hence in this case, MyInterface doesn't contain non-overriding abstract method. Therefore it is not a Functional interface.

```
interface MyInterface {
    CharSequence test();
} ✓
```

MyInterface has only one non-overriding abstract method [test()] and hence a Functional interface.
As String class implements CharSequence, therefore value returned by Lambda expression [""] (String type) can easily be assigned to CharSequence reference.
`() -> "";` is the correct implementation of `CharSequence test();`

```
interface MyInterface {
    StringBuilder m();
} ✗
```

MyInterface has only one non-overriding abstract method [m()] and hence a Functional interface.
Value returned by Lambda expression [""] (String type) can not be assigned to StringBuilder reference.
`() -> "";` is NOT the correct implementation of `StringBuilder m();`

3.1.45 Answer: A,B,D,E,G,H

Reason :
print(String) method accepts parameter of String type, so left side of lambda expression should specify one parameter, then arrow operator and right side of lambda expression should specify the body.

Let's check all the options one by one:
Printable obj = (String msg) -> {System.out.println(msg);}; => ✓ Valid Lambda expression syntax .
Printable obj = (String msg) -> {System.out.println(msg); return;}; => ✓ As print(String) method of Printable interface returns void, hence `return;` statement works even though it is not mandatory.
Printable obj = String msg -> {System.out.println(msg);}; => ✗ Round brackets are expected around `String msg`.
Printable obj = (msg) -> {System.out.println(msg);}; => ✓ Type of variable can be removed from left side. Java compiler handles it using type inference.
Printable obj = (msg) -> System.out.println(msg); => ✓ If there is only

one statement in the right-hand side then semicolon inside the body, curly brackets and return keyword(if available) can be removed.
Printable obj = msg -> {System.out.println(msg)}; => ✗ Expression on right-hand side uses curly brackets, therefore inner semicolon is required.
Printable obj = msg -> System.out.println(msg); => ✓ If there is only one parameter in left part, then parentheses or round brackets '()' can be removed. If there is only one statement in the right-hand side then semicolon inside the body, curly brackets and return keyword(if available) can be removed.
Printable obj = x -> System.out.println(x); => ✓ Any valid java identifier can be used in lambda expression.
Printable obj = y - > System.out.println(y); => ✗ Compilation error as there should not be any space between - and > of arrow operator.

3.1.46 Answer: B

Reason :
By default a module doesn't export anything.
Package name without any wild-card (*) is expected after exports directive. And all the public type members (classes, interfaces, enums) of the exported package become available for public use.
But yes the reading module code must have requires directive in its module descriptor file.

Hence, out of all the option below option is correct:

```
module com.udayankhattry.utility {
    exports com.udayankhattry.utility;
}
```

Both the classes FileUtility and StringUtility are defined in exported package (com.udayankhattry.utility) and hence become available for public use .

There is no 'not exports' directive and with exports directives class names are not allowed.

Please note that module names live in a global namespace, separate from other namespaces in Java. Hence, module name can use the name of the package, class or interface.

3.1.47 Answer: C

Reason :
127 + 21 = 148 (int type: 32-bits) = 00000000 00000000 00000000 10010100
Above binary number is positive, as left most bit is 0.
Same binary number after type-casting to byte (8-bits): 10010100, negative number as left most bit is 1.
There is only one negative number in the option and hence -108 is the correct answer.
Binary 10010100 = -108.

3.1.48 Answer: B,C

Reason :
Arrays.asList(...) method returns a fixed-size list backed by the specified array and as list is backed by the specified array therefore, you cannot add or remove elements from this list. Using add/remove methods cause an exception at runtime. But you can invoke the set(int index, E element) method on the returned list.
This behavior is bit different from the List.of(...) method, which returns unmodifiable list, hence calling add/remove/set methods on the unmodifiable list throws an exception at runtime.

Iterable<T> interface has forEach(Consumer) method. List<E> extends Collection<E> & Collection<E> extends Iterable<E>, therefore forEach(Consumer) can easily be invoked on reference variable of List<E> type.
As Consumer is a Functional Interface, hence a lambda expression can be passed as argument to forEach() method.
forEach(Consumer) method performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

If you replace /*INSERT*/ with List.of(arr), then for the statement `var list = List.of(arr);`, list refers to a List of String type.

Let's check all the options one by one:
List.of(arr) ✗ Line n1 will throw an exception at runtime.

Arrays.asList(arr) ✓ set(...) method can be invoked on the list object returned by Arrays.asList(arr) method. Line n1 will change the last list element from "PLUTO" to "JUPITER". Line n2 will display the expected output.

new ArrayList<>(List.of(arr)) ✓ It constructs the ArrayList instance containing the elements of the specified collection (list object returned by List.of(arr)). This ArrayList object is not backed by the passed collection and hence can be modified without any issues. Line n1 will change the last list element from "PLUTO" to "JUPITER". Line n2 will display the expected output.

ArrayList.of(arr) ✗ static overloaded methods of(...) were added to List interface in Java 9 and can only be invoked by using List.of(...). This statement causes compilation error.

new ArrayList<>(ArrayList.of(arr)) ✗ static overloaded methods of(...) were added to List interface in Java 9 and can only be invoked by using List.of(...). This statement causes compilation error.

3.1.49 Answer: E

Reason :
Module graph is all about modules and their dependencies.
In a modular graph, modules are represented by nodes and dependencies between the modules are represented by arrows/edges. For example,

java.se ————> java.compiler ————> java.base

In the above module graph portion, the nodes 'java.se', 'java.compiler' and 'java.base' are modules and ————> represents dependencies between the modules.
'java.se' module depends upon 'java.compiler' module and 'java.compiler' module depends upon 'java.base' module.

3.1.50 Answer: C

Reason :

As per Java 8, default and static methods were added in the interface. Interface M defines static method log(), there is no compilation error in interface M.
Also the scope of static log() method of M is limited to interface M and it can be invoked by using Interface name only, M.log().

Abstract class A also defines the static log() method. Abstract class can have 0 or more abstract methods. Hence, no compilation error in class A as well.

Super type reference variable can refer to an instance of Sub type, therefore the statement `M obj1 = new MyClass();` compiles successfully. obj1 is of M type, hence `obj1.log();` tries to tag the static method of M but static log() method of M can only be invoked by using M.log();. Therefore, Line n1 causes compilation error.

Scope of static log() method of A is not limited to class A only but MyClass also gets A.log() method in its scope.
There are different ways in which static method of an abstract class can be accessed:
1. By using the name of the abstract class: M.log(); //Preferred way
2. By using the reference variable of abstract class: A o1 = null; o1.log();
3. By using the name of the subclass: MyClass.log();
4. By using the reference variable of the subclass: MyClass o2 = null; o2.log();
Hence, Line n2 and Line n3 compile successfully.

3.1.51 Answer: D

Reason :
Variable 'i' declared inside interface I1 is implicitly public, static and final and similarly variable i declared inside interface I2 is implicitly public, static and final as well.
In Java a class can extend from only one class but an interface can extend from multiple interfaces. static variables are not inherited and hence there is no issue with Line n1.

I1.i points to variable 'i' of interface I1 .
I2.i points to variable 'i' of interface I2.
I3.i is an ambiguous call as compiler is not sure whether to point to I1.i or

I2.i and therefore, Line n4 causes compilation error.

3.1.52 Answer: C

Reason :

class Dog {}: can be sub-classed within the same package.
abstract class Cat {}: can be sub-classed within the same package.
final class Electronics {}: a class with final modifier cannot be sub-classed.
private class Car {}: a top level class cannot be declared with private modifier.

3.1.53 Answer: A

Reason :

Given expression:
val += 10 - -val-- - --val
val = val + 10 - -val-- - --val
val = val + 10 - -(val--) - --val //Postfix operator has higher precedence than other available operators
val = val + 10 - (-(val--)) - (--val) //Unary minus and prefix operator has same preference
val = (val + 10) - (-(val--)) - (--val) // + and - have same preference and both are left associative, hence grouping + first.
val = ((val + 10) - (-(val--))) - (--val) //Grouping - next
Expression on right side of assignment operator has 2 parts: Left: ((val + 10) - (-(val--))) and Right: (--val). Expression on Left side needs to be evaluated first.
val = ((9 + 10) - (-(val--))) - (--val) //val=9
val = (19 - (-(val--))) - (--val) //val=9
val = (19 - (-9)) - (--val) //val=8
val = 28 - (--val) //val=8
val = 28 - 7 //val=7
val = 21
21 is assigned to val and 21 is printed on to the console as well.

3.1.54 Answer: D

Reason :

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index

variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

At Line n1, variable 'x' infers to int type.

Even though compiler is aware that Line n2 will never execute, but it doesn't tag it as unreachable code. Reason for this odd behavior is explained in the Java Language specification:
https://docs.oracle.com/javase/specs/jls/se11/html/jls-14.html#jls-14.21
Following statement results in a compile-time error:
while (false) { x=3; }
because the statement x=3; is not reachable; but the superficially similar case:
if (false) { x=3; }
does not result in a compile-time error. An optimizing compiler may realize that the statement x=3; will never be executed and may choose to omit the code for that statement from the generated class file, but the statement x=3; is not regarded as "unreachable" in the technical sense specified here.

The rationale for this differing treatment is to allow programmers to define "flag" variables such as:

static final boolean DEBUG = false;
and then write code such as:

if (DEBUG) { x=3; }
The idea is that it should be possible to change the value of DEBUG from false to true or from true to false and then compile the code correctly with no other changes to the program text .

Line n2 is not executed but Line n3 executes successfully and prints HELLO on to the console.

3.1.55 Answer: A

Reason :
i1 is a static variable and i2 is an instance variable. Preferred way to access static variable i1 inside add() method is by using 'i1' or 'Test.i1'. Even though 'this.i1' is not the recommended way but it works.
And instance variable i2 can be accessed inside add() method by using 'i2' or 'this.i2'. Hence, Line n1 compiles successfully.

As add() is an instance method of Test class, so an instance of Test class is needed to invoke the add() method. `new Test().add()` correctly invokes the add() method of Test class and returns 30. Line n2 prints 30 on to the console.

3.1.56 Answer: D

Reason :
NOTE: Question is asking for "incorrect" implementation and not "correct" implementation.

java.io.FileNotFoundException extends java.io.IOException
and
java.io.IOException extends java.lang.Exception

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:
1. May not declare to throw any checked exception.
2. May declare to throw the same checked exception thrown by super class / interface method.
3. May declare to throw the sub class of the exception thrown by super class / interface method.
4. Cannot declare to throw the super class of the exception thrown by super class / interface method.
5. Cannot declare to throw unrelated checked exception.
6. May declare to throw any RuntimeException or Error .

Based on above rules, class C4 incorrectly implements method m1() of interface I1 as it declares to throw Exception, which is super class of java.io.IOException.

3.1.57 Answer: C

Reason :

Line n1 creates an int array object of 3 elements: 1, 2, 3 and arr1 refers to it.
Line n2 creates and int array object of 2 elements: 65, 66 [char type is compatible with int type] and arr2 refers to it.
At Line n3, variable arr1 copies the content of arr2, as arr2 refers to an int array object of 2 elements: 65, 66 hence arr1 also starts referring to the same array object.
At Line n4, `arr1[i] + " "`, + operator behaves as concatenation operator as left operand (arr1[i]) is of int type and right operand (" ") is of String type.
for loop prints all the elements of an array object referred by arr1, which is 65 66

3.1.58 Answer: B

Reason :
process(List, Predicate) method prints all the records passing the Predicate's test and test is to process the records having Furniture's weight less than 45. There are 3 records (Chair, Table & Sofa) with weight < 45 and these are printed in the insertion order.

NOTE: toString() method just returns the name.

3.1.59 Answer: A,B,F

Reason :
save() method throws IOException (which is a Checked Exception) and log() method throws SQLException (which is also a Checked Exception).

Let's check all the options one by one (I am just using the catch-block as try-block of all the options are same):
catch(IOException | SQLException ex) {}: ✓ As IOException and SQLException are not related to each other in multi-level inheritance, hence this multi-catch syntax is valid.

catch(SQLException | IOException ex) {}: ✓ Same as above, order of exceptions in multi-catch syntax doesn't matter .

catch(IOException | Exception ex) {}: ✗ Causes compilation error as IOException extends Exception.

catch(SQLException | Exception ex) {}: ✗ Causes compilation error as

SQLException extends Exception.

catch(Exception | RuntimeException ex) {}: ✗ Causes compilation error as RuntimeException extends Exception.

catch(Exception ex) {}: ✓ As Exception is the super class of both IOException and SQLException, hence it can handle both the exceptions.

3.1.60 Answer: D

Reason :
Variable x is of X type (superclass) and refers to an instance of Z type (subclass).
At Line n1, compiler checks whether greet() method is available in class X or not. As greet() method is available in class X, hence no compilation error for Line n1.
At Line n2, x is casted to Y and compiler checks whether greet() method is available in class Y or not. As greet() method is available in class Y, hence no compilation error for Line n2.
At Line n3, x is casted to Z and compiler checks whether greet() method is available in class Z or not. As greet() method is available in class Z, hence no compilation error for Line n3.

There is no compilation error in the given code it compiles successfully.

Variable x refers to an instance of Z class and at Line n1, n2 and n3 same instance is being used. Which overriding method to invoke is decided at runtime based on the instance.
At runtime, all three statements, at Line n1, Line n2 and Line n3 would invoke the greet() method of Z class, which would print Good Night! three times on to the console.

3.1.61 Answer: C

Reason :
As we are required to replace /*INSERT*/ to resolve compilation error, which means other codes compile fine, no need to check for any other logic.
We need to just check the readability of the modules.

Let's start with the root module 'com.udayankhattry.test'
It uses both 'com.udayankhattry.game.Game' and

'com.udayankhattry.gifts.Gift' classes. Though it has `requires com.udayankhattry.game;` directive but `requires com.udayankhattry.gifts;` is missing. And we are not suppose to make changes in C:\src\com.udayankhattry.test\module-info.java file.
Only file that can be changed is: C:\src\com.udayankhattry.game\module-info.java.

If /*INSERT*/ is replaced with 'requires', then module 'com.udayankhattry.test' will still not be able to read the module 'com.udayankhattry.gifts' as readability is not transitive.
For example, if module C has `requires B;` (C reads B) and module B has `requires A;` (B reads A), then C doesn't read A.
To have the transitive readability, 'requires transitive' directive is used. If module C has `requires B;` (C reads B) and module B has `requires transitive A;` (B reads A [transitive dependency]), then C will also read A.

Hence if /*INSERT*/ is replaced with 'requires transitive', then module 'com.udayankhattry.test' will also have readability access to module 'com.udayankhattry.gifts'.
This is also known as implied readability.

You will find several examples of this in modular JDK. For example, module java.se:

```
C:\>java --describe-module java.se
java.se@11.0.3
requires java.xml.crypto transitive
requires java.datatransfer transitive
requires java.naming transitive
requires java.management transitive
requires java.instrument transitive
requires java.sql transitive
requires java.xml transitive
requires java.security.sasl transitiv e
requires java.base mandated
requires java.prefs transitive
requires java.security.jgss transitive
requires java.transaction.xa transitive
requires java.desktop transitive
requires java.sql.rowset transitive
requires java.management.rmi transitive
requires java.rmi transitive
requires java.net.http transitive
requires java.compiler transitive
requires java.logging transitive
requires java.scripting transitive
```

So any module reading java.se will also read java.xml.crypto, java.datatransfer etc.

3.1.62 Answer: C,D

Reason :
List is an interface so its instance can't be created using new keyword. List<String> and List<> will cause compilation failure.
ArrayList implements List interface, so it can be it can be used to replace /*INSERT*/. `List<String> list = new ArrayList<String>();` compiles successfully.

Starting with JDK 7, Java allows to not specify type while initializing the ArrayList. Type is inferred from the left side of the statement.

So `List<String> list = new ArrayList<>();` is a valid syntax starting with JDK 7.

3.1.63 Answer: F

Reason :
`private void emp() {}` is a valid method declaration.
Class name and method name can be same and that is why given method can be declared in any of the given classes: 'emp', 'Emp', 'employee', 'Employee', 'Student' and '_emp_'.
'_emp_' is also a valid Java identifier.

3.1.64 Answer: B

Reason :
Special main method (called by JVM on execution) should be static and should have public access modifier. It also takes argument of String [] type (Varargs syntax String... can also be used).
String [] or String... argument can use any identifier name, even though in

most of the cases you will see "args" is used.
final modifier can be used with this special main method.

Hence, from the given five definitions of main method, below two definitions will print expected output on to the console.

```
public static final void main(String... a) {
    System.out.println("Java Rocks!");
}
```

and

```
public static void main(String [] args) {
    System.out.println("Java Rocks!");
}
```

3.1.65 Answer: B

Reason :

If you want to remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove() method.

In this case ListIterator.remove() method is used. startsWith("A") returns true for "Arrow" and "Anchor" so these elements are removed from the list. In the output, [Watch, Drum] is displayed.

3.1.66 Answer: B

Reason :

`int indexOf(String str)` method of String class returns the index within this string of the first occurrence of the specified substring. e.g. "Java".indexOf("a") returns 1.

`char charAt(int index)` method of String class returns the char value at the specified index. e.g. "Java".charAt(2) returns 'v' .

Let's check the given expression:

```
str.charAt(str.indexOf("A") + 1)
= "ALASKA".charAt("ALASKA".indexOf("A") + 1)
= "ALASKA".charAt(0 + 1) //"ALASKA".indexOf("A") returns 0.
= "ALASKA".charAt(1)
= 'L'
```

Hence, L is printed on to the console.

3.1.67 Answer: B,C,D

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

Compound declarations are allowed in Java for primitive type and reference type but not for var type. Hence, var cannot be used to replace /*INSERT*/.

Range of byte data type is from -128 to 127, hence if byte is used to replace /*INSERT*/, then y = 200 would cause compilation error as 200 is out of range value for byte type. Hence, byte cannot be used to replace /*INSERT*/.

short, int, long, float & double can replace /*INSERT*/ without causing any error. x + y will evaluate to 207 for short, int and long types whereas, x + y will evaluate to 207.0 for float and double types.
String class has overloaded valueOf methods for int, char, long, float, double, boolean, char[] and Object types. valueOf method returns the corresponding String object and length() method returns number of characters in the String object.
So, `String.valueOf(x + y).length()` in case of short, int and long returns 3, on the other hand, in case of float and double it would return 5.

Hence, only 3 options (short, int and long) print expected output on to the console .

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

3.1.68 Answer: A

Reason :

list1 --> [A, D],
list2 --> [B, C],
list1.addAll(1, list2); is almost equal to list1.addAll(1, [B, C]); => Inserts B at index 1, C takes index 2 and D is moved to index 3. list1 --> [A, B, C, D]

3.1.69 Answer: B

Reason :
Instance variable color is shadowed by the parameter variable color of parameterized constructor. So, color = color will have no effect, because short-hand notation within constructor body will always refer to LOCAL variable. To refer to instance variable, this reference is needed. Hence `this.color = color;` is correct.

`color = GREEN;` and `this.color = GREEN;` cause compilation error as GREEN is not within double quotes("").

NOTE: `color = "GREEN";` will only assign "GREEN" to local variable and not instance variable but `this.color = "GREEN";` will assign "GREEN" to instance variable.

3.1.70 Answer: D

Reason :
isEmpty() method of String class returns true if and only if length() is 0.
isBlank() method of String class (available since Java 11) returns true if the string is empty or contains only white space codepoints, otherwise false.

strip() method of String class (available since Java 11) returns a string whose value is this string, with all leading and trailing white space removed.

To find out what is white space character in Java, check Character.isWhitespace(int) method, you will find below:
A character is a Java whitespace character if and only if it satisfies one of the following criteria:
It is a Unicode space character (SPACE_SEPARATOR, LINE_SEPARATOR, or PARAGRAPH_SEPARATOR) but is not also a non-breaking space ('\u00A0', '\u2007', '\u202F').
It is '\t', U+0009 HORIZONTAL TABULATION.

It is '\n', U+000A LINE FEED.
It is '\u000B', U+000B VERTICAL TABULATION.
It is '\f', U+000C FORM FEED.
It is '\r', U+000D CARRIAGE RETURN.
It is '\u001C', U+001C FILE SEPARATOR.
It is '\u001D', U+001D GROUP SEPARATOR.
It is '\u001E', U+001E RECORD SEPARATOR.
It is '\u001F', U+001F UNIT SEPARATOR.

White space character in java includes space character along with above characters.

Let's check the code:
'str' refers to " " and its length is 1.
`str.isEmpty()` returns false, as length() is not 0.
`str.isBlank()` returns true, as it contains only white space character.

So, Line n1 prints false : true on to the console.

At Line n2, `str.strip();` returns an empty string "". As String is immutable, hence a new String object is created and 'str' still refers to " ".

As, there is no change in 'str', Line n3 also prints false : true on to the console.

3.1.71 Answer: E

Reason :
Class A, B and C are declared public and are inside same package 'com.udayankhattry.ocp1'. Method print() of class A has correctly been overridden by B and C.
print() method is public so no issues in accessing it anywhere.

Let's check the code inside main method.
A obj1 = new C(); => obj1 refers to an instance of C class, it is polymorphism.
A obj2 = new B(); => obj2 refers to an instance of B class, it is polymorphism.
C obj3 = (C)obj1; => obj1 actually refers to an instance of C class, so at runtime obj3 (C type) will refer to an instance of C class. As obj1 is of A type so explicit typecasting is necessary.

C obj4 = (C)obj2; => obj2 actually refers to an instance of B class, so at runtime obj4 (C type) will refer to an instance of B class. B and C are siblings and can't refer to each other, so this statement will throw ClassCastException at runtime.

3.1.72 Answer: A

Reason :

Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].

Equality is not a Functional Interface as it contains one overriding abstract method but non-overriding abstract method is not available.

`Equality eq = x -> true;` causes compilation error as target type must be a Functional interface.

3.1.73 Answer: B

Reason :

Inside for-each loop, System.out.println(arr[i]); is used instead of System.out.println(i);

When loop executes 1st time, i stores the first array element, which is 2 but System.out.println statement prints arr[2] which is 0. Loop executes in this manner and prints 0 1 2 on to the console.

3.1.74 Answer: B

Reason :

You have to print element at index 0, 2 and 4, which means index must start with 0 and step expression should increment the index by 2.

Hence, int n = 0; n < arr.length; n += 2 is the correct option.

3.1.75 Answer: B

Reason :

Local variable Type inference was added in JDK 10.

Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,

var x = "Java"; //x infers to String

var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

Given code compiles successfully.
At Line n1, variable 'res' infers to String and at line n2, variable 's' infers to String as well.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C "
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Let's check the iterations:
1st iteration: s refers to "1". `String.join(".", s)` returns "1" and res = "" + "1" = "1".
2nd iteration: s refers to "2". `String.join(".", s)` returns "2" and res = "1" + "2" = "12".
3rd iteration: s refers to "3". `String.join(".", s)` returns "3" and res = "12" + "3" = "123".
Loop finishes its execution and Line n4 prints 123 on to the console.

3.1.76 Answer: C

Reason :
Let's check all the options one by one:
No changes are necessary, code as is compiles successfully ✗ All modules implicitly requires java.base(which exports java.io package but not java.sql package).
To get a list of all the system modules, use below command:
java --list-modules
And to check details of specific module (e.g. java.base), use below command:
java --describe-module java.base
(you will notice that it exports java.io package)

Replace /*INSERT*/ with requires java.io; ✗ There is no module with the name java.io, package java.io is exported by java.base, which is implicitly required by all the modules.

Replace /*INSERT*/ with requires java.sql; ✓ Module java.sql exports java.sql package [java --describe-module java.sql], which is needed for import java.sql.SQLException; to work.

Replace /*INSERT*/ with requires java.*; ✗ Specific module name needs to be provided, wild-card character (*) is not allowed.

Replace /*INSERT*/ with requires java.base; java.sql; ✗ requires directive expects only one module. For multiple modules, use multiple requires directive.

3.1.77 Answer: B

Reason :
There is no element at index 0 so call to add element at index 1, "trafficLight.add(1, "RED");" throws an instance of java.lang.IndexOutOfBoundsException.

`trafficLight.remove(Integer.valueOf(2));` matches with trafficLight.remove(Object) and hence no compilation error.

3.1.78 Answer: A

Reason :
If a method declares to throw Exception or its sub-type other than RuntimeException types, then calling method should follow handle or declare rule.

In this case, as method check() declares to throw Exception, so main method should either declare the same exception or its super type in its throws clause OR check(); should be surrounded by try-catch block.

Line n1 in this case causes compilation error.

3.1.79 Answer: D

Reason :
`int [] arr1 = new int[8];` Creates one-dimensional array object to store 8 elements and arr1 refers to it. This statement compiles without any error.

`int [][] arr2 = new int[8][8];` Creates two-dimensional array object to store 8 * 8 = 64 elements. This statement also compiles fine.

`int [] arr3 [] = new int[8][];` Creates two-dimensional array object, whose 1st dimension is 8 but 2nd dimension is not yet defined. On the left side array symbols can be used before the reference variable or after the reference variable or can be mixed, hence `int [][] arr3`, `int [] arr3 []` and `int arr3[][]` all are valid. This statement compiles successfully.

`int arr4[][] = new int[][8];`: 1st array dimension must be specified at the time of declaration. new int[][8]; causes compilation error as 1st dimension is not specified.

3.1.80 Answer: A,B,C,D,G

Reason :

Both Traveller and BeachTraveller are abstract classes and BeachTraveller extends Traveller. It is possible to have abstract class without any abstract method. Code as is compiles successfully as BeachTraveller inherits travel(String) method of Traveller class.
But as per the question, /*INSERT*/ must be replaced such that there is no compilation error.

Let's check all the options one by one:
abstract void travel(); ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `abstract void travel()`.
abstract void travel(String beach); ✓ As BeachTraveller is abstract, hence travel(String) method can be declared abstract.
public abstract void travel(); ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `abstract void travel()`.
public void travel() throws RuntimeException {}: ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `public void travel() throws RuntimeException {}`.
public void travel(String beach) throws Exception {}: ✗ As overridden method doesn't declare to throw any checked Exception hence overriding method is not allowed to declare to throw Exception.
void travel(String beach) throws java.io.IOException {} ✗ As overridden method doesn't declare to throw any checked Exception hence overriding method is not allowed to declare to throw java.io.IOException.
public void travel(Object obj) {} ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `public void travel(Object){}`.

4 Practice Test-4

4.1 80 Questions covering all topics.

4.1.1 On Windows platform, you have below modular jar file for 'com.udayankhattry.modularity' module:

C:\third-party\test.jar

Which of the following commands executed from C:\ describes the module 'com.udayankhattry.modularity'?

A. java -d com.udayankhattry.modularity

B. java --describe-module com.udayankhattry.modularity

C. java -p third-party\test.jar -m com.udayankhattry.modularity

D. java -p third-party -d com.udayankhattry.modularity

4.1.2 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;

public class Test {
    public static void main(String[] args) {
        var list = new ArrayList<>();
        list.add( null );
        list.add( null );
        list.add( null );
        System. out .println(list.remove( 0 ) +
                    ":" + list.remove( null ));
    }
}
```

What will be the result of compiling and executing Test class?

A. true:true

B. true:false

C. null:true

D. null:null

E. NullPointerException is thrown at runtime

4.1.3 Which of the following correctly defines the Animal class?

A.	**public class** Animal {
	}
	package com.zoo;

B.	*/* Java Developer Comment. */*
	package com.zoo;
	public class Animal {
	}

C.	**public class** Animal {
	    package com.zoo;
	}

D.	**import** java.util.*;
	package com.zoo;
	public class Animal {
	}

4.1.4 For the code below, what should be the name of java file?

```
class Hello {
   public static void main(String... args) {}
}

public class HelloWorld {
   public static void main(String [] args) {}
}

class World {
   public static void main(String... args) {}
}
```

A. Hello.java

B. World.java

C. HelloWorld.java

D. helloworld.java

E. hello.java

F. world.java

4.1.5 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       System. out .println( "1" + "2" + "3"
                   == "1" + "2" + "3" );
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. 123

C. true

D. false

4.1.6 Consider below code of AvoidThreats.java file:

```
public class AvoidThreats {
    public static void evaluate(Threat t) { //Line n5
        t = new Threat(); //Line n6
        t. name = "PHISHING" ; //Line n7
    }

    public static void main(String[] args) {
       Threat obj = new Threat(); //Line n1
        obj.print(); //Line n2
        evaluate (obj); //Line n3
        obj.print(); //Line n4
    }
}

class Threat {
   String name = "VIRUS" ;

    public void print() {
        System. out .println( name );
    }
}
```

And the command :

```
java AvoidThreats.java
```

What is the result?

A.	VIRUS PHISHING	B.	PHISHING PHISHING
C.	VIRUS VIRUS	D.	null VIRUS
E.	null null	F.	None of the other options

4.1.7 Consider below code of Test.java file:

```
//Test.java
package com.udayankhattry.ocp1;

abstract class Animal {
    private String name ;

    Animal(String name) {
        this . name = name;
    }

    public String getName() {
        return name ;
    }
}

class Dog extends Animal {
    private String breed ;

    Dog(String breed) {
        this . breed = breed;
    }
    Dog(String name, String breed) {
        super (name);
        this . breed = breed;
    }

    public String getBreed() {
        return breed ;
    }
}

public class Test {
    public static void main(String[] args) {
        Dog dog1 = new Dog( "Beagle" );
        Dog dog2 = new Dog( "Bubbly" , "Poodle" );
        System. out .println(dog1.getName() + ":" +
                     dog1.getBreed() + ":" + dog2.getName() +
                    ":" + dog2.getBreed());
    }
}
```

What will be the result of compiling and executing above code?

A. Compilation error for Test Class

B. Compilation error for Animal(String) constructor

C. Compilation error for Dog(String) constructor

D. Compilation error for Dog(String, String) constructor

E. null:Beagle:Bubbly:Poodle

F. :Beagle:Bubbly:Poodle

4.1.8 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Base {
    String msg = "INHALE" ; //Line n1
}

class Derived extends Base {
    Object msg = "EXHALE" ; //Line n2
}
```

```
public class Test {
    public static void main(String[] args) {
       Base obj1 = new Base(); //Line n3
        Base obj2 = new Derived(); //Line n4
        Derived obj3 = (Derived) obj2; //Line n5
        var var = obj1. msg + "-" + obj2. msg +
               "-" + obj3. msg ; //Line n6
        System. out .println(var); //Line n7
    }
}
```

What will be the result of compiling and executing above code ?

A. Line n2 causes compilation error

B. Line n5 throws Exception at runtime

C. Line n6 causes compilation error

D. It executes successfully and prints INHALE-EXHALE-EXHALE

E. It executes successfully and prints INHALE-INHALE-EXHALE

F. It executes successfully and prints INHALE-INHALE-INHALE

G. None of the other options

4.1.9 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       StringBuilder sb =
              new StringBuilder( "Hakuna" ); //Line n3
        change (sb); //Line n4
        System. out .println(sb); //Line n5
    }

     private static void change(StringBuilder s) {
       s.append( "_Matata" ); //Line n9
    }
}
```

What will be the result of compiling and executing Test class?

A. Hakuna
B. _Matata
C. Hakuna_Matata
D. None of the other options

4.1.10 For the given code of Implementer.java file:

```
package com.udayankhattry.ocp1;

interface I01 {
    void m1();
}

public class Implementer extends Object implements I01 {
   protected void m1() {

     }
}
```

Which of the following statements is true?

A. Interface I01 causes compilation error as method m1 is not public
B. Implementer class declaration is not correct
C. Method m1() in Implementer class is not implemented correctly
D. None of the other options

4.1.11 For the given code:

```
package com.udayankhattry.ocp1;

interface Operator {
   int operate( int i, int j);
}

public class Test {
    public static void main(String[] args) {
       /*INSERT*/
```

```
        System. out .println(opr.operate( 10 , 20 ));
    }
}
```

Which of the following options successfully replace /*INSERT*/ such that on execution, 30 is printed on to the console?
Select ALL that apply .

A. Operator opr = (int x, int y) -> { return x + y; };

B. Operator opr = (x, y) -> { return x + y; };

C. Operator opr = (x, y) -> return x + y;

D. Operator opr = (x, y) -> x + y;

E. Operator opr = x, y -> x + y;

4.1.12 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var i = 4 ; //Line n1
        for (i = 0 ; i <= 2 ; i++){} //Line n2
        System. out .println(i); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A. 0

B. 2

C. 3

D. 4

E. Compilation error at Line n1

F. Compilation error at Line n2

4.1.13 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
class A {
    A() {
        this ( 1 );
        System. out .println( "M" );
    }

     A( int i) {
        System. out .println( "N" );
    }
}

class B extends A {

}

public class Test {
     public static void main(String[] args) {
         new B();
    }
}
```

What will be the result of compiling and executing above code?

A.	M	B.	N
C.	N M	D.	M N

4.1.14 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
     public static void main(String[] args) {
        StringBuilder sb = new StringBuilder(
                "ELECTROTHERMAL" ); //Line n1
         sb.setLength( 7 ); //Line n2
         System. out .print(sb.toString().strip()); //Line n3
         System. out .print( ":" ); //Line n4
         sb.setLength( 14 ); //Line n5
         System. out .println(sb.toString().strip()); //Line n6
```

```
    }
}
```

What will be the result of compiling and executing Test class?

A. THERMAL:ELECTROTHERMAL

B. ELECTRO:ELECTROTHERMAL

C. ELECTROTHERMAL:THERMAL

D. ELECTROTHERMAL:ELECTRO

E. ELECTRO:ELECTRO

F. ELECTRO:THERMAL

G. THERMAL:ELECTRO

H. THERMAL:THERMAL

4.1.15 Consider the code of TestPerson.java file:

```
package com.udayankhattry.ocp1;

class Person {
    String name ;
    int age ;

    Person() {
        Person( "Rohit" , 25 );
    }

    Person(String name, int age) {
        this . name = name;
        this . age = age;
    }
}

public class TestPerson {
    public static void main(String[] args) {
        Person p = new Person();
        System. out .println(p. name + ":" + p. age );
    }
}
```

There is a compilation error in the Person class.

Which two modifications, done independently, print "Rohit:25" on to the console?

A.	Add below code in the Person class:
	void Person(String name, int age) {
	    this . name = name;
	    this . age = age;
	}

B.	Replace Person("Rohit", 25) ; with super("Rohit", 25);
C.	Replace Person("Rohit", 25) ; with this("Rohit", 25);
D.	Replace Person("Rohit", 25) ; with this.Person("Rohit", 25);

4.1.16 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Leveller {
    int level();
}

public class Test {
   int i = 100 ; //Line n1

    Leveller level = () -> {
       int i = 200 ; //Line n2
       return this . i ; //Line n3
    };

    public static void main(String[] args) {
      System. out .println( new Test().
            level .level()); //Line n4
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error at Line n2

B. Compilation error at Line n3

C. Compilation error at Line n4

D. 100

E. 200

4.1.17 Given:

```
class TestException extends Exception {
    public TestException() {
        super ();
    }

     public TestException(String s) {
        super (s);
    }
}

public class Test {
    public void m1() throws __________ {
        throw new TestException();
    }
}
```

For the above code, fill in the blank with one option.

A. Exception

B. Object

C. RuntimeException

D. Error

4.1.18 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       StringBuilder sb = new StringBuilder( 100 );
       System. out .println(sb.length() + ":"
             + sb.toString().length());
    }
}
```

What will be the result of compiling and executing Test class?

A. 100:100
B. 100:0
C. 16:16
D. 16:0
E. 0:0

4.1.19 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int x = 7 ;
        boolean res = x++ == 7 && ++x == 9 || x++ == 9 ;
       System. out .println( "x = " + x);
       System. out .println( "res = " + res);
    }
}
```

What will be the result of compiling and executing Test class?

A.	x = 10
	res = true
		B.	x = 9
			res = true

C.	x = 10
	res = false
		D.	Compilation error

4.1.20 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        final boolean flag;
       flag = false ;
        while (flag) {
           System. out .println( "Good Night!" );
        }
    }
}
```

Which of the following statements is correct for above code?

A. Program compiles and executes successfully but produces no output

B. Compilation error

C. Infinite loop

D. It will print "Good Night!" once

4.1.21 Consider below code snippet:

```
interface ILog {
    default void log() {
        System. out .println( "ILog" );
    }
}

abstract class Log {
    public static void log() {
        System. out .println( "Log" );
    }
}

class MyLogger extends Log implements ILog {}
```

Which of the following statements is correct?

A. There is no compilation error in the above code

B. There is a compilation error in interface ILog

C. There is a compilation error in abstract class Log

D. There is a compilation error in MyLogger class

4.1.22 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        String text = "RISE " ;
        text = text + (text = "ABOVE " );
        System. out .println(text);
    }
}
```

What will be the result of compiling and executing Test class?

A. RISE RISE ABOVE

B. RISE ABOVE

C. ABOVE ABOVE

D. RISE ABOVE RISE

4.1.23 Consider below code snippet:

```
package com.udayankhattry.ocp1;

public class Test {
    String testNo ;
    String desc ;
     /*
    Other codes...
    */
}
```

Which of the options are correct so that instance variables 'testNo' and 'desc' are accessible only within 'com.udayankhattry.ocp1' package?

A.	No changes are necessary
B.	Change the instance variable declarations to: **private** String **testNo** ; **private** String **desc** ;
C.	Change the instance variable declarations to: **protected** String **testNo** ; **protected** String **desc** ;
D.	Change the instance variable declarations to: **public** String **testNo** ;

	public String **desc** ;

4.1.24 Given Code:

```
import java.io.*;

class ReadTheFile {
    static void print() { //Line n1
        throw new IOException(); //Line n2
    }
}

public class Test {
    public static void main(String[] args) { //Line n3
        ReadTheFile. print (); //Line n4
    }
}
```

Which 2 changes are necessary so that code compiles successfully?

A.	Replace Line n1 with **static void** print() **throws** Exception {
B.	Replace Line n1 with **static void** print() **throws** Throwable {
C.	Replace Line n3 with **public static void** main(String[] args) **throws** IOException {

D.	Surround Line n4 with below try-catch block:
	try {
	    ReadTheFile. *print* ();
	} **catch** (IOException e) {
	    e.printStackTrace();
	}

E.	Surround Line n4 with below try-catch block:
	try {
	    ReadTheFile. *print* ();
	} **catch** (IOException | Exception e) {
	    e.printStackTrace();
	}

F.	Surround Line n4 with below try-catch block:

```
try {
   ReadTheFile. print ();
} catch (Exception e) {
   e.printStackTrace();
}
```

4.1.25 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.List;

public class Test {
    public static void main(String[] args) {
        var list = List. of ( new String[]{ "A" , "BB" , "CCC" },
                new String[]{ "DD" , "E" }); //Line n1
        list.forEach(x ->
                System. out .print(x. length )); //Line n2
    }
}
```

What will be the result of compiling and executing Test class ?

A. Compilation error at Line n1

B. Compilation error at Line n2

C. 12321

D. 32

4.1.26 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int [] arr = { 3 , 2 , 1 };
        for ( int i : arr) {
            System. out .println(arr[i]);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	3 2 1	B.	1 2 3
C.	Compilation error	D.	An Exception is thrown at runtime

4.1.27 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int [] arr = { 1 , 2 , 3 , 4 , 5 };
        int x = 0 ;
        for ( /*INSERT*/ ) {
           x += arr[n];
       }
       System. out .println(x);
    }
}
```

Which 3 options, if used to replace /*INSERT*/, on execution will print 9 on to the console?

A. int n = 0; n < arr.length; n++

B. int n = 0; n < arr.length; n += 2

C. int n = 3; n < arr.length; n++

D. int n = 1; n < arr.length - 1; n++

E. int n = 1; n < arr.length; n += 2

4.1.28 java.sql.SQLException extends java.lang.Exception and java.sql.SQLWarning extends java.sql.SQLException

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.sql.*;
```

```
interface Multiplier {
    void multiply( int ... x) throws SQLException;
}

class Calculator implements Multiplier {
    public void multiply( int ... x) throws /*INSERT*/ {

     }
}
public class Test {
     public static void main(String[] args) {
         try {
            Multiplier obj = new Calculator(); //Line n1
             obj.multiply( 1 , 2 , 3 );
         } catch (SQLException e) {
            System. out .println(e);
         }
    }
}
```

Which of the options can be used to replace /*INSERT*/ such that there is no compilation error?
Select ALL that apply .

A. java.io.IOException

B. SQLException

C. SQLWarning

D. Throwable

E. RuntimeException

F. Error

G. Exception

H. NullPointerException

4.1.29 Given below the directory/file structure on Windows platform:

```
C:
+---my_codes
```

```
|   +---basic.calc
|   |   |   module-info.java
|   |   |
|   |   \---com
|   |       \---udayankhattry
|   |           \---ocp1
|   |               \---calc
|   |                   \---basic
|   |                           BasicCalculator.java
|   |                           Test.java
|   |
|   \---advance.calc
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                   \---calc
|                       \---advance
|                               AdvanceCalculator.java
|
\---classes
```

Below are the file details:-

C:\my_codes\basic.calc\module-info.java:

```
module basic.calc {
    /*INSERT-1*/
}
```

C:\my_codes\basic.calc\com\udayankhattry\ocp1\calc\basic\BasicCalc

```
package com.udayankhattry.ocp1.calc.basic;

import com.udayankhattry.ocp1.calc.advance.AdvanceCalculator;

public class BasicCalculator {
    public static void multiply( int i1, int i2, int i3) {
      System. out .println(
              AdvanceCalculator.multiply(i1, i2, i3));
    }
}
```

C:\my_codes\basic.calc\com\udayankhattry\ocp1\calc\basic\Test.java:

```
package com.udayankhattry.ocp1.calc.basic;

public class Test {
    public static void main(String[] args) {
      BasicCalculator.multiply( 2 , 3 , 4 );
    }
}
```

C:\my_codes\advance.calc\module-info.java:

```
module advance.calc {
    /*INSERT-2*/
}
```

C:\my_codes\advance.calc\com\udayankhattry\ocp1\calc\advance\Adv

```
package com.udayankhattry.ocp1.calc.advance;

public class AdvanceCalculator {
    public static int multiply( int ... nums) {
       int res = 1 ;
       for ( int i: nums) {
         res *= i;
      }
       return res;
    }
}
```

And the below command executed from C:\

```
javac -d classes --module-source-path my_codes -m basic.calc,advance.calc
```

Which of the following changes needs to be done such that above command successfully executes?

A.

Replace /*INSERT-1*/ with requires com.udayankhattry.ocp1.calc.advance;
and
Replace /*INSERT-2*/ with exports com.udayankhattry.ocp1.calc.advance;

B.

Replace /*INSERT-1*/ with requires com.udayankhattry.ocp1.calc.advance;
and
Replace /*INSERT-2*/ with exports advance.calc;

C.

Replace /*INSERT-1*/ with requires advance.calc;
and
Replace /*INSERT-2*/ with exports com.udayankhattry.ocp1.calc.advance;

D.

Replace /*INSERT-1*/ with requires advance.calc;
and
Replace /*INSERT-2*/ with exports advance.calc;

E.

Replace /*INSERT-1*/ with requires advance.calc;
and
Replace /*INSERT-2*/ with exports com.udayankhattry.ocp1.calc.advance to basic.calc;

F.

Given javac command executes successfully without making any changes.

4.1.30 Following statement in a Java program compiles successfully:

student.report(course);

What can you say for sure?

A. student is the reference variable name

B. student is the class name

C. report is the method name

D. course must be of String type

4.1.31 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       var res = "" ; //Line n1
       String [] arr = { "Dog" , null , "Friendly" };
       for (String s : arr) { //Line n2
          res += String. join ( "-" , s); //Line n3
       }
      System. out .println(res); //Line n4
    }
}
```

What will be the result of compiling and executing Test class?

A. DogFriendly

B. Dog-Friendly

C. DognullFriendly

D. Dog-null-Friendly

E. An exception is thrown at runtime

F. Compilation error

4.1.32 Consider below codes of 3 java files:

```
//M.java
package com.udayankhattry.ocp1;

public class M {
    public void printName() {
      System. out .println( "M" );
    }
}


//N.java
package com.udayankhattry.ocp1;

public class N extends M {
    public void printName() {
      System. out .println( "N" );
    }
}
```

```
//Test.java
package com.udayankhattry.ocp1.test;

import com.udayankhattry.ocp1.*;

public class Test {
    public static void main(String[] args) {
       M obj1 = new M();
       N obj2 = (N)obj1;
       obj2.printName();
    }
}
```

What will be the result of compiling and executing Test class?

A. It executes successfully and prints M on to the console

B. It executes successfully and prints N on to the console

C. Compilation error

D. An exception is thrown at runtime

4.1.33 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   |  [POSITION-1]
|   |
|   \---office
|       |  [POSITION-2]
|       |
|       \---com
|           |  [POSITION-3]
|           |
|           \---xyz
|               +---hr
|               |   |  Hiring.java
|               |   |   [POSITION-4]
|               |   |
|               |   \---training
|               |           JavaTraining.java
|               |           [POSITION-5]
|               |
|               \---projects
|                       JavaProject.java
|
\---cls
```

[POSITION-1], [POSITION-2], [POSITION-3], [POSITION-4], [POSITION-5] are possible place holders for module-info.java file.

Below are the file details:-

C:\src\office\com\xyz\hr\Hiring.java:

```
package com.xyz.hr;

public class Hiring {
   //Lots of valid codes
}
```

C:\src\office\com\xyz\hr\training\JavaTraining.java:

```
package com.xyz.hr.training;

public class JavaTraining {
   //Lots of valid codes
}
```

C:\src\office\com\xyz\projects\JavaProject.java:

```
package com.xyz.projects;

public class JavaProject {
   //Lots of valid codes
}
```

Contents of module-info.java:

```
module office {
    exports com.xyz.hr.training;
}
```

And consider the command to be executed from C:\

```
javac -d cls --module-source-path src -m office
```

What should be the location of module-info.java file such that above javac command executes successfully?

A. [POSITION-1]

B. [POSITION-2]

C. [POSITION-3]

D. [POSITION-4]

E. [POSITION-5]

4.1.34 Consider below code of Animal.java file:

```
class Cat {
    public static void main(String args) {
       System. out .println( "Cat" );
   }
}

class Dog {
    public static void main(String [] args) {
       System. out .println( "Dog" );
   }
}
```

And the command:

```
java Animal.java
```

What is the result?

A.	Cat	B.	Dog
C.	Cat Dog	D.	Dog Cat
E.	Above command causes error		

4.1.35 Consider below code of TestStudent.java file:

```
package com.udayankhattry.ocp1;

//TestStudent.java
class Student {
   String name ;
    int age ;
    boolean result ;
    double height ;
}

public class TestStudent {
    public static void main(String[] args) {
        Student stud = new Student(); //Line n1
         System. out .println(stud. name + stud. height +
                stud. result + stud. age ); //Line n2
    }
}
```

What will be the result of compiling and executing TestStudent class?

A. null0.0false0

B. null0false0

C. null0.0ffalse0

D. null0.0true0

E. Compilation error

4.1.36 java.se is a/an ____________ module.

A. aggregator

B. unnamed

C. non-standard

D. deprecated

4.1.37 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;
```

```
public class Test {
    public static void main(String[] args) {
       Integer i = 10 ;
       List<Integer> list = new ArrayList<>();
       list.add(i);
       list.add(i *= 2 );
       list.add(i);

          list.removeIf(i -> i == 10 );

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [10, 20, 10]

B. [10, 20, 20]

C. [20, 10]

D. [20, 20]

E. [20]

F. []

G. Compilation Error

4.1.38 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;
import java.io.IOException;

class Base {
   Base() throws IOException {
      System. out .print( 1 );
   }
}

class Derived extends Base {
   Derived() throws FileNotFoundException {
      System. out .print( 2 );
```

```
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        new Derived();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in both Base and Derived classes

B. Compilation error only in Base class

C. Compilation error only in Derived class

D. Test class executes successfully and prints 12 on to the console

E. Test class executes successfully and prints 21 on to the console

4.1.39 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Operator< T > {
    public abstract T operation( T t1, T t2);
    private void temp() {
       System. out .println( "private method" );
    }
}

public class Test {
    public static void main(String[] args) {
       Operator<String> opr1 = (s1, s2) -> s1 + s2;
       Operator<Integer> opr2 = (i1, i2) -> i1 + i2;
       opr1.operation( "Think" , "First" );
       opr2.operation( 10 , 40 );
    }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation error

B.	ThinkFirst 50
C.	Program compiles and executes successfully but nothing is printed on to the console
D.	ThinkFirst 1040

4.1.40 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int [] arr1 = { 10 , 100 , 1000 }; //Line n1
        char [] arr2 = { 'x' , 'y' , 'z' }; //Line n2
        arr1 = arr2; // Line n3
        for ( int i = 0 ; i < arr1. length ; i++) {
           System. out .print(arr1[i] + " " ); //Line n4
        }
    }
}
```

ASCII code of 'x' is 120, 'y' is 121 and z is 122. What will be the result of compiling and executing Test class?

A. 10 100 1000

B. x y z

C. 120 121 122

D. 0 0 0

E. Compilation error

4.1.41 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<StringBuilder> animals = new ArrayList<>();
       animals.add( new StringBuilder( "Walrus" ));
       animals.add( new StringBuilder( "Anaconda" ));
       animals.add( new StringBuilder( "Alligator" ));
       animals.add( new StringBuilder( "Dog" ));

          for ( int i = 0 ; i < animals.size(); i++) {
           if (i == 0 ) {
              animals.remove(
               new StringBuilder( "Alligator" ));
           }
       }

          System. out .println(animals);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Walrus, Anaconda, Alligator, Dog]

B. [Walrus, Dog]

C. An exception is thrown at runtime

D. [Walrus, Anaconda, Dog]

4.1.42 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String [] args) {
       List<Integer> list = new ArrayList<Integer>();
       list.add( 2 );
       list.add( 1 );
       list.add( 0 );

          list.remove(list.indexOf( 0 ));
```

```
        System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. [1, 0]

D. [2, 1]

E. [2, 1, 0]

4.1.43 Consider below code of Utils.java file:

```
package com.udayankhattry.ocp1;

public class Utils {
    public static void main(String... args) {
        System. out .println( "Inside Utils class" );
    }
}
```

Location of Utils.java file:

```
F:.
└───codes
    ├───src
    │   Utils.java
    │
    └───classes
```

You are currently at 'F:':

```
F:\>
```

Which of the following javac commands, typed from above location, will generate Utils.class file structure under 'classes' directory?

```
F:.
└───codes
    ├───src
    │   Utils.java
    │
```

```
│
└───classes
    └───com
        └───udayankhattry
            └───ocp1
                Utils.class
```

A.	javac -d codes\classes\ codes\src\Utils.java
B.	javac -d classes\ src\Utils.java
C.	javac -d codes\classes\ codes\src\com.udayankhattry.ocp1.Utils.java
D.	javac -d classes\ src\com.udayankhattry.ocp1.Utils.java
E.	javac -d codes\classes\ codes\src\com\udayankhattry\ocp1\Utils.java

4.1.44 Consider below public classes: com.discount.shopping.Product, com.discount.shopping.Order and com.discount.shopping.payments.Discount.

The requirement is to create a modular application for above classes. Module name must be com.discount.shopping and above 3 classes must readable by all the modules, which of the following definitions of module-info.java file will achieve this?

A.	module com.discount.shopping {
	}

B.	module com.discount.shopping {
	exports com.discount.shopping.Product;
	exports com.discount.shopping.Order;
	exports com.discount.shopping.payments.Discount;
	}

C.	module com.discount.shopping {
	exports com.discount.shopping.*;
	exports com.discount.shopping.payments.*;
	}

D.	module com.discount.shopping {
	exports com.discount.shopping.*;
	}

E.	module shopping {
	exports com.discount.shopping;
	exports com.discount.shopping.payments;
	}

F.	module com.discount.shopping {
	exports com.discount.shopping to all;

	exports com.discount.shopping.payments to all;
	}

G.	module com.discount.shopping {
	export com.discount.shopping;
	export com.discount.shopping.payments;
	}

H.	None of the other options

4.1.45 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        try { //outer
            try { //inner
                System. out .println( 1 / 0 );
            } catch (ArithmeticException e) {
                System. out .println( "INNER" );
            } finally {
                System. out .println( "FINALLY 1" );
            }
        } catch (ArithmeticException e) {
            System. out .println( "OUTER" );
        } finally {
            System. out .println( "FINALLY 2" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	INNER FINALLY 1	B.	OUTER FINALLY 2
C.	INNER FINALLY 2	D.	INNER FINALLY 1 FINALLY 2

4.1.46 Consider below code:

```
public class Test {
    public static void main(String[] args) {
        long a = 100_00l ;
        int b = 100 ;
        float c = 2.02f ;
        double d = 10_0.35d ;
       c = a;
       d = a;
       c = d;
       d = c;
       c = b;
       b = c;
       b = ( int )d;
    }
}
```

How many statement(s) cause(s) compilation failure?

A. One

B. Two

C. Three

D. Four

E. Five

F. Six

4.1.47 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<String> list = new ArrayList<>();
       list.add( 0 , "Array" );
       list.add( 0 , "List" );

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Array]
B. [List]
C. [Array, List]
D. [List, Array]
E. An exception is thrown at runtime

4.1.48 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface ILogger {
    void log();
}

public class Test {
    public static void main(String[] args) {
       ILogger [] loggers = new ILogger[ 2 ]; //Line n1
        for (ILogger logger : loggers)
           logger.log(); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error
B. Line n2 causes compilation error
C. An exception is thrown at runtime
D. No output is displayed but program terminates successfully

4.1.49 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int [] arr = { 10 , 20 , 30 }; //Line n1
        int i = 0 ;
        arr[i++] = arr[++i] = 40 ; //Line n2
        for ( var x : arr) //Line n3
            System. out .println(x);
    }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation error at Line n2	B.	An exception is thrown by Line n2
C.	10 20 30	D.	10 40 30
E.	40 40 30	F.	10 40 40
G.	40 20 40		

4.1.50 Consider below codes of 2 java files:

```
//Counter.java
package com.udayankhattry.ocp1;

public interface Counter {
    int count = 10 ; //Line n1
}


//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        Counter [] arr = new Counter[ 2 ]; //Line n2
        for (Counter ctr : arr) {
            System. out .print(ctr.count); //Line n3
        }
    }
}
```

Which of the following statements is correct?

A. Only Line n1 causes compilation error

B. Only Line n2 causes compilation error

C. Line n1 and Line n2 cause compilation error

D. Only Line n3 causes compilation error

E. Line n3 throws an exception at runtime

F. Test class compiles successfully and on execution prints 1010 on to the console

4.1.51 Consider below code snippet:

```
public static void process( /*INSERT*/ list) {
    list.add( 100 ); //Line n2
     int x = list.get( 0 ); //Line n3
     System. out .println(list.size() + ":" + x);
}
```

Which of the following options, if used to replace /*INSERT*/, compiles successfully?

A. java.util.List

B. java.util.List<Integer>

C. java.util.List<Object>

D. java.util.List<int>

4.1.52 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        int num = 10 ;
        if (num++ == num++) {
           System. out .println( "EQUAL " + num);
        } else {
           System. out .println( "NOT EQUAL " + num);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. EQUAL 12

B. EQUAL 11

C. NOT EQUAL 12

D. NOT EQUAL 11

4.1.53 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String str = "*" ;
        /*INSERT*/
        System. out .println(str);
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print *** on to the console?**
Select ALL that apply.

A.	str.repeat(5);
B.	str = str.repeat(5);
C.	String.repeat(str, 5);
D.	str = String.repeat(str, 5);
E.	str *= 5 ;
F.	str = str * 5 ;
G.	for(var var = 0 ; var < 4 ; var++) {

	str += "*" ;
	}

H.	for(int i = 0 ; i < 5 ; i++) {
	str += "*" ;
	}

4.1.54 Which of the following is a checked Exception?

A. ClassCastException
B. FileNotFoundException
C. ExceptionInInitializerError
D. RuntimeException

4.1.55 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String s1 = "1Z0-815" ;
       String s2 = "1Z0-815" + "" ;
       System. out .println(s1 == s2);
    }
}
```

What will be the result of compiling and executing Test class?

A. 1Z0-815
B. true
C. false
D. Compilation error

4.1.56 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;
```

```
public class Test {
    public static void main(String[] args) {
      List<String> days = new ArrayList<>();
      days.add( "SUNDAY" );
      days.add( "SUNDAY" );
      days.add( "MONDAY" );
      System. out .println(days.size());
      days.clear();
      System. out .println(days.size());
    }
}
```

What will be the result of compiling and executing Test class?

A.	3 3	B.	3 0
C.	2 0	D.	2 2
E.	An exception is thrown at runtime		

4.1.57 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
 boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
       printNumbers (i -> i % 2 != 0 );
    }

     private static void printNumbers(
             Predicate<Integer> predicate) {
       for ( int i = 1 ; i <= 10 ; i++) {
            if (predicate.test(i)) {
                System. out .print(i);
            }
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. 12345678910

B. 1234567891011

C. 246810

D. 13579

E. 1357911

4.1.58 Your Vendor has provided you a modular jar to test the database connection API.

Below are the Jar details:

Jar File name: dbconnection.jar

Module name: dbconnection

Class name: com.database.util.DBConnect

Method: public static void connect(String dbURL, String username, String passowrd) {...}

module-info.java contents:

```
module dbconnection {
    exports com.database.util to dbtester;
}
```

File/Directory structure on your Windows platform is:

```
C:
+---source
|   \---dbtester
|       |  module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---test
|                DBTester.java
|
+---classes
\---jars
      dbconnection.jar
```

C:\source\dbtester\module-info.java:

```
module dbtester {
    requires dbconnection;
}
```

C:\source\dbtester\com\udayankhattry\test\DBTester.java:

```
package com.udayankhattry.test;

import com.database.util.DBConnect;

public class DBTester {
    public static void main(String... args) {
      DBConnect.connect( "temp" , "admin" , "pT12uY3$$" );
   }
}
```

Which of the following javac commands compile your module code successfully?

A.	javac -d classes --module-source-path source;jars\dbconnection.jar -m dbtester
B.	javac -d classes --module-source-path source -p jars\dbconnection.jar -m dbtester
C.	javac -d classes --module-source-path source,jars\dbconnection.jar -m dbtester
D.	javac -d classes --module-source-path src --module-class-path jars\dbconnection.jar -m dbtester

4.1.59 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Counter {
    static int ctr = 0 ;
    int count = 0 ;
}

public class Test {
    public static void main(String[] args) {
       Counter ctr1 = new Counter();
       Counter ctr2 = new Counter();
       Counter ctr3 = new Counter();

          for ( int i = 1 ; i <= 5 ; i++ ) {
          ctr1. ctr ++;
          ctr1. count ++;
          ctr2. ctr ++;
          ctr2. count ++;
          ctr3. ctr ++;
          ctr3. count ++;
       }

          System. out .println(ctr3. ctr + ":" + ctr3. count );
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. 5:5 is printed on to the console

C. 15:15 is printed on to the console

D. 15:5 is printed on to the console

4.1.60 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

class Student {
    private String name ;
    private int age ;

     Student(String name, int age) {
        this . name = name;
        this . age = age;
    }

     public String toString() {
        return "Student[" + name + ", " + age + "]" ;
    }
}

public class Test {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add( new Student( "James" , 25 ));
        students.add( new Student( "James" , 27 ));
        students.add( new Student( "James" , 25 ));
        students.add( new Student( "James" , 25 ));

            students.remove( new Student( "James" , 25 ));

            for (Student stud : students) {
             System. out .println(stud);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	Student[James, 27] Student[James, 25] Student[James, 25]	B.	Student[James, 25] Student[James, 27] Student[James, 25]
C.	Student[James, 27]	D.	Student[James, 25] Student[James, 27] Student[James, 25] Student[James, 25]

4.1.61 Consider below code of Greetings.java file:

```
package com.udayankhattry.ocp1;

public class Greetings {
    String msg = null ;

    public Greetings() {
        this ( "Good Morning!" );
    }

    public Greetings(String str) {
        msg = str;
    }

    public void displayMsg() {
        System. out .println( msg );
    }

    public static void main(String [] args) {
        Greetings g1 = new Greetings();
        Greetings g2 = new Greetings( "Good Evening!" );
        g1.displayMsg();
        g2.displayMsg();
    }
}
```

What will be the result of compiling and executing Greetings class?

A.	null Good Evening!	B.	Good Morning! Good Evening!
C.	Good Morning! null	D.	null null

4.1.62 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int ctr = 100 ;
        one: for ( var i = 0 ; i < 10 ; i++) {
            two: for ( var j = 0 ; j < 7 ; j++) {
                three: while ( true ) {
                    ctr++;
                    if (i > j) {
                        break one;
                    } else if (i == j) {
                        break two;
                    } else {
                        break three;
                    }
                }
            }
        }
        System. out .println(ctr);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error
B. 100
C. 101
D. 102
E. 103
F. 104
G. 105
H. 106

4.1.63 Consider below code of TestStyle. java file:

```
package com.udayankhattry.ocp1;

class Style {
    String pattern = "*" ;
}

public class TestStyle {
    public static void main(String[] args) {
        var style = new Style(); //Line n1
        System. out .println(style. pattern .
                repeat( 5 ).length()); //Line n2
    }
}
```

What will be the result of compiling and executing TestStyle class?

A. 0
B. 1
C. 5
D. null
E. An exception is thrown at runtime
F. Compilation error at Line n1
G. Compilation error at Line n2

4.1.64 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Printable {
    void print();
    boolean equals(Object obj);
}

public class Test {
    public static void main(String[] args) {
       Printable obj = () ->
              System. out .println( "AIM HIGH" );
       obj.print();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error
B. AIM HIGH
C. No output
D. An exception is thrown at runtime

4.1.65 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
       char [][] arr = {
         { 'A' , 'B' , 'C' },
         { 'D' , 'E' , 'F' },
         { 'G' , 'H' , 'I' }
      };

         for ( int i = 0 ; i < arr. length ; i++) {
          for ( int j = 0 ; j < arr[i]. length ; j++) {
            System. out .print(arr[i][ 1 ]);
         }
         System. out .println();
      }
   }
}
```

What will be the result of compiling and executing Test class?

A.	ABC DEF GHI	B.	BBB EEE HHH
C.	AAA DDD GGG	D.	CCC FFF III

4.1.66 What will be the output of compiling and executing the Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       int x = 2 ;
       switch (x) {
          default :
            System. out .println(
                "Still no idea what x is" );
            case 1 :
              System. out .println( "x is equal to 1" );
               break ;
            case 2 :
              System. out .println( "x is equal to 2" );
               break ;
            case 3 :
              System. out .println( "x is equal to 3" );
               break ;
        }
    }
}
```

A.	x is equal to 2
B.	Compilation error
C.	Still no idea what x is x is equal to 1
D.	Produces no output

4.1.67 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   \---messages
|       |  module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                       Main.java
|                       Test.java
|
+---bin
|   \---messages
|       |  module-info.class
|       |
|       \---com
|           \---udayankhattry
|               \---ocp1
|                       Test.class
|                       Main.class
|
\---jars
        messages.jar
```

Contents of C:\src\messages\com\udayankhattry\ocp1\Main.java file:

```java
package com.udayankhattry.ocp1;

public class Main {
    public static void main(String... args) {
        System. out .println( "FOCUS AND WIN" );
    }
}
```

Contents of C:\src\messages\com\udayankhattry\ocp1\Test.java file:

```java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String... args) {
        System. out .println( "DRILL YOUR SKILLS" );
    }
}
```

Contents of C:\src\messages\module-info.java file:

```java
module messages {
}
```

'messages.jar' file was created by executing below command from C:\

```
C:\>jar -cfe jars\messages.jar com.udayankhattry.ocp1.Test -C bin\messages .
```

Which of the following commands, if executed from C:\,
displays FOCUS AND WIN on to the console?
Select ALL that apply.

A.	java -p jars\messages.jar -m messages
B.	java -p jars -m messages
C.	java -p bin -m messages

D.	java -p bin -m messages/com.udayankhattry.ocp1.Main
E.	java -p jars -m messages/com.udayankhattry.ocp1.Main
F.	java -m messages/com.udayankhattry.ocp1.Main -p jars
G.	java -m messages/com.udayankhattry.ocp1.Main

4.1.68 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

abstract class Log {
    abstract long count(); //Line n1
    abstract Object get(); //Line n2
}

class CommunicationLog extends Log {
    int count() { //Line n3
        return 100 ;
    }

     String get() { //Line n4
        return "COM-LOG" ;
    }
}

public class Test {
    public static void main(String[] args) {
        Log log = new CommunicationLog(); //Line n5
         System. out .print(log.count());
        System. out .print(log.get());
    }
}
```

Which of the following statement is correct ?

A. Line n3 causes compilation error

B. Line n4 causes compilation error

C. Line n5 causes compilation error

D. Given code compiles successfully and on execution prints 100COM-LOG on to the console

4.1.69 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String [] arr = { "1st" , "2nd" , "3rd" ,
            "4th" , "5th" };
      String place = "faraway" ;
      System. out .println(
            arr[place.indexOf( "a" , 3 )]); //Line n1
    }
}
```

What will be the result of compiling and executing Test class?

A. 1st

B. 3rd

C. 5th

D. 2nd

E. 4th

F. An exception is raised by Line n1

4.1.70 Suppose you have created a java file, "MyClass.java".

Which of the following commands will compile above java file?

A. javac MyClass

B. java MyClass

C. javac MyClass.class

D. javac MyClass.java

E. java MyClass.java

4.1.71 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Base {
   int id = 1000 ; //Line n1
```

```
    Base() {
      Base(); //Line n2
    }

    void Base() { //Line n3
       System. out .println(++ id ); //Line n4
    }
}

class Derived extends Base {
   int id = 2000 ; //Line n5

    Derived() {} //Line n6

    void Base() { //Line n7
       System. out .println(-- id ); //Line n8
    }
}

public class Test {
    public static void main(String[] args) {
      Base base = new Derived(); //Line n9
    }
}
```

What will be the result of compiling and executing above code ?

A. 1000

B. 1001

C. 999

D. 2000

E. 1999

F. 2001

G. 0

H. -1

I. Compilation error

J. An exception is thrown

4.1.72 Consider below code of Wall.java file:

```
package com.udayankhattry.ocp1;

public class Wall {
    public static void main(String args[]) {
        double area = 5.7 ;
       String color;
        if (area < 7 )
           color = "BLUE" ;

          System. out .println(color);
    }
}
```

What will be the result of compiling and executing Wall class?

A. BLUE

B. null

C. An exception is thrown at runtime

D. Compilation error

4.1.73 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var x = "100" ; //Line n1
        var y = 100 ; //Line n2
        System. out .println(x + y); //Line n3
    }
}
```

What is the result of compiling and executing Test class?

A. Only Line n1 causes compilation error

B. Only Line n2 causes compilation error

C. Only Line n3 causes compilation error

D. Only Line n1 and Line n3 cause compilation error

E. Line n1, Line n2 and Line n3 cause compilation error

F. Code compiles successfully and prints 200 on to the console

G. Code compiles successfully and prints 100100 on to the console

4.1.74 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String furniture = new String(
            new char [] { 'S' , 'o' , 'f' , 'a' });
       switch (furniture) {
          default :
            System. out .println( "CHAIR" );
          case "Recliner" :
            System. out .println( "RECLINER" );
          case "Sofa" :
            System. out .println( "SOFA" );
          case "Bed" :
            System. out .println( "BED" );
             break ;
      }
   }
}
```

A.	CHAIR	B.	SOFA
C.	SOFA BED	D.	CHAIR RECLINER SOFA BED

4.1.75 Which of these keywords can be used to prevent inheritance of a class?

A. constant

B. super

C. final

D. class

E. const

4.1.76 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.sql.SQLException;

public class Test {
    private static void getData() throws SQLException {
        try {
            throw new SQLException();
        } catch (Exception e) {
            e = new SQLException();
            throw e;
        }
    }

    public static void main(String[] args) {
        try {
            getData ();
        } catch (SQLException e) {
            System. out .println( "SQL" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Method getData() causes compilation error

B. Method main(String []) causes compilation error

C. SQL is printed on to the console and program terminates successfully

D. Program ends abruptly

4.1.77 Does modular JDK improves performance and security of the application?

A. Yes

B. No

4.1.78 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var m = 10 ; //Line n1
        var n = 20 ; //Line n2
       /*INSERT*/ p = m = n = 30 ; //Line n3
        System. out .println(m + n + p); //Line n4
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print 90 on to the console?
Select ALL that apply.

A. byte

B. short

C. int

D. long

E. float

F. double

G. var

4.1.79 Consider below method declarations/definitions:

1.

```
 void test();
```

2.

```
default void test(String name) {
    System. out .println( "Testing " + name);
}
```

3.

```
static void test( int x) {
    System. out .println(x);
}
```

4.

```
    private default void log1() {}
```

5.

```
    private void log2() {}
```

6.

```
    private static void log3() {}
```

How many of the above declarations/definitions can be used inside an interface?

A. One declaration/definition

B. Two declaration(s)/definition(s)

C. Three declaration(s)/definition(s)

D. Four declaration(s)/definition(s)

E. Five declaration(s)/definition(s)

F. All six declaration(s)/definition(s)

4.1.80 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        String text = "ONE " ;
        System. out .println(text.concat(
                text.concat( "ELEVEN " )).trim());
    }
}
```

What will be the result of compiling and executing Test class?

A. ONE ELEVEN

B. ONE ONE ELEVEN

C. ONE ELEVEN ONE ELEVEN

D. ONE ELEVEN ONE

4.2 Answers of Practice Test - 4 with Explanation

4.1.1 Answer: D

Reason :

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
(to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

--describe-module or -d: It should be followed by module name and it describes the specified module.
If you pass the system module name, then it describe passed system module, e.g.
java -d java.sql

But to use it with application modules, module path needs to be specified.
Hence, below options are correct:
java -p third-party -d com.udayankhattry.modularity (given option)
or
java -p third-party\test.jar -d com.udayankhattry.modularity

Commands `java -d com.udayankhattry.modularity` and `java --describe-module com.udayankhattry.modularity` complain for missing module as -p option is missing.

Command `java -p third-party\test.jar -m com.udayankhattry.modularity` will try to execute the Main-class specified in MANIFEST.MF file of test.jar.

4.1.2 Answer: C

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

Given statement:
var list = new ArrayList<>(); => list refers to an ArrayList of Object type, because Generic type is not defined on the right side.

It is possible to add null to ArrayList instance.
Initially list has 3 elements: [null, null, null].
remove(int) returns the deleted member of the list. In this case `list.remove(0);` returns null as null was deleted from the 0th index. So, list is left with 2 elements: [null, null].
return(Object) returns true if deletion was successful otherwise false. In this case `list.remove(null)` removes first null from the list and returns true and list is left with just one element: [null].
Hence, the output is: 'null:true'.

4.1.3 Answer: B

Reason :
If package is used, then it should be the first statement. But javadoc and developer comments are not considered as java statements, so a class can have developer and javadoc comments before the package statement.
If import and package both are available, then correct order is package, import, class declaration.

4.1.4 Answer: C

Reason :
Java is case sensitive language. File name should match with public class's name, which is "HelloWorld".
Please note: "helloworld" is different from "HelloWorld".

4.1.5 Answer: C

Reason :
Operator + has higher precedence over == and + operator is left to right associative, so let's group the given expression:
"1" + "2" + "3" == "1" + "2" + "3"
= ("1" + "2") + "3" == "1" + "2" + "3"
= ("1" + "2") + "3" == ("1" + "2") + "3"
= (("1" + "2") + "3") == (("1" + "2") + "3")
Let's solve it now:
= ("12" + "3") == (("1" + "2") + "3")
= "123" == (("1" + "2") + "3")
= "123" == ("12" + "3")
= "123" == "123"
= true

Please note that Strings computed by concatenation at compile time are referred by String Pool. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc. This means the result of constant expression is calculated at compile time and later referred by String Pool.
For the given question "123" is a String pool object and that is why true is returned by the given expression.

4.1.6 Answer: C

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
   (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java AvoidThreats.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java AvoidThreats.java` is equivalent to (but not exactly same as):

```
javac -d <memory> AvoidThreats.java
java -cp <memory> AvoidThreats
```

Hence, in this case `java AvoidThreats.java` command executes main(String[]) method defined in AvoidThreats class.

Threat class doesn't specify any constructor, hence Java compiler adds below default constructor:
Threat() {super();}

Line n1 creates an instance of Threat class and initializes instance variable 'name' to "VIRUS". Variable 'obj' refers to this instance.
Line n2 prints VIRUS on to the console.
Line n3 invokes evaluate(Threat) method, as it is a static method defined in AvoidThreats class, hence `evaluate(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 't' copies the content of variable 'obj' (which stores the address to Threat instance created at Line n1). 't' also refers to the same instance referred by 'obj'.

On execution of Line n6, another Threat instance is created, its instance variable 'name' refers to "VIRUS" and 't' starts referring to this newly created instance of Threat class. Variable 'obj' of main(String[]) method still refers to the Threat instance created at Line n1. So, 'obj' and 't' now refer to different Threat instances.

Line n7, assigns "PHISHING" to the 'name' variable of the instance referred by 't'. evaluate(Threat) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg still refers to "VIRUS", so this statement prints VIRUS on to the console.

Hence in the output, you get:

```
VIRUS
VIRUS
```

4.1.7 Answer: C

Reason :
abstract class can have constructors and it also possible to have abstract

class without any abstract method. So, there is no issue with Animal class.

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.

Inside Animal class Constructor, compiler adds super(); => Animal(String name) { super(); this.name = name; }, super() in this case invokes the no-argument constructor of Object class and hence no compilation error here.

Compiler changes Dog(String) constructor to: Dog(String breed) { super(); this.breed = breed; }. No-argument constructor is not available in Animal class and as another constructor is provided, java compiler doesn't add default constructor. Hence Dog(String) constructor causes compilation error.

There is no issue with Dog(String, String) constructor and code in Test class is also fine.

4.1.8 Answer: E

Reason :
Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

At Line n3, obj1 is of Base type and refers to an instance of Base class.

At Line n4, obj2 is of Base type and refers to an instance of Derived class.

At Line n5, as obj2 refers to an instance of Derived class, hence typecasting it to Derived type doesn't cause any Exception. obj3 is of Derived type and refers to an instance of Derived class.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

Let's check the expression of Line n6:
obj1.msg + "-" + obj2.msg + "-" + obj3.msg;
=> (obj1.msg + "-") + obj2.msg + "-" + obj3.msg; //+ operator is left to right associative and behaves as concatenation operator as one of the operand is of String type.
=> ((obj1.msg + "-") + obj2.msg) + "-" + obj3.msg;
=> (((obj1.msg + "-") + obj2.msg) + "-") + obj3.msg;
Let's solve the expression now:
=> ((("INHALE" + "-") + obj2.msg) + "-") + obj3.msg; //obj1 is of Base type, hence obj1.msg refers to "INHALE"
=> (("INHALE-" + obj2.msg) + "-") + obj3.msg;
=> (("INHALE-" + "INHALE") + "-") + obj3.msg; //obj2 is of Base type, hence obj2.msg refers to "INHALE"
=> ("INHALE-INHALE" + "-") + obj3.msg;
=> "INHALE-INHALE-" + obj3.msg;
In above expression, left operand is of String type and right operand is of Object type, so toString() method is invoked. So, given expression is similar to:
=> "INHALE-INHALE-" + obj3.msg.toString();
=> "INHALE-INHALE-" + "EXHALE"; //As obj3.msg is of Object type and refers to an instance of String type, hence toString() method on "EXHALE" instance is invoked and this returns "EXHALE".
=> "INHALE-INHALE-EXHALE";

So, at Line n6, variable 'var' is of String type and refers to "INHALE-INHALE-EXHALE".
Line n7 prints INHALE-INHALE-EXHALE on to the console.

4.1.9 Answer: C

Reason :
At Line n3, 'sb' refers to StringBuilder object {"Hakuna"}.
When change method is called, both the variables 's' and 'sb' refer to same StringBuilder object {"Hakuna"}.
Line n9 modifies the passed object and appends "_Matata" to it. As a result, 's' now refers to "Hakuna_Matata" and 'sb' also refers to "Hakuna_Matata".
So, when control goes back to calling method main(String[]), Line n5

prints "Hakuna_Matata" on to the console.

4.1.10 Answer: C

Reason :
void m1(); in interface I01 is equivalent to `public abstract void m1();`.
So method m1() is implicitly public and abstract.
In java, a class can extend from only one class but can implement multiple interfaces. Correct keywords are: extends and implements. So, class declaration is correct.
As method m1() is implicitly public in I01, hence overriding method in Implementer class should also be public. But it is protected and hence compiler complains.

4.1.11 Answer: A,B,D

Reason :
Operator opr = (int x, int y) -> { return x + y; }; => ✓ operate(int, int) method accepts two int type parameters and returns the addition of passed parameters.
Operator opr = (x, y) -> { return x + y; }; => ✓ Type is removed from left part, type inference handles it.
Operator opr = (x, y) -> return x + y; => ✗ Compilation error, if there is only one statement in the right side then semicolon inside the body, curly brackets and return keyword(if available) can be removed. But all should be removed. You can't just remove one and leave others.
Operator opr = (x, y) -> x + y; => ✓ Semicolon inside the body, curly brackets and return keyword, all 3 are removed from right side.
Operator opr = x, y -> x + y; => ✗ Compilation error, if there are no parameters or more than one parameter available, then parentheses or round brackets '()' cannot be removed from left side.

4.1.12 Answer: C

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

At Line n1, 'i' infers to int type, there is no issue at Line n1.

At Line n2, variable 'i' is reinitialized to 0 and this loop executes 3 times, for i = 0, i = 1 and i = 2. For i = 3, control goes out of the for loop.
Now, as i is declared outside for loop, hence it is accessible outside loop body.
Line n3 prints 3 to the console.

4.1.13 Answer: C

Reason :
Default constructor added by Java compiler in B class is:

```
B() {
    super();
}
```

On executing new B(); statement, class B's default constructor is invoked, which invokes no-argument constructor of class A [super();].

no-argument constructor of class A invokes parameterized constructor of class A [this(1);].

N is printed first and after that M is printed.

4.1.14 Answer: E

Reason :
strip() method of String class (available since Java 11) returns a string whose value is this string, with all leading and trailing white space removed.
To find out what is white space character in Java, check Character.isWhitespace(int) method, you will find below:
A character is a Java whitespace character if and only if it satisfies one of the following criteria:
It is a Unicode space character (SPACE_SEPARATOR, LINE_SEPARATOR, or PARAGRAPH_SEPARATOR) but is not also a non-breaking space ('\u00A0', '\u2007', '\u202F').
It is '\t', U+0009 HORIZONTAL TABULATION.
It is '\n', U+000A LINE FEED.
It is '\u000B', U+000B VERTICAL TABULATION.
It is '\f', U+000C FORM FEED.
It is '\r', U+000D CARRIAGE RETURN.
It is '\u001C', U+001C FILE SEPARATOR.
It is '\u001D', U+001D GROUP SEPARATOR.

It is '\u001E', U+001E RECORD SEPARATOR.
It is '\u001F', U+001F UNIT SEPARATOR.

White space character in java includes space character along with above characters.

At Line n1, 'sb' refers to StringBuilder object {'E','L','E','C','T','R','O','T','H','E','R','M','A,'L'}. So length of this StringBuilder object is 14.
At Line n2, `sb.setLength(7);` sets the length of StringBuilder object referred by 'sb' to 7 (it is reducing the length of StringBuilder object from 14 to 7), hence 'sb' refers to {'E','L','E','C','T','R','O'}. Last 7 characters are gone.
There is no white space character in above StringBuilder object, hence at Line n3 `sb.toString()` returns "ELECTRO" and strip() method at Line n3 returns "ELECTRO".
"ELECTRO" is printed on to the console.
When code reaches Line n4, console shows "ELECTRO:"

At Line n5, `sb.setLength(14);` sets the length of StringBuilder object referred by 'sb' to 7 (it is increasing the length of StringBuilder object by 7 by filling available space with null characters '\u0000'). Now 'sb' refers to StringBuilder object
{'E','L','E','C','T','R','O','\u0000','\u0000','\u0000','\u0000','\u0000','\u0000,'\
At Line n6, `sb.toString()` returns corresponding String object "ELECTRO   ". As '\u0000' is white space character, hence "ELECTRO   ".strip() returns "ELECTRO". Line n6 prints "ELECTRO" on to the console.

Hence, the output is ELECTRO:ELECTRO.

4.1.15 Answer: A,C

Reason :
First find out the reason for compilation error, all the options are giving hint :)

no-argument constructor of Person class calling another overloaded constructor by the name and this causes compilation error. This problem can be fixed in 2 ways:
1st one: replace Person("Rohit", 25); with this("Rohit", 25) OR 2nd one: add void Person(String, int) method in the Person class.

Method can have same name as the class name and constructor can call other methods.

4.1.16 Answer: D

Reason :
Reference variable to which lambda expression is assigned is known as target type. Target type can be a static variable, instance variable, local variable, method parameter or return type.

Given lambda expression is the correct implementation of the Leveller interface.
Variable 'i' at Line n2 shadows the instance variable 'i' at Line n1.
Keyword 'this' within lambda expression refers to the instance of enclosing class where lambda expression is written, so this.i in lambda expression is 100.

As, 'level' is an instance variable of Test class, therefore to access it inside static main(String []) method, an instance of Test class is needed.
'new Test().level' correctly refers to the 'level' variable and `new Test().level.level()` at Line n4 executes the code of lambda expression and returns 100.

Line n4 prints 100 on to the console.

4.1.17 Answer: A

Reason :
Method m1() throws an instance of TestException, which is a checked exception as it extends Exception class.
So in throws clause we must provide:
1. Checked exception.
2. Exception of TestException type or it's super types (Exception, Throwable), Object cannot be used in throws clause.

Out of the given options only Exception can be filled in the blank space.

4.1.18 Answer: E

Reason :
`new StringBuilder(100);` creates a StringBuilder instance, whose internal char array's length is 100 but length() method of StringBuilder object returns the number of characters stored in the internal array and in this case it is 0. So, `sb.length()` returns 0.

sb.toString() is the String representation of StringBuilder instance and in this case as there are no characters inside the StringBuilder object, hence `sb.toString()` returns an empty String "", so `sb.toString().length()` also returns 0.
Output is 0:0.

4.1.19 Answer: B

Reason :
Given statement:
boolean res = x++ == 7 && ++x == 9 || x++ == 9;
boolean res = (x++) == 7 && ++x == 9 || (x++) == 9; // Postfix operator has higher precedence than other available operators
boolean res = (x++) == 7 && (++x) == 9 || (x++) == 9; //Then comes prefix operators
boolean res = ((x++) == 7) && ((++x) == 9) || ((x++) == 9); //== operator comes next
boolean res = (((x++) == 7) && ((++x) == 9)) || ((x++) == 9); //&& has higher precedence over ||
Right hand side is left with just one operator '||', it is a binary operator, hence let's solve the left hand side first.
boolean res = ((7 == 7) && ((++x) == 9)) || ((x++) == 9); //x = 8
boolean res = (true && ((++x) == 9)) || ((x++) == 9); //x = 8
boolean res = (true && (9 == 9)) || ((x++) == 9); //x = 9
boolean res = (true && true) || ((x++) == 9); //x = 9
boolean res = true || ((x++) == 9); //x = 9, || is a short-circuit operator, given expression evaluates to true without evaluating `((x++) == 9)`
boolean res = true; //x = 9
So,
x = 9
res = true

4.1.20 Answer: A

Reason :
`final boolean flag; flag = false;` doesn't make flag a compile time constant.
Compiler doesn't know flag's value at compile-time and hence it allows this syntax.
At runtime, as boolean expression of while loop is false, loop's body doesn't execute even once and hence no output.

4.1.21 Answer: D

Reason :
As per Java 8, default and static methods were added in the interface. Interface ILog defines static method log(), there is no compilation error in interface ILog.

Abstract class Log defines the static log() method. Abstract class can have 0 or more abstract methods. Hence, no compilation error in class Log as well.

Default methods of an interface are implicitly public and are inherited by the implementer class. Class MyLogger implements ILog interface and therefore it inherits the default log() method of ILog interface.
Also, the scope of static log() method of abstract class Log is not limited to class Log only but MyLogger also gets Log.log() method in its scope.
So, MyLogger class has instance method log() [inherited from ILog interface] and static method log() [from Log class] and this causes conflict. Static and non-static methods with same signature are not allowed in one scope, therefore class Log fails to compile.

4.1.22 Answer: B

Reason :
Initially text refers to "RISE ".
Given statement:
text = text + (text = "ABOVE ");
text = "RISE " + (text = "ABOVE "); //Left operand of + operator is evaluated first, text --> "RISE "
text = "RISE " + "ABOVE "; //Right operand of + operator is evaluated next, text --> "ABOVE "
text = "RISE ABOVE "; //text --> "RISE ABOVE "

Hence `System.out.println(text);` print 'RISE ABOVE ' on to the console.

4.1.23 Answer: A

Reason :
As member variables 'testNo' and 'desc' are declared with no explicit access specifier, this means these variables have package scope, hence these variables are accessible only to classes within the same package.
Hence, no changes are necessary.
If you use private, then instance variables will not be accessible to any

other classes, even within the same package.
If you use protected, then instance variables will be accessible to the subclasses outside 'com.udayankhattry.ocp1' package.
If you use public, then instance variables will be accessible to all the classes.

4.1.24 Answer: A,F

Reason :
This question is tricky as 2 changes are related and not independent. Let's first check the reason for compilation error. Line n2 throws a checked exception, IOException but it is not declared in the throws clause. So, print method should have throws clause for IOException or the classes in top hierarchy such as Exception or Throwable.

Based on this deduction, Line n1 can be replaced with either "static void print() throws Exception {" or "static void print() throws Throwable" but we will have to select one out of these as after replacing Line n1, Line n4 will start giving error as we are not handling the checked exception at Line n4.

This part is easy, do we have other options, which mention "Throwable"? NO. Then mark the first option as "Replace Line n1 with static void print() throws Exception {".

As, print() method throws Exception, so main method should handle Exception or its super type and not it's subtype. Two options working only with IOException can be ruled out.

Multi-catch statement "catch(IOException | Exception e)" causes compilation error as IOException and Exception are related to each other in multilevel inheritance. So you are left with only one option to pair with the 1st choice:
Surround Line n4 with below try-catch block:

```
try {
    ReadTheFile.print();
} catch(Exception e) {
    e.printStackTrace();
}
```

4.1.25 Answer: D

Reason :

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

At Line n1, list refers to List instance of String [] type, which means variable 'list' refers to List instance containing two elements of String[] type. Line n1 compiles successfully .

Iterable<T> interface has forEach(Consumer) method. List<E> extends Collection<E> & Collection<E> extends Iterable<E>, therefore forEach(Consumer) can easily be invoked on reference variable of List<E> type.
As Consumer is a Functional Interface, hence a lambda expression can be passed as argument to forEach() method.
forEach(Consumer) method performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

The lambda expression at Line n2 is the correct implementation of Consumer<String []> interface. It compiles successfully and on execution prints the length of each array element.
1st array object has 3 elements and 2nd array object has 2 elements, therefore output is: 32

4.1.26 Answer: D

Reason :
Inside for-each loop, System.out.println(arr[i]); is used instead of System.out.println(i);
When loop executes 1st time, i stores the first array element, which is 3 but System.out.println statement prints arr[3] and this causes java runtime to throw the instance of ArrayIndexOutOfBoundsException.

4.1.27 Answer: B,C,D

Reason :
Logic in for loop is adding array elements. You need to find out which array elements when added will result in 9. Possible options are: {1+3+5, 2+3+4, 4+5}.

Based on these 3 combinations you can select 3 correct options.

4.1.28 Answer: B,C,E,F,H

Reason :

At Line n1, reference variable 'obj' is of Multiplier type (supertype) and it refers to an instance of Calculator class (subtype). This is polymorphism and allowed in Java.
multiply(int...) method declared in Multiplier interface declares to throw SQLException, hence the catch handler for Line n1 should provide handler for SQLException or its supertype. As catch-handler for SQLException is available, therefore Test class compiles successfully.

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:
1. May not declare to throw any checked exceptio n
2. May declare to throw the same checked exception thrown by super class / interface method: SQLException is a valid option.
3. May declare to throw the sub class of the exception thrown by super class / interface method: SQLWarning is a valid option.
4. Cannot declare to throw the super class of the exception thrown by super class / interface method: Exception, Throwable are not valid options.
5. Cannot declare to throw unrelated checked exception: java.io.IOException is not a valid option as it is not related java.sql.SQLException in multi-level inheritance.
6. May declare to throw any RuntimeException or Error: RuntimeException, NullPointerException and Error are valid options.

Therefore 5 options can successfully replace /*INSERT*/: SQLException, SQLWarning, RuntimeException, Error and NullPointerException

4.1.29 Answer: C,E

Reason :

BasicCalculator class (in basic.calc module) uses AdvanceCalculator (in advance.calc module), which means 'basic.calc' requires 'advance.calc' module.
Hence, /*INSERT-1*/ should be replaced with `requires advance.calc;`

Also, check the import statement of BasicCalculator: import com.udayankhattry.ocp1.calc.advance.AdvanceCalculator;
AdvanceCalculator is defined in 'com.udayankhattry.ocp1.calc.advance' package, so module descriptor of 'advance.calc' module must export this package.
Hence, /*INSERT-2*/ should be replaced with `exports com.udayankhattry.ocp1.calc.advance;`
or qualified exports can also be used `exports com.udayankhattry.ocp1.calc.advance to basic.calc;`

So, only two options are correct:
Replace /*INSERT-1*/ with requires advance.calc;
and
Replace /*INSERT-2*/ with exports com.udayankhattry.ocp1.calc.advance;

&
Replace /*INSERT-1*/ with requires advance.calc;
and
Replace /*INSERT-2*/ with exports com.udayankhattry.ocp1.calc.advance to basic.calc;

4.1.30 Answer: C

Reason :
It is good practice to have first character of class name in upper case, but it is not mandatory.
student can be either class name or reference variable name.

Syntax to invoke static method is:
Class_Name.method_name(<arguments>); OR
reference_variable_name.method_name(<arguments>);
Syntax to invoke instance method is:
reference_variable_name.method_name(<arguments>);
If student represents class_name or refernce_variable_name, then report might be the static method of the class.
If student represents reference_variable_name, then report is the instance method of the class.
In both the cases, report must be the method name.

Type of argument cannot be found out by looking at above syntax.

4.1.31 Answer: C

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

Given code compiles successfully.
At Line n1, variable 'res' infers to String.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO "

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Let's check the iterations:
1st iteration: s refers to "Dog". `String.join(".", s)` returns "Dog" and res = "" + "Dog" = "Dog".
2nd iteration: s refers to null. `String.join(".", s)` returns "null" and res = "Dog" + "null" = "Dognull".
3rd iteration: s refers to "Friendly". `String.join(".", s)` returns "Friendly" and res = "Dognull" + "Friendly" = "DognullFriendly".
Loop finishes its execution and Line n4 prints DognullFriendly on to the console.

4.1.32 Answer: D

Reason :
Class M and M are declared public and inside same package com.udayankhattry.ocp1. Method printName() of class M has correctly been overridden by N.
printName() method is public so no issues in accessing it anywhere.

Let's check the code inside main method.
M obj1 = new M(); => obj1 refers to an instance of class M .
N obj2 = (N)obj1; => obj1 is of type M and it is assigned to obj2 (N type), hence explicit casting is necessary. obj1 refers to an instance of class M, so at runtime obj2 will also refer to an instance of class M. sub type can't refer to an instance of super type so at runtime `N obj2 = (N)obj1;` will throw ClassCastException.

4.1.33 Answer: B

Reason :
Though this question looks complex but it is very simple. You are expected to know the location of module-info.java file.

module-info.java file is placed under the module directory, which in this case is 'office', hence module-info.java file must be placed at [POSITION-2].

If module-info.java file is missing or is placed at wrong position, then given javac command would cause below error:

```
error: module office not found in module source path
```

4.1.34 Answer: E

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
     (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Animal.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java Animal.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Animal.java
java -cp <memory> Cat
```

Please note that source-file can contain multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

First class defined in Animal.java file is Cat and as it doesn't contain special main method (parameter is of String type and not String [] type), hence given command causes error.

```
F:\>java Animal.jav a
error: can't find main(String[]) method in class: Cat
```

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

4.1.35 Answer: A

Reason :
name, height, result and age are instance variables of Student class. And instance variables are initialized to their respective default values.
For the Student object created at Line n1, 'name' is initialized to null, 'age' to 0, 'result' to false and 'height' to 0.0.

Hence, Line n2 prints null0.0false0

4.1.36 Answer: A

Reason :
'java.se' is the name of the module, hence it is a named module and not unnamed.

In modular JDK, module names starting with "java." are known as standard modules. Non-standard module names don't start with "java.", such as module names starting with "jdk." are non-standard.
'java.se' is standard module and it is not deprecated.

An aggregator module collects and re-exports the content of other modules but adds no content of its own.

If you describe the java.se module:

```
C:\>java --describe-module java.se
java.se@11.0.3
requires java.xml.crypto transitive
requires java.datatransfer transitive
requires java.naming transitive
requires java.management transitive
requires java.instrument transitive
requires java.sql transitive
requires java.xml transitive
requires java.security.sasl transitive
requires java.base mandated
requires java.prefs transitive
requires java.security.jgss transitiv e
requires java.transaction.xa transitive
requires java.desktop transitive
requires java.sql.rowset transitive
requires java.management.rmi transitive
requires java.rmi transitive
requires java.net.http transitive
requires java.compiler transitive
requires java.logging transitive
requires java.scripting transitive
```

It is just re-exporting other java standard modules, hence 'java.se' is an aggregator module.

4.1.37 Answer: G

Reason :
Variable 'i' used in lambda expression clashes with another local variable 'i' and hence causes compilation error.

4.1.38 Answer: C

Reason :
It is legal for the constructors to have throws clause.
Constructors are not inherited by the Derived class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Derived class's constructor:

```
Derived() throws FileNotFoundException {
    super(); //added by the compiler
    System.out.print(2);
}
```

As super(); invokes the constructor of Base class (which declares to throw IOException), compiler complains as Derived class no-argument constructor doesn't declare to throw IOException. It declares to throw FileNotFoundException (subclass of IOException), which is not enough for the instances of IOException.

4.1.39 Answer: C

Reason :

Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class]. Operator<T> is a functional interface as it has exactly one non-overriding abstract method. It is also a generic interface, hence it can work with any java class. There are absolutely no issues with lambda expressions but we are not capturing or printing the return value of operation method, hence nothing is printed on to the console.

To print the return values you can use below statements:
System.out.println(opr1.operation("Think", "First")); => Prints ThinkFirst.
System.out.println(opr2.operation(10, 40)); => Prints 50. Over here int literals 10 and 40 are converted to Integer instances by auto-boxing.

4.1.40 Answer: E

Reason :
Line n1 creates an int array object of 3 elements: 10, 100, 1000 and arr1 refers to it.
Line n2 creates an char array object of 3 elements: 'x', 'y', 'z'.
Statement `arr1 = arr2;` causes compilation error as char [] is not compatible with int [] even though char is compatible with int.

4.1.41 Answer: A

Reason :
In this example, code is trying to remove an item from the list while iterating using traditional for loop so one can think that this code would throw java.util.ConcurrentModificationException.

But note, java.util.ConcurrentModificationException will never be thrown for traditional for loop. It is thrown for for-each loop or while using Iterator/ListIterator.

In this case `animals.remove(new StringBuilder("Alligator"));` will never remove any items from the list as StringBuilder class doesn't override the equals(Object) method of Object class.

StringBuilder instances created at "animals.add(new StringBuilder("Alligator"));" and "animals.remove(new StringBuilder("Alligator"));" are at different memory locations and equals(Object) method returns false for these instances.

4.1.42 Answer: D

Reason :
list.add(2); => 2 is auto-boxed to Integer and added to the list.
list.add(1); => 1 is auto-boxed to Integer and added to the list.
list.add(0); => 0 is auto-boxed to Integer and added to the list.

remove method of List interface is overloaded: remove(int) and remove(Object).
indexOf method accepts argument of Object type, in this case list.indexOf(0) => 0 is auto-boxed to Integer object so no issues with indexOf code. list.indexOf(0) returns 2 (index at which 0 is stored in the list). So list.remove(list.indexOf(0)); is converted to list.remove(2);

remove(int) version is matched, it's a direct match so compiler doesn't do auto-boxing in this case. list.remove(2) removes the element at index 2, which is 0.

Hence in the output, you get [2, 1].

4.1.43 Answer: A

Reason :
Use -d option with javac command. -d option is used to specify where to place generated class files (which is under the 'F:\codes\classes' directory).
As you are typing javac command from within 'F:', hence path of 'classes' directory and Utils.java file relative to 'F:' can be given.
So, correct command is: javac -d codes\classes\ codes\src\Utils.java

Please note file name is 'Utils.java' but not 'com.udayankhattry.ocp1.Utils.java'.
Also note that it is recommended practice to put the source files under the package directory (com\udayankhattry\ocp1\) and all the major IDEs (Eclipse, IntelliJ IDEA etc.) follow this practice but it isn't a compulsion.

4.1.44 Answer: H

Reason :
Let's check all the options one by one:

```
module com.discount.shopping {
}
```

✗ By default module doesn't export any package.

```
module com.discount.shopping {
    exports com.discount.shopping.Product;
    exports com.discount.shopping.Order ;
    exports com.discount.shopping.payments.Discount;
}
```

✗ after exports keyword package name is expected and not the class name.

```
module com.discount.shopping {
    exports com.discount.shopping.*;
    exports com.discount.shopping.payments.*;
}
```

✗ Wild-card (*) is not allowed with the package names.

```
module com.discount.shopping {
    exports com.discount.shopping.*;
}
```

✗ Wild-card (*) is not allowed with the package names.

```
module shopping {
    exports com.discount.shopping;
    exports com.discount.shopping.payments;
}
```

✗ Even though both the exports are correct but module name is 'shopping' and not 'com.discount.shopping'.

```
module com.discount.shopping {
    exports com.discount.shopping to all;
    exports com.discount.shopping.payments to all;
}
```

✗ It is a valid syntax of qualified exports but 'all' has no special meaning here.
Syntax of qualified exports is:

```
exports <package_name> to <comma_separated_list_of_modules>;
```

Given exports directives will just make packages 'com.discount.shopping' and 'com.discount.shopping.payments' readable by module named 'all' and not by all the modules.
If module 'all' is not available, then compiler compiles all the files but

displays a warning for missing module 'all'.

```
module com.discount.shopping {
    export com.discount.shopping;
    export com.discount.shopping.payments;
}
```

✗ 'export' is not a valid directive, it should be 'exports'.

Hence, correct answer is: None of the other options

4.1.45 Answer: D

Reason :
`System.out.println(1/0);` throws ArithmeticException, handler is available in inner catch-block, it executes and prints "INNER" to the console.

Once an exception is handled, no other catch block will get executed unless the exception is re-thrown.

Inner finally-block gets executed and prints "FINALLY 1" to the console.

Rule is finally-block always gets executed, so outer finally-block gets executed and prints "FINALLY 2" to the console.

4.1.46 Answer: B

Reason :
For readability purpose underscore (_) is used to separate numeric values. This is very useful in representing big numbers such as credit card numbers (1234_7654_9876_0987). long data can be suffixed by l, float by f and double by d. So first 4 variable declaration and assignment statements don't cause any compilation error.

long can be easily assigned to float and double type variables, hence no issues with c = a; and d = a;
double cannot be assigned to float without explicit casting, so c = d; causes compilation error.

float can easily be assigned to double and int can easily be assigned to float, so d = c; and c = b; compile successfully.
float can't be assigned to int without explicit casting, so b = c; causes compilation error.

double can't be assigned to int without explicit casting, statement b = (int)d; is casting double to int, so no issues.

In total, two statements are causing compilation error: c = d; and b = c;

4.1.47 Answer: D

Reason :
list.add(0, "Array"); means list --> [Array],
list.add(0, "List"); means insert "List" to 0th index and shift "Array" to right. So after this operation, list --> [List, Array]. In the console, [List, Array] is printed.

4.1.48 Answer: C

Reason :
Line n1 creates an array instance of ILogger containing 2 elements. null is assigned to both the array elements. Line n1 compiles successfully.

As, log() method is declared in ILogger interface, hence statement at Line n2: logger.log(); doesn't cause any compilation error. Compiler is happy to see that log() method is invoked on the reference variable of ILogger type.

1st iteration:
logger --> null, logger.log(); throws NullPointerException as method log() is invoked on null reference.

4.1.49 Answer: G

Reason :
At Line n1, an int [] object of three elements is created and 'arr' refers to this array object.
arr[0] = 10, arr[1] = 20 and arr[2] = 30;

Given expression at Line n2:
arr[i++] = arr[++i] = 40;
Multiple assignment operators are available, so lets group it first.
=> arr[i++] = (arr[++i] = 40); //Assignment operator is right to left associative
Above expression is valid, hence Line n2 compiles successfully.
Let's solve the expression now. Left operand is 'arr[i++]' and right operand is '(arr[++i] = 40)'. Left operand is evaluated first.
=> arr[0] = (arr[++i] = 40); //i = 1

Right hand operand is evaluated next.
=> arr[0] = (arr[2] = 40); //i = 2
=> arr[0] = 40; //i = 2, arr[2] = 40.
Hence after Line n2, arr refers to int [] object {40, 20, 40}.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name .

For Line n3, variable 'x' infers to int type.

Given loop prints below on to the console:
40
20
40

4.1.50 Answer: F

Reason :
Variable 'count' declared inside interface Counter is implicitly public, static and final. Line n1 compiles successfully.
Line n2 creates one dimensional array of 2 elements of Counter type and both the elements are initialized to null. Line n2 compiles successfully.
Though correct way to refer static variable is by using the type name, such as Counter.count but it can also be invoked by using Counter reference variable. Hence ctr.count at Line n3 correctly points to the count variable at Line n1.
For invoking static fields, object is not needed, therefore even if 'ctr' refers to null, ctr.count doesn't throw NullPoiointerException. Given loop executes twice and therefore output is: 1010

4.1.51 Answer: B

Reason :
Generic type can only be reference type and not primitive type, hence java.util.List<int> is not a valid syntax.

If you use raw type java.util.List or java.util.List<Object> then Line n3 will cause compilation failure as list.get(0) will return Object type. Object type cannot be converted to primitive type int.

java.util.List<Integer> is the only correct option left.

4.1.52 Answer: C

Reason :
Given boolean expression:
(num++ == num++) //num=10
(10 == num++) //Left side operand is evaluated first, value 10 is used in the expression and variable num is incremented by 1, so num=11
(10 == 11) //Right side operand is evaluated next, value 11 is used in the expression and variable num is incremented by 1, so num = 1 2
Above expression evaluates to false, hence else block is executed and NOT EQUAL 12 is printed on to the console.

4.1.53 Answer: B,G

Reason :
Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".

In this question, 'str' refers to "*". Let's check all the options:
`str.repeat(5);`: returns a new String Object "*****" but as String is immutable, so there is no change is the String object referred by 'str'. It will not give expected output.
`str = str.repeat(5);`: returns a new String Object "*****" and 'str' refers to this newly created object. It will print the expected output.
`String.repeat(str, 5);`: repeat is an instance method and not static. It causes compilation error.
`str = String.repeat(str, 5);`: repeat is an instance method and not static. It causes compilation error.
`str *= 5;`: Operator '*=' is not defined for String type and hence it causes compilation error.
`str = str * 5;`: Operator '*' is not defined for String type and hence it causes compilation error.
`for(var var = 0; var < 4; var++) { str += "*"; }`:

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int
The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.
It is a valid loop syntax in which var infers to int type, loop executes 4 times and at the end 'str' refers to "*****". It will also print expected output on to the console.

for(int i = 0; i < 5; i++) { str += "*"; }: Loop syntax is correct but this loop executes 5 times and at the end 'str' refers to "******". It will not print expected output.

4.1.54 Answer: B

Reason :
ClassCastException extends RuntimeException (unchecked exception),
FileNotFoundException extends IOException, IOException extends Exception (checked exception),
ExceptionInInitializerError is from Error family and is thrown by an static initializer block,
RuntimeException and all its sub classes are unchecked exceptions.

4.1.55 Answer: B

Reason :
Please note that Strings computed by concatenation at compile time are referred by String Pool. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc. This means the result of constant expression is calculated at compile time and later referred by String Pool.
For the given question,
's1' refers to "1Z0-815" (String Pool object).
"1Z0-815" + "" = "1Z0-815". As "1Z0-815" is already available in the String Pool, hence same object is used.
On execution, 's1' and 's2' both refer to the same String object and that is why s1 == s2 returns true.

Please note that Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.
For below code snippet:

```
String str1 = "1Z0-815";
String str2 = str1 + "";
System.out.println(str1 == str2);
```

Output is false, as 'str1' is a variable and `str1 + ""` is not a constant expression, therefore a new non-pool String object is created.

4.1.56 Answer: B

Reason :
ArrayList can have duplicate elements, so after addition, list is: [SUNDAY, SUNDAY, MONDAY]. days.size() returns 3, so 3 is printed on to the console.
days.clear(); removes all the elements from the days list, in fact days list will be empty after successful execution of days.clear();
2nd System.out.println statement prints 0 on to the console.

4.1.57 Answer: D

Reason :
In the boolean expression (predicate.test(i)): i is of primitive int type but auto-boxing feature converts it to Integer wrapper type.
test(Integer) method of Predicate returns true if passed number is an odd number, so given loop prints only odd numbers. for loops works for the numbers from 1 to 10.

4.1.58 Answer: B

Reason :
Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this case, module source path is 'source' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file

specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'classes' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
In this case, it is 'jars\dbconnection.jar'.

It is clear that module 'dbtester' requires the module 'dbconnection' to access the 'com.database.util.DBConnect' class. Jar file of module 'dbconnection' is available under 'jars' directory.
To access compiled module code/jar files of other modules at compile time use --module-path or -p option.
Hence, correct command to compile 'dbtester' module is :
javac -d classes --module-source-path source -p jars\dbconnection.jar -m dbtester

Let's check other 3 javac commands:
javac -d classes --module-source-path source;jars\dbconnection.jar -m dbtester ✗ 'dbconnection.jar' doesn't contain module source, hence compiler complains about missing module.

javac -d classes --module-source-path source,jars\dbconnection.jar -m dbtester ✗ Path separator is ; (on windows) or : (on Linux/Mac), hence comma should not be used. In this case, module source path is considered as source,jars\dbconnection.jar'.

javac -d classes --module-source-path src --module-class-path jars\dbconnection.jar -m dbtester ✗ --module-class-path is not a valid option, correct option is: --module-path.

4.1.59 Answer: D

Reason :
Each instance of the class contains separate copies of instance variable and share one copy of static variable.
There are 3 instances of Counter class created by main method and these are referred by ctr1, ctr2 and ctr3.
As 'ctr' is a static variable of Counter class, hence ctr1.ctr, ctr2.ctr and ctr3.ctr refer to the same variable. In fact, 'Counter.ctr' is the preferred way to refer the static variable 'ctr' but ctr1.ctr, ctr2.ctr and ctr3.ctr are also allowed.

As 'count' is an instance variable, so there are 3 separate copies: ctr1.count, ctr2.count, ctr3.count.

On the completion of for loop: ctr1.count = 5, ctr2.count = 5 and ctr3.count = 5 and Counter.ctr = 15.

15:5 is printed on to the console.

4.1.60 Answer: D

Reason :
Before you answer this, you must know that there are 5 different Student object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Student objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method defined in Object class uses == operator to check the equality and in this case as 5 Student objects are stored at different memory location, hence not equal.

Nothing is removed from the students list, all the 4 Student objects are printed in the insertion order.

4.1.61 Answer: B

Reason :
Greetings class contains overloaded constructors: a no-argument constructor Greetings() and a parameterized constructor Greetings(String).
Greetings g1 = new Greetings(); invokes no-argument constructor.
No-argument constructor calls parameterized constructor with the

argument "Good Morning!"
Parameterized constructor assigns "Good Morning!" to 'msg' variable of the object referred by g1.
Greetings g2 = new Greetings("Good Evening!"); invokes parameterized constructor, which assigns "Good Evening!" to 'msg' variable of the object referred by g2.
g1.displayMsg(); prints Good Morning!
g2.displayMsg(); prints Good Evening!

4.1.62 Answer: D

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name, package name or loop's label but it cannot be used as a class or interface name.

For the 1st loop variable 'i' infers to int type, so no issues for 1st loop and for the 2nd loop variable 'j' infers to int type, so no issues for 2nd loop as well.
Let's check the iteration:
1st iteration of loop one: i = 0
    1st iteration of loop two: j = 0
        1st iteration of loop three: ctr = 101. As `i == j` evaluates to true, hence `break two;` gets executed, which takes the control out of loop two and hence to the increment expression (i++) of loop one.
2nd iteration of loop one; i = 1
    1st iteration of loop two: j = 0
        1st iteration of loop three; ctr = 102. As `i > j` evaluates to true, hence `break one;` gets executed, which takes the control out of the loop one.

`System.out.println(ctr);` prints 102 on to the console.

4.1.63 Answer: C

Reason :
Style class has an instance variable 'pattern', which is initialized to "*" for all the instances.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

At Line n1, variable 'style' infers to Style type. An instance of Style class is created, pattern instance variable is initialized to "*" and variable 'style' refers to this instance.

Instance method 'repeat()' was added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".

Expression at Line n2: style.pattern.repeat(5).length()
=> "*".repeat(5).length() //style.pattern refers to String object "*".
=> "*****".length() //"*".repeat(5) returns "*****".
=> 5 //"*****".length() returns 5.

Line n2 prints 5 on to the console.

4.1.64 Answer: B

Reason :
Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].
Printable is a Functional Interface.

Given code compiles successfully and on execution prints AIM HIGH on to the console.

4.1.65 Answer: B

Reason :

NOTE: System.out.print statement is printing arr[i][1], which means it prints 2nd array element of a particular row, for each iteration of inner loop.

That is why output is:

```
BBB
EEE
HHH
```

To get all the array elements printed correctly, use arr[i][j] in System.out.print statement.

4.1.66 Answer: A

Reason :

Even though default block is available at the top but matching case is present. So control goes inside matching case and prints "x is equal to 2" on to the console. After that break; statement takes the control out of the switch-case block.

4.1.67 Answer: D,E

Reason :

Options of the jar command:

-c or --create: Create the archive

-f or --file=FILE: The archive file name

-e or --main-class=CLASSNAME: The application entry point for stand-alone applications bundled into a modular, or executable, jar archive

-C DIR: Change to the specified directory and include the following file

Given jar command:

```
C:\>jar -cfe jars\messages.jar com.udayankhattry.ocp1.Test -C bin\messages .
```

Jar command instructs to create a new archive 'jars\messages.jar' and set the entry point to 'com.udayankhattry.ocp1.Test' class .

-C bin\messages . instructs jar tool to change to the 'bin\messages' directory and put all compiled files from this directory in the JAR file.

Important point to note here is that entry point (mainclass) is: com.udayankhattry.ocp1.Test

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
(to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

Let's check all the options one by one:
java -p jars\messages.jar -m messages: ✗ As <mainclass> has not been specified, hence it considers the mainclass set at jar creation, which is: com.udayankhattry.ocp1.Test. Output is: DRILL YOUR SKILLS

java -p jars -m messages: ✗ With -p option, you can specify directory containing modules or individual modules as well. Same as above, it also prints DRILL YOUR SKILLS on to the console

java -p bin -m messages: ✗ As directory 'bin' contains exploded module, hence class name must be specified after module name 'messages'

java -p bin -m messages/com.udayankhattry.ocp1.Main: ✓ As directory 'bin' contains exploded module and class name is also specified after module name 'messages', main(String...) method of Main class gets executed and prints FOCUS AND WIN on to the console.

java -p jars -m messages/com.udayankhattry.ocp1.Main: ✓ Specifying class name after the module name (in case of jar file) executes the given class and not the entry-point (mainclass) set at jar creation. main(String...) method of Main class gets executed and prints FOCUS

AND WIN on to the console.

java -m messages/com.udayankhattry.ocp1.Main -p jars: ✗ -m must be the last option.

java -m messages/com.udayankhattry.ocp1.Main: ✗ --module-path or -p option is missing.

4.1.68 Answer: A

Reason :
CommunicationLog class overrides count() and get() methods of Log class.
There are 2 rules related to return types:
1. If return type of overridden method is of primitive type, then overriding method should use same primitive type.
2. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

count() method at Line n1 returns long but overriding method at Line n3 returns int and that is why Line n3 causes compilation error.
get() method at Line n2 returns Object but overriding method at Line n4 returns String. String is a subclass of Object, so it is a case of covariant return type and hence allowed. Line n4 compiles successfully.

4.1.69 Answer: E

Reason :
`int indexOf(String str, int fromIndex)` method of String class returns the index within this string of the first occurrence of the specified substring, starting at the specified index. e.g.
"alaska".indexOf("a", 1) returns 2
"alaska".indexOf("a", 2) returns 2
"alaska".indexOf("a", 3) returns 5

In the given question, 'arr' refers to a String array of size 5. Element at index 0 refers to "1st", element at index 1 refers to "2nd" and so on.

Let's solve the given expression of Line n1:
arr[place.indexOf("a", 3)]
= arr["faraway".indexOf("a", 3)] //Starts looking for "a" from index 3 of the given String "faraway" and "a" is found at index 3.

= arr[3]
= "4th" //Array element at index 3 refers to "4th" .

Hence, 4th is printed on to the console.

4.1.70 Answer: D

Reason :
Command to compile a java file: javac <java_file_name>.java [.java extension is compulsory],
Command to execute a java class: java <class_file_name> [.class extension should not be used]

4.1.71 Answer: H

Reason :
Method can have same name as that of the Class. Hence, void Base() is a valid method declaration in Base class.

Line n2 invokes the Base() method and not the constructor.

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n5 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

Line n7 correctly overrides the Base() method of class Base.

Compiler adds super(); as the 1st statement inside the no-argument constructor of Base class and Derived class.

There is no compilation error, so let's check the execution.

new Derived() at Line n9 invokes the constructor of Base class, at this point instance variable id is declared and 0 is assigned to it. In fact, instance variable id of Base class is also declared and 0 is assigned to it. Compiler added super(); as the first statement inside this constructor, hence control goes to the no-argument constructor of Base class.

Compiler added super(); as the first statement inside this constructor as well, hence it invokes the no-argument constructor of the Object class. No-argument constructor of Object class finishes its execution and control goes back to the constructor of Base class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 1000 is assigned to variable id of Base class .

Line n2 is executed next, Base() method defined in Derived class is executed. Which overriding method to invoke, is decided at runtime based on the instance. Instance is of Derived class (because of Line n9), hence control starts executing Base() method of Derived class.
Line n8 is executed next, Derived class hides the id variable of Base class and that is why at Line n8, id points to variable created at Line n5. This id variable still stores the value 0 as Base class's constructor has not finishes its execution.

value of id is decremented by 1, so id becomes -1 and -1 is printed on to the console. Base() method finishes its execution and control goes back to Line n2. No-argument constructor of Base class finishes its execution and control goes back to the constructor of Derived class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 2000 is assigned to variable id of Base class.

No-argument constructor of Derived class finishes its execution and control goes back to Line n9. main(String []) method finishes its execution and program terminates successfully.

Hence, output is -1.

4.1.72 Answer: D

Reason :
'color' is LOCAL variable and it must be initialized before it can be used.
As area is not compile time constant, java compiler doesn't have an idea of the value of variable area.
There is no else block available as well.

So compiler cannot be sure of whether variable color will be initialized or not. So `System.out.println(color);` causes compilation error.

4.1.73 Answer: G

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable

declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

For Line n1, x infers to String type.
For Line n2, y infers to int type.
As, x is of String type and y is of int type, hence x + y results in "100100".

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

4.1.74 Answer: C

Reason :
'furniture' refers to String object "Sofa". Matching case is available, SOFA is printed on to the console. No break statement inside 'case "Sofa":', hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends. So in this case, it prints BED and after that break; statement takes control out of switch block. main method ends and program terminates successfully.

4.1.75 Answer: C

Reason :
Class declared as final cannot be inherited. Examples are: String, Integer, System etc.

4.1.76 Answer: A

Reason :
If you don't initialize variable e inside catch block using `e = new SQLException();` and simply throw e, then code would compile successfully as compiler is certain that 'e' would refer to an instance of SQLException only.

But the moment compiler finds `e = new SQLException();`, `throw e;` causes compilation error as at runtime 'e' may refer to any Exception type.

4.1.77 Answer: A

Reason :

According to the link: https://openjdk.java.net/jeps/200
Motivation behind Java modularity is:
Project Jigsaw aims to design and implement a standard module system for the Java SE Platform and to apply that system to the Platform itself, and to the JDK. Its primary goals are to make implementations of the Platform more easily scalable down to small devices, improve security and maintainability, enable improved application performance, and provide developers with better tools for programming in the large .

You can create a runtime image of your application containing only the necessary modules to run an application. No need to load everything. This reduces the size of the image and thus increases the performance.
Also modular JDK encapsulates the internal types (even public) and less number of classes at runtime decreases the attack surface, which automatically increases security.

4.1.78 Answer: C,D,G

Reason :

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

At Line n1, m infers to int type as 10 is an int literal.
At Line n2, n infers to int type as 20 is an int literal.

var type can easily replace /*INSERT*/ and in this case p would infer to int type.
As m and n are non-final variables, hence /*INSERT*/ cannot be replaced with byte or short because int variable cannot be implicitly casted to byte or short types.
int, long, float and double types can also replace /*INSERT*/ without causing any error but for float and double type output will be 90.0 where as for int, long and var type output will be 90.

Hence, only 3 options (int, long and var) print expected output on to the console.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

4.1.79 Answer: E

Reason :
Let's check all the options one by one:
void test(); ✓ By default test method is public and abstract and it is allowed inside an interface.

```
default void test(String name) {
    System.out.println("Testing " + name);
} ✓
```

As per Java 8, default methods were added in the interface. It is a valid syntax for the default method to be used inside an interface.

```
static void test(int x) {
    System.out.println(x);
} ✓
```

As per Java 8, static methods were added in the interface. It is a valid syntax for the static method to be used inside an interface.

Even if all the above 3 methods [test(), test(String) and test(int)] are available in the same interface, there is no issue at all, it is the case of method overloading.

private default void log1() {} ✗ As per Java 9, private methods were added in the interface, these can be static or non-static but not default.

private void log2() {} ✓ As per Java 9, private methods were added in the interface, these can be static or non-static.

private static void log3() {} ✓ As per Java 9, private methods were added in the interface, these can be static or non-static.

Out of 6 declarations/definitions, 5 can be used inside an interface.

4.1.80 Answer: B

Reason :
Given statement:
System.out.println(text.concat(text.concat("ELEVEN ")).trim()); //'text'

refers to "ONE "

System.out.println(text.concat("ONE ELEVEN ").trim()); //As String is immutable, hence there is no change in the String object referred by 'text', 'text' still refers to "ONE "

System.out.println(("ONE ONE ELEVEN ").trim()); //'text' still refers to "ONE "

System.out.println("ONE ONE ELEVEN"); //trim() method removes the trailing space in this case

ONE ONE ELEVEN is printed on to the console.

5 Practice Test-5

5.1 80 Questions covering all topics.

5.1.1 Given code of Sport.java file:

```
package com.udayankhattry.ocp1;

public class Sport {
    String name = "" ;

}
```

And below definitions:

1.

```
public String Sport() {
     return name ;
}
```

2.

```
public void Sport(String name) {
    this . name = name;
}
```

3.

```
public void Sport(String... sports) {
    name = String. join ( "," , sports);
}
```

4.

```
public Sport() {
     name = "" ;
}
```

5.

```
public Sport(String name) {
    this . name = name;
}
```

6.

```
public Sport(String name1, String name2) {
    name = String. join ( "," , name1, name2);
}
```

How many of the given definitions can be added to Sport class, such that there are no compilation errors ?

A. All 6 definitions
B. Only 5 definitions
C. Only 4 definitions
D. Only 3 definitions
E. Only 2 definitions
F. Only 1 definition

5.1.2 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        int a = 1000 ;
       System. out .println(-a++);
    }
}
```

A. Compilation error
B. -1000
C. -1001
D. 999
E. -999

5.1.3 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        System. out .println( "Equals??? " + 10 != 5 );
    }
}
```

What will be the result of compiling and executing Test class ?

A. Equals??? true

B. Equals??? false

C. Compilation error

D. Equals??? 10 != 5

5.1.4 For the class Test, which options, if used to replace /*INSERT*/, will print TEN on to the console? Select ALL that apply.

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        /*INSERT*/
        switch (var) {
            case 10 :
                System. out .println( "TEN" );
                 break ;
            default :
                System. out .println( "DEFAULT" );
        }
    }
}
```

A. byte var = 10;

B. long var = 10;

C. Short var = 10;

D. Integer var = 10;

E. char var = 10;

F. double var = 10;

5.1.5 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   +---string.util
|   |   |   module-info.java
|   |   |
|   |   \---com
|   |       \---udayankhattry
|   |           +---utility
|   |           |       StringUtility.java
|   |           |
|   |           \---ocp1
|   |               \---util
|   |                       StringUtil.java
|   |
|   \---testing
|       |   module-info.java
|       |
|       \---com
|           \---udayankhattry
|               \---test
|                       Test.java
|
\---out
```

Below are the file details:-

C:\src\string.util\module-info.java:

```
module string.util {
    exports com.udayankhattry.ocp1.util;
}
```

C:\src\string.util\com\udayankhattry\ocp1\util\StringUtil.java:

```
package com.udayankhattry.ocp1.util;

import com.udayankhattry.utility.StringUtility;

public class StringUtil {
    public static String reverse(String str) {
        return StringUtility.reverse(str);
    }
}
```

C:\src\string.util\com\udayankhattry\utility\StringUtility.java:

```
package com.udayankhattry.utility;

public class StringUtility {
    public static String reverse(String str) {
        return new StringBuilder(str).reverse().toString();
    }
}
```

C:\src\testing\module-info.java:

```
module testing {
    requires string.util;
}
```

C:\src\testing\com\udayankhattry\test\Test.java:

```
package com.udayankhattry.test;

import com.udayankhattry.ocp1.util.StringUtil;
import com.udayankhattry.utility.StringUtility;

public class Test {
    public static void main(String[] args) {
        System. out .println(StringUtil.reverse( "ROTATOR" ));
        System. out .println(StringUtility.reverse( "NOON" ));
    }
}
```

And the command executed from C:\

```
javac -d out --module-source-path src -m testing
```

Which of the following statements is correct?

A. Given command compiles successfully

B. Given command causes compilation error because of the code in Test.java file

C. Given command causes compilation error because of the code in StringUtility.java file

D. Given command causes compilation error because of the code in StringUtil.java file

5.1.6 Given below the directory/file structure on Windows platform:

```
C:
+---out
\---src
    \---medicines
        |  module-info.java
        |
        \---com
            \---allopathic
                    Medicines.java
```

Below are the codes of Java files:-

C:\src\medicines\module-info.java:

```
module medicines {
    exports com.allopathic to elizabeth_hospital;
}
```

Please note that module 'elizabeth_hospital' is not available.

C:\src\medicines\com\allopathic\Medicines.java:

```
package com.allopathic;

public class Medicines {
     public static java.util.List<String> getAll() {
         return null ;
    }
}
```

And the command executed from C:\

```
javac -d out --module-source-path src -m medicines
```

What is the result?

A. Compilation error for missing module elizabeth_hospital

B. Code compiles but with a warning for missing module elizabeth_hospital

C. Code compiles without any warning

D. Compilation error in Medicines.java file

5.1.7 Consider below code of TestBaseDerived.java file:

```
//TestBaseDerived.java
package com.udayankhattry.ocp1;

class Base {
    protected void m1() {
       System. out .println( "Base: m1()" );
    }
}

class Derived extends Base {
    void m1() {
       System. out .println( "Derived: m1()" );
    }
}

public class TestBaseDerived {
    public static void main(String[] args) {
       Base b = new Derived();
       b.m1();
    }
}
```

What will be the result of compiling and executing TestBaseDerived class?

A.	Base: m1()	B.	Derived: m1()
C.	Base: m1() Derived: m1()	D.	None of the other options

5.1.8 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
      String [] arr = { "*" , "**" , "***" ,
            "****" , "*****" , "******" };
      Predicate<String> pr1 = s -> s.length() < 4 ;
       print (arr, pr1);
   }

    private static void print(String [] arr,
              Predicate<String> predicate) {
      for (String str : arr) {
         if (predicate.test(str)) {
           System. out .println(str);
         }
      }
   }
}
```

What will be the result of compiling and executing Test class?

A.	**** ***** ******	B.	* ** ***
C.	* ** *** ****	D.	* ** *** **** ***** ******

5.1.9 Below are the possible definitions of Printable interface:

1.

```
   public abstract interface Printable {}
```

2.

```
public interface Printable {
   protected void print();
}
```

3.

```
public interface Printable {
    int i ;
    void print();
}
```

4.

```
public interface Printable {
    void print();

     default void log() {
    }

     private default void log1() {
    }

     private static void log2() {
    }

     static int getNumLogs() {
        return - 1 ;
    }
}
```

5.

```
@MarkerInterface
public interface Printable {}
```

6.

```
public interface Printable {
   public static final int x = 10 ;
   public final int y = 20 ;
   int z = 30 ;
}
```

How many definitions are valid ?

A. None of the definitions is valid

B. Only 1 definition is valid

C. 2 definitions are valid

D. 3 definitions are valid

E. 4 definitions are valid

F. 5 definitions are valid

G. All 6 definitions are valid

5.1.10 Below options represent the module descriptor file name and its contents for the module named 'com.data.sort':

1.

File name: module-info.java

```
package com.data.sort;

module com.data.sort {
}
```

2.

File name: module-info.java

```
package x;

module com.data.sort {
    exports x;
}
```

3.

File name: module-info.java

```
import java.io.*;

module com.data.sort {
}
```

4.

File name: module-info.java

```
import java.sql.*;

module com.data.sort {
}
```

5 .

File name: module-info.java

```
package com.udayankhattry;
```

```
import java.lang.*;

module com.data.sort {
}
```

6.

File name: module-desc.java

```
module com.data.sort {
}
```

7.

File name: module-info.java

```
import java.sql.*;

module com.data.sort {
    requires java.sql;
}
```

How many of the above options are valid module descriptor?

A. None of the options is valid

B. Only one option is valid

C. Two options are valid

D. Three options are valid

E. More than three options are valid

5.1.11 Consider the code of Test.java file:

```
package com.udayankhattry.ocp1;

class Report {
    public String generateReport() {
        return "CSV" ;
    }

     public Object generateReport() {
        return "XLSX" ;
    }
}

public class Test {
    public static void main(String[] args) {
        Report rep = new Report();
        String csv = rep.generateReport();
        Object xlsx = rep.generateReport();
        System. out .println(String. join ( ":" , csv, (String)xlsx));
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. CSV:XLSX

D. CSV

E. XLSX

F. CSVXLSX

5.1.12 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        StringBuilder sb =
                new StringBuilder( 20 ); //Line n1
         sb.append( "A" .repeat( 25 )); //Line n2
         System. out .println(sb.
                toString().length()); //Line n3

           sb.setLength( 10 ); //Line n4
         System. out .println(sb.
                toString().length()); //Line n5
    }
}
```

What will be the result of compiling and executing Test class?

A.	20 20	B.	25 10

C.	20 10	D.	25 25
E.	10 10		

5.1.13 Given code of Thought.java file:

```
public class Thought {
    /*INSERT*/ {
      System. out .println( "All is well" );
    }
}
```

Which 3 options, if used to replace /*INSERT*/, will compile successfully and on execution will print "All is well" on to the console ?

A. public void static main(String [] args)

B. protected static void main(String [] args)

C. public void main(String... args)

D. static public void Main(String [] args)

E. static public void main(String [] args)

F. public static void main(String [] a)

G. public static Void main(String [] args)

H. public static void main(String... a)

5.1.14 Consider below code of Test.java file:

```
public class Test {
    public static void main(String [] args) {
      System. out .println( "String" );
    }

    public static void main(Integer [] args) {
      System. out .println( "Integer" );
    }
    public static void main( byte [] args) {
      System. out .println( "byte" );
   }
}
```

And the commands:

```
javac Test.java
java Test 10
```

What is the result?

A. Integer
B. String
C. byte
D. Compilation error
E. An Exception is thrown at runtime

5.1.15 Which of the following commands displays the list of system modules?

A. javac --list
B. javac --list-modules
C. javac --list-system-modules
D. java --list
E. java --list-modules
F. java --list-system-modules

5.1.16 Consider below code fragment:

```
interface Printable {
    public void setMargin();
    public void setOrientation();
}

abstract class Paper implements Printable { //Line 7
    public void setMargin() {}
    //Line 9
}
```

```
class NewsPaper extends Paper { //Line 12
    public void setMargin() {}
    //Line 14
}
```

Above code currently causes compilation error. Which 2 modifications, done independently, enable the code to compile?

A.	Replace the code at Line 7 with: class Paper implements Printable {
B.	Insert at Line 9: public abstract void setOrientation();
C.	Replace the code at Line 12 with: abstract class NewsPaper extends Paper {
D.	Insert at Line 14: public void setOrientation() {}

5.1.17 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface I9 {
    void print();
}

public class Test {
    public static void main(String[] args) {
        int i = 400 ;
        I9 obj = () -> System. out .println( i );
        obj.print();
        System. out .println(++i);
    }
}
```

What will be the result of compiling and executing Test class?

A.	400 400	B.	400 401
C.	Compilation error	D.	Exception is thrown at runtime

5.1.18 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.sql.SQLException;

public class Test {
    private static void checkData() throws SQLException {
        try {
            throw new SQLException();
        } catch (Exception e) {
            e = null ; //Line 10
            throw e; //Line 11
        }
    }

    public static void main(String[] args) {
        try {
            checkData (); //Line 17
        } catch (SQLException e) {
            System. out .println( "NOT AVAILABLE" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. NOT AVAILABLE is printed on to the console and program terminates successfully

B. Program ends abruptly

C. Line 10 causes compilation failure

D. Line 11 causes compilation failure

E. Line 17 causes compilation failure

5.1.19 Given code of Test.java file:

```
public class Test {
    private static int [] arr ;
    public static void main(String [] args) {
        if ( arr . length > 0 && arr != null ) {
            System. out .println( arr [ 0 ]);
        }
    }
}
```

Predict Output, if the above code is run with given command?

```
java Test.java
```

A. No Output

B. NullPointerException is thrown at runtime

C. ArrayIndexOutOfBoundsException is thrown at runtime

D. Error is thrown at runtime

5.1.20 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   \---com.udayan.test
|           module-info.java
|           Test.java
|
\---cls
```

Below is the code of module-info.java file:

```
module com.udayan.test {
}
```

and below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String... args) {
        System. out .println( "HEALTH IS WEALTH" );
    }
}
```

And below command executed from C:\

```
javac -d cls --module-source-path src --module com.udayan.test
```

What will be the full path of generated Test.class file?

A. C:\cls\com.udayan.test\Test.class

B. C:\cls\com\udayan\test\Test.class

C. C:\cls\com.udayankhattry.ocp1\Test.class

D. C:\cls\com\udayankhattry\ocp1\Test.class

E. C:\cls\com.udayan.test\com\udayankhattry\ocp1\Test.class

F. C:\cls\com\udayan\test\com\udayankhattry\ocp1\Test.class

5.1.21 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       StringBuilder sb = new StringBuilder(
              "Friends are treasures" );
       sb.delete( 0 , 100 );
       System. out .println(sb.length());
    }
}
```

What will be the result of compiling and executing Test class?

A. 21

B. 0

C. 16

D. An exception is thrown at runtime

5.1.22 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        for ( var i = 5 ; i >= 1 ; i--) { //Line n1
            System. out .println(
              "*" .repeat(i)); //Line n2
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	* ** *** ****	B.	**** *** ** *
C.	* ** *** **** *****	D.	***** **** *** ** *
E.	Compilation error	F.	Line n2 causes runtime error

5.1.23 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.IOException;

class Parent {
    Parent() throws IOException {
        System. out .print( "HAKUNA" );
    }
}

class Child extends Parent {
    Child() throws Exception {
        System. out .println( "MATATA" );
    }
}

public class Test {
    public static void main(String[] args)
                throws Exception {
         new Child();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in both Parent and Child classes
B. Compilation error only in Parent class
C. Compilation error only in Child class
D. Test class executes successfully and prints HAKUNAMATATA on to the console
E. Test class executes successfully and prints MATATAHAKUNA on to the console

5.1.24 Interface java.util.function.Consumer<T> declares below non-overriding abstract method:

```
void accept(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.List;

class Book {
    private String name ;
    private String author ;
    private double price ;

    Book(String name, String author, double price) {
        this . name = name;
        this . author = author;
        this . price = price;
    }

    public String getName() {
        return name ;
    }

    public String getAuthor() {
        return author ;
    }

    public double getPrice() {
        return price ;
    }
    public void setPrice( double price) {
        this . price = price;
    }

    public String toString() {
        return String. join ( ":" , name , author , price + "" );
    }
}

public class Test {
    public static void main(String[] args) {
        var books = List. of (
                new Book ( "Head First Java" , "Kathy Sierra" , 19.5 ),
       new Book ( "Java SE 11 Programmer I -1Z0-815 Practice Tests" ,
"Udayan Khattry" , 9.99 ),
         new Book ( "Java - The Complete Reference" , "Herbert
Schildt" , 14.0 ));

          books.forEach(b ->
                b.setPrice(b.getPrice() - 4 )); //Line n1
        books.forEach(b ->
                System. out .println(b)); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation error in Book class
B.	Compilation error in Test class
C.	On execution Test class throws an exception
D.	Head First Java:Kathy Sierra:19.5 Java SE 11 Programmer I -1Z0-815 Practice Tests:Udayan Khattry:9.99 Java - The Complete Reference:Herbert Schildt:14.0
E.	Head First Java:Kathy Sierra:15.5 Java SE 11 Programmer I -1Z0-815 Practice Tests:Udayan Khattry:5.99 Java - The Complete Reference:Herbert Schildt:10.0

5.1.25 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;
import java.io.IOException;

abstract class Super {
   public abstract void m1() throws IOException;
}

class Sub extends Super {
    @Override
    public void m1() throws IOException {
        throw new FileNotFoundException();
    }
}

public class Test {
    public static void main(String[] args) {
        Super s = new Sub();
        try {
            s.m1();
        } catch (IOException e) {
            System. out .print( "A" );
        } catch (FileNotFoundException e) {
            System. out .print( "B" );
        } finally {
            System. out .print( "C" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. AC

B. BC

C. class Sub causes compilation error

D. class Test causes compilation error

5.1.26 Consider below codes of 2 java files:

```
//GetSetGo.java
package com.udayankhattry.ocp1;

public interface GetSetGo {
    int count = 1 ; //Line n1
}


//Test.java
package com.udayankhattry.ocp1;

public class Test {
     public static void main(String[] args) {
        GetSetGo [] arr = new GetSetGo[ 5 ]; //Line n2
         for (GetSetGo obj : arr) {
            obj.count++; //Line n3
         }
        System. out .println(GetSetGo.count); //Line n4
     }
}
```

Which of the following statements is correct?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Line n4 causes compilation error

E. Test class compiles successfully and on execution prints 5 on to the console

F. Test class compiles successfully and on execution prints 6 on to the console

5.1.27 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;

public class Test {
    static String [] names = { "Williamson.pdf" ,
                  "Finch.pdf" , "Kohli.pdf" , "Morgan.pdf" };
     public static void main(String[] args) {
        try {
            if ( search ( "virat.pdf" ))
               System. out .println( "FOUND" );

           } catch (FileNotFoundException ex) {
           System. out .println( "NOT FOUND" );
        }
    }

     private static boolean search(String name)
                     throws FileNotFoundException {
        for ( int i = 0 ; i <= 4 ; i++) {
            if ( names [i].equalsIgnoreCase(name)) {
                return true ;
            }
        }
        throw new FileNotFoundException();
    }
}
```

What will be the result of compiling and executing Test class?

A. FOUND

B. NOT FOUND

C. Compilation error

D. None of the other options

5.1.28 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Rideable {
    void ride(String name);
}

class Animal {}

class Horse extends Animal implements Rideable {
    public void ride(String name) {
        System. out .println(name.toUpperCase() +
                " IS RIDING THE HORSE" );
    }
}

public class Test {
    public static void main(String[] args) {
        Animal horse = new Horse();
        /*INSERT*/
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print EMMA IS RIDING THE HORSE on to the console? Select ALL that apply.

A. horse.ride("EMMA");

B. (Horse)horse.ride("EMMA");

C. ((Horse)horse).ride("Emma");

D. (Rideable)horse.ride("emma");

E. ((Rideable)horse).ride("emma");

F. (Rideable)(Horse)horse.ride("EMMA");

G. (Horse)(Rideable)horse.ride("EMMA");

H. ((Rideable)(Horse)horse).ride("EMMA");

I. ((Horse)(Rideable)horse).ride("emma");

5.1.29 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String... args) {
        System. out .println( "Java 11 is awesome!!!" );
    }
}
```

Location of files:

```
F:
└───────codes
    ├───────src
    │    └───────com
    │        └───────udayankhattry
    │            └───────ocp1
    │                  Test.java
    │
    └───────classes
        └───────com
            └───────udayankhattry
                └───────ocp1
                      Test.class
```

You are currently at 'codes' folder.

F:\codes>

Which of the following java commands will print 'Java 11 is awesome!!!' on to the console?

A.	java Test
B.	java com.udayankhattry.ocp1.Test
C.	java -cp classes\com\udayankhattry\ocp1\ Test
D.	java -cp classes\ com.udayankhattry.ocp1.Test
E.	java -cp classes\com com.udayankhattry.ocp1.Test

5.1.30 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Operator< T > {
   public abstract T operation( T t1, T t2);
}

public class Test {
    public static void main(String[] args) {
       System. out .println( /*INSERT*/ );
    }
}
```

Which of the following statements can be used to replace /*INSERT*/ such that there are no compilation errors?

A. (String s1, String s2) -> s1 + s2

B. (s1, s2) -> s1 + s2

C. (s1, s2) -> { return s1 + s2; }

D. None of the other options

5.1.31 Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
      P p = new R(); //Line n1
       System. out .println(p.compute( "Go" )); //Line n2
    }
}

class P {
   String compute(String str) {
       return str.repeat( 3 );
   }
}

class Q extends P {
   String compute(String str) {
       return super .compute(str.toLowerCase());
   }
}

class R extends Q {
   String compute(String str) {
       return super .compute(str.replace( 'o' , 'O' ));
   }
}
```

And the command:

java Test.java

What is the result?

A. Above command causes error

B. gogogo

C. GoGoGo

D. GOGOGO

E. Go

F. GO

G. go

5.1.32 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    static var arr = new Boolean[ 1 ];
    public static void main(String[] args) {
        if ( arr [ 0 ]) {
            System. out .println( true );
        } else {
            System. out .println( false );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. true

B. false

C. Compilation error

D. An exception is thrown at runtime

5.1.33 Consider below code of University.java file:

```
package com.udayankhattry.ocp1;

public class Teacher {
    public static void main(String [] args) {
        System. out .println( "Teacher" );
    }
}
```

```
public class Student {
    public static void main(String [] args) {
      System. out .println( "Student" );
    }
}
```

And the command:

```
java University.java
```

What is the result?

A.	Error as single-file source-code programs cannot contain package statement
B.	Error as multiple public classes are there in University.java file
C.	Teacher
D.	Student
E.	Teacher Student
F.	Student Teacher

5.1.34 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       int x = 5 ;
       while (x < 10 )
          System. out .println(x);
      x++;
    }
}
```

A.	Compilation error	B.	5

			6 7 8 9
C.	It will go in an infinite loop	D.	Produces no output

5.1.35 You have below 3rd party modular jar:
C:\third-party\jars\conncetivity.jar

Module name is: com.db.connect
and it contains com.jdbc.utils.Connect class which contains main(String...) method.

Which of the following commands executed from C:\ will successfully execute the com.jdbc.utils.Connect class?

A.	java -p third-party -m com.db.connect/com.jdbc.utils.Connect
B.	java -p third-party\jars -m com.db.connect
C.	java -p third-party* -m com.db.connect/com.jdbc.utils.Connect
D.	java -p third-party\jars\conncetivity.jar -m com.db.connect
E.	java -p third-party\jars -m com.db.connect/com.jdbc.utils.Connect
F.	java -m com.db.connect/com.jdbc.utils.Connect -p third-party\jars

5.1.36 You have below two modular jars available on your windows system:
1. C:\jars\jar1\secret_one.jar
Module name is 'secret' and it contains 'com.udayankhattry.ocp1.Secret' class, corresponding java source has below code:

```
package com.udayankhattry.ocp1;

public class Secret {
    public static void main(String... args) {
        var str = "ADTETFAECNKD" ;
        for ( int i = 0 ; i < str.length(); i+= 2 ) {
            System. out .print(str.charAt(i));
        }
    }
}
```

2. C:\jars\jar2\secret_two.jar
Module name is 'secret' and it contains 'com.udayankhattry.ocp1.Secret' class, corresponding java source has below code:

```
package com.udayankhattry.ocp1;

public class Secret {
    public static void main(String... args) {
        var str = "ADTETFAECNKD" ;
        for ( int i = 1 ; i < str.length(); i+= 2 ) {
            System. out .print(str.charAt(i));
        }
    }
}
```

What will be the result of executing below command from C:\?

```
java -p jars\jar2\secret_two.jar;jars\jar1\secret_one.jar -m
secret/com.udayankhattry.ocp1.Secret
```

A. ATTACK

B. DEFEND

C. It causes error because of the duplicate modules found

5.1.37 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        var x = 7.85 ; //Line n1
        var y = 5.25f ; //Line n2
        var a = ( int )x + ( int )y; //Line n3
        var b = ( int )(x + y); //Line n4
        System. out .println(a + b);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error at Line n3
B. Compilation error at Line n4
C. 24
D. 25
E. 26

5.1.38 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      StringBuilder sb =
            new StringBuilder( "HavePatience" );
      sb.delete( 4 , 5 ).insert( 4 , " P" ).
            toString().toUpperCase();
      System. out .println(sb);
   }
}
```

What will be the result of compiling and executing Test class ?

A. HAVE PTIENCE
B. HAVE PATIENCE
C. Have Patience
D. Have Ptience
E. HavePatience
F. HAVEPATIENCE
G. Haveatience
H. HAVEATIENCE

5.1.39 java.lang.IllegalArgumentException extends java.lang.RuntimeException

Given code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
   public static void convert(String s) throws
            IllegalArgumentException, RuntimeException,
            Exception {
       if (s.length() == 0 ) {
          throw new RuntimeException(
            "LENGTH SHOULD BE GREATER THAN 0" );
       }
   }
    public static void main(String [] args) {
       try {
          convert ( "" );
       }
       catch (IllegalArgumentException |
            RuntimeException | Exception e) { //Line 14
          System. out .println(e.getMessage()); //Line 15
       } //Line 16
       catch (Exception e) {
         e.printStackTrace();
       }
   }
}
```

Line 14 causes compilation error. Which of the following changes enables to code to print LENGTH SHOULD BE GREATER THAN 0 ?

A. Replace Line 14 with 'catch(RuntimeException | Exception e) {'

B. Replace Line 14 with 'catch(IllegalArgumentException | Exception e) {'

C. Replace Line 14 with 'catch(IllegalArgumentException | RuntimeException e) {'

D. Replace Line 14 with 'catch(RuntimeException e) {'

E. Comment out Line 14, Line 15 and Line 16

5.1.40 Consider the code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static void m( int i) {
        System. out .print( 1 );
    }

    private static void m( int i1, int i2) {
        System. out .print( 2 );
    }

    private static void m( char ... args) {
        System. out .print( 3 );
    }

    public static void main(String... args) {
        m ( 'A' );
        m ( 'A' , 'B' );
        m ( 'A' , 'B' , 'C' );
        m ( 'A' , 'B' , 'C' , 'D' );
    }
}
```

What will be the result of compiling and executing Test class?

A. Above code causes compilation error

B. It compiles successfully and on execution prints 3333 on to the console

C. It compiles successfully and on execution prints 1233 on to the console

D. It compiles successfully and on execution prints 1333 on to the console

5.1.41 Consider below code of Test.java file:

```
class Super {
    Super( int i) {
        System. out .println( 100 );
    }
}
```

```
class Sub extends Super {
    Sub() {
        System. out .println( 200 );
    }
}

public class Test {
     public static void main(String[] args) {
         new Sub();
    }
}
```

What will be the result of compiling and executing above code?

A.	200	B.	200 100
C.	100 200	D.	Compilation Error in Super class
E.	Compilation Error in Sub class		

5.1.42 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java. util.ArrayList;
import java.util.List;

class Employee {
     private String name ;
     private int age ;

      Employee(String name, int age) {
         this . name = name;
         this . age = age;
    }

      public String toString() {
         return "Employee[" + name + ", " + age + "]" ;
    }

     public boolean equals(Object obj) {
        if (obj instanceof Employee) {
           Employee emp = (Employee)obj;
            if ( this . name .equals(emp. name ) &&
              this . age == emp. age ) {
                return true ;
            }
        }
        return false ;
    }
}

public class Test {
    public static void main(String[] args) {
       List<Employee> employees = new ArrayList<>();
       employees.add( new Employee( "William" , 25 ));
       employees.add( new Employee( "William" , 27 ));
       employees.add( new Employee( "William" , 25 ));
       employees.add( new Employee( "William" , 25 ));

          employees.remove( new Employee( "William" , 25 ));

          for (Employee emp : employees) {
           System. out .println(emp);
       }
    }
}
```

What will be the result of compiling and executing Test class?

A.	Employee[William, 27] Employee[William, 25] Employee[William, 25]	B.	Employee[William, 25] Employee[William, 27] Employee[William, 25]
C.	Employee[William, 27]	D.	Employee[William, 25] Employee[William, 27] Employee[William, 25] Employee[William, 25]

5.1.43 Consider below code of Circus.java file:

```
//Circus.java
package com.udayankhattry.ocp1;

class Animal {
    protected void jump() {
       System. out .println( "ANIMAL" );
    }
}

class Cat extends Animal {
    public void jump( int a) {
       System. out .println( "CAT" );
    }
}

class Deer extends Animal {
    public void jump() {
       System. out .println( "DEER" );
    }
}

public class Circus {
    public static void main(String[] args) {
       Animal cat = new Cat();
       Animal deer = new Deer();
       cat.jump();
       deer.jump();
    }
}
```

What will be the result of compiling and executing above code?

A.	ANIMAL DEER	B.	CAT DEER
C.	ANIMAL ANIMAL	D.	CAT ANIMAL

5.1.44 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
      List<StringBuilder> gemStones = new ArrayList<>();
      gemStones.add( new StringBuilder( "Sapphire" ));
      gemStones.add( new StringBuilder( "Emerald" ));
      gemStones.add( new StringBuilder( "Ruby" ));

         if (gemStones.contains( new StringBuilder( "Sapphire" ))) {
          gemStones.add( new StringBuilder( "Diamond" ));
      }

         System. out .println(gemStones.size());
   }
}
```

What will be the result of compiling and executing Test class?

A. 4

B. 3

C. Compilation error

D. Runtime exception

5.1.45 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       var list = new ArrayList<>(); //Line n1
       list.add( "TAKE" );
      list.add( "THE" );
```

```
        list.add( "RISK" );
          System. out .println(String.join( "." , list)); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error at Line n1
B. Compilation error at Line n2
C. An exception is thrown at runtime
D. TAKE.THE.RISK.
E. TAKE.THE.RISK
F. TAKETHERISK
G. TAKETHERISK.

5.1.46 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;

interface Printer {
    default void print() throws FileNotFoundException {
      System. out .println( "PRINTER" );
    }
}

class FilePrinter implements Printer {
    public void print() { //Line n1
       System. out .println( "FILE PRINTER" );
    }
}
public class Test {
    public static void main(String[] args) {
      Printer p = new FilePrinter(); //Line n2
       p.print(); //Line n3
       var fp = new FilePrinter(); //Line n4
       fp.print(); //Line n5
    }
```

}

What will be the result of compiling and executing Test class?

A. Compilation error at Line n1
B. Compilation error at Line n2
C. Compilation error at Line n3
D. Compilation error at Line n4
E. Compilation error at Line n5
F. 1122 is printed on to the console
G. 1111 is printed on to the console
H. 2211 is printed on to the console
I. 2222 is printed on to the console

5.1.47 Which of the following 2 options correctly define the Exam class?

A.	**package** com.udayankhattry.oca;
	package com.udayankhattry.ocp;
	public class Exam {
	}

B.	**package** com.udayankhattry.oca;
	package com.udayankhattry.ocp;
	import java.io.*
	public class Exam {
	}

C.	**package** com.udayankhattry.oca;
	import java.util.*;
	public class Exam {
	}

D.	**public class** Exam {
	    package com.udayankhattry.oca;
	}

E.	*/*** *Exam class.* **/* **public class** Exam {

	}
	package com.udayankhattry.oca;

F.	//Exam.java
	package com.udayankhattry.oca;
	public class Exam {
	}

5.1.48 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var var = 0 ; //Line n1
        var: for (; var < 3 ; var++) {   //Line n2
            if (var % 2 == 0 ) {
                continue var; //Line n3
            }
            var++; //Line n4
        }
        System. out .println(var);
    }
}
```

Which of the following statements is true?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. Line n4 causes compilation error

E. Code compiles successfully

5.1.49 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        /*INSERT*/
```

```
         arr[ 0 ] = 5 ;
        arr[ 1 ] = 10 ;
        System. out .println( "[" + arr[ 0 ] +
                 ", " + arr[ 1 ] + "]" );
    }
}
```

For the class Test, which options, if used to replace /*INSERT*/, will print [5, 10] on to the console? Select 3 options.

A	. **short** arr [] = **new short** [2];
B	. **short** [2] arr;

C	. **short** [] arr;
	arr = **new short** [2];

D	. **short** [] arr = {};
E	. **short** [] arr = **new short** []{ 100 , 100 };
F	. **short** [] arr = **new short** [2]{ 100 , 100 };

5.1.50 Consider below code of Test.java file:

```
public class Test {
    public static void print() {
        System. out .println( "STATIC METHOD!!!" );
    }

     public static void main(String[] args) {
        Test obj = null ; //Line n1
         obj. print (); //Line n2
     }
}
```

What will be the result of compiling and executing Test class?

A. An Exception is thrown at runtime

B. Compilation error at Line n2

C. It prints STATIC METHOD!!! on to the console

D. None of the other options

5.1.51 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        final String fName = "James" ;
        String lName = "Gosling" ;
        String name1 = fName + lName;
        String name2 = fName + "Gosling" ;
        String name3 = "James" + "Gosling" ;
        System. out .println(name1 == name2);
        System. out .println(name2 == name3);
    }
}
```

What will be the result of compiling and executing Test class?

A.	true true	B.	true false
C.	false false	D.	false true

5.1.52 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        final String str = "+" ;
        System. out .println(str.repeat( 2 ) == "++" );
    }
}
```

And the command:

```
java --source 10 Test.java
```

What is the result ?

A. true

B. false

C. +

D. ++

E. Above command causes an error

5.1.53 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        double price = 90000 ;
        String model;
        if (price > 100000 ) {
            model = "Tesla Model X" ;
        } else if (price <= 100000 ) {
            model = "Tesla Model S" ;
        }
        System. out .println(model);
    }
}
```

What will be the result of compiling and executing Test class?

A. Tesla Model X

B. Tesla Model S

C. null

D. Compilation Error

5.1.54 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface ILog {
    void log();
    boolean equals();
}

public class Test {
    public static void main(String[] args) {
        ILog obj = () ->
```

```
            System. out .println( "FEELING FANTASTIC" );
        obj.log();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. FEELING FANTASTIC

C. No output

D. An exception is thrown at runtime

5.1.55 Choose the option that meets the following specification: Create a well encapsulated class Clock with one instance variable model. The value of model should be accessible and modifiable outside Clock.

A.
```
public class Clock {
    public String model ;
}
```

B.
```
public class Clock {
    public String model ;
    public String getModel() { return model ; }
    public void setModel(String val) { model = val; }
}
```

C.
```
public class Clock {
    private String model ;
    public String getModel() { return model ; }
    public void setModel(String val) { model = val; }
}
```

D.
```
public class Clock {
    public String model ;
    private String getModel() { return model ; }
    private void setModel(String val) { model = val; }
}
```

5.1.56 For the implied readability, which of the below directives

is used in the module descriptor?

A. exports

B. exports to

C. requires

D. requires from

E. requires transitive

F. require

G. require transitive

5.1.57 Consider below code snippet:

```
import java.util.ArrayList;
import java.util.List;

public class Test {
    List list1 = new ArrayList<String>(); //Line n5
    List<String> list2 = new ArrayList(); //Line n6
    List<> list3 = new ArrayList<String>(); //Line n7
    List<String> list4 = new ArrayList<String>(); //Line n8
    List<String> list5 = new ArrayList<>(); //Line n9
}
```

Which of the following statements compile without any warning? Select ALL that apply.

A. Line n5

B. Line n6

C. Line n7

D. Line n8

E. Line n9

5.1.58 How can you force JVM to run Garbage Collector?

A. By calling: Runtime.getRuntime().gc();

B. By calling: System.gc();

C. By setting the reference variable to null

D. JVM cannot be forced to run Garbage Collector

5.1.59 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        char c1 = 'a' ; //ASCII code of 'a' is 97
        int i1 = c1; //Line n1
        System. out .println(i1); //Line n2
    }
}
```

What is the result of compiling and executing Test class?

A. a

B. 97

C. Line n1 causes compilation failure

D. Line n1 causes runtime error

5.1.60 What will be the output of compiling and executing the Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int a = 5 ;
        int x = 10 ;
        switch (x) {
            case 10 :
                a *= 2 ;
            case 20 :
                a *= 3 ;
            case 30 :
                a *= 4 ;
        }
        System. out .println(a);
    }
}
```

A. 5

B. 10

C. 30

D. 120

5.1.61 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        for :
        for ( int i = 2 ; i <= 100 ; i = i + 2 ) {
            for ( int j = 1 ; j <= 10 ; j++) {
               System. out .print(i * j + " \t " );
           }
           System. out .println();
            if (i == 10 ) {
                break for ;
           }
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Total 5 non-blank rows will be there in the output

B. Total 6 non-blank rows will be there in the output

C. Total 50 non-blank rows will be there in the output

D. Total 51 non-blank rows will be there in the output

E. Total 100 non-blank rows will be there in the output

F. Total 101 non-blank rows will be there in the output

G. Compilation error

5.1.62 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
   boolean test(T t);
```

Consider below Lambda expression:

```
Predicate<String> predicate = s -> true ;
```

Which of the lambda expressions can successfully replace the lambda expression in above statement?

A.	s -> { true }	B.	s -> { true ;}
C.	s -> { return true }	D.	s -> { return true ;}

5.1.63 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var i = 1 ;
        var j = 5 ;
        var k = 0 ;
        A: while ( true ) {
            i++;
            B: while ( true ) {
                j--;
                C: while ( true ) {
                    k += i + j;
                    if (i == j)
                        break A;
                    else if (i > j)
                        continue A;
                    else
                        continue B;
                }
            }
        }
        System. out .println(k);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Program never terminates as above code causes infinite loop

C. 6

D. 11

E. 15

F. None of the other options

5.1.64 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util. ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
      List<StringBuilder> list = new ArrayList<>();
      list.add( new StringBuilder( "AAA" )); //Line n1
       list.add( new StringBuilder( "BBB" )); //Line n2
       list.add( new StringBuilder( "AAA" )); //Line n3

         list.removeIf(sb -> sb.equals(
            new StringBuilder( "AAA" ))); //Line n4
       System. out .println(list);
   }
}
```

What will be the result of compiling and executing Test class?

A. [AAA, BBB, AAA]

B. [BBB, AAA]

C. [BBB]

D. []

E. None of the other options

5.1.65 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String arr1 [], arr2, arr3 = null ; //Line n1
```

```
            arr1 = new String[ 2 ];
           arr1[ 0 ] = "A" ;
           arr1[ 1 ] = "B" ;
           arr2 = arr3 = arr1; //Line n2
            System. out .println(String. join (
                  "-" , arr2)); //Line n3
        }
    }
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Line n3 causes compilation error

D. It executes successfully and prints A-B on to the console

E. It executes successfully and prints B-A on to the console

F. It executes successfully and prints A on to the console

G. It executes successfully and prints B on to the console

5.1.66 Consider codes of 3 java files:

```
//Class1.java
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;

public class Class1 {
    public void read() throws FileNotFoundException {}
}


//Class2.java
public class Class2 {
    String Class2 ;
     public void Class2() {}
}


//Class3.java
public class Class3 {
     private void print() {
        private String msg = "HELLO" ;
        System. out .println(msg);
    }
}
```

Which of the following statements is true ?

A. Only Class1.java compiles successfully

B. Only Class2.java compiles successfully

C. Only Class3.java compiles successfully

D. Class1.java and Class2.java compile successfully

E. Class1.java and Class3.java compile successfully

F. Class2.java and Class3.java compile successfully

5.1.67 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String text = "    BE YOURSELF!    " ; //Contains multiple space characters
        System. out .println(text. /*INSERT*/ );
    }
}
```

**Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print "BE YOURSELF!" on to the console?
Select ALL that apply.**

A. trim()

B. strip()

C. stripLeading().stripTrailing()

D. ltrim().rtrim()

E. trimLeading().trimTrailing()

F. trimBoth()

5.1.68 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
       StringBuilder sb = new StringBuilder( "INHALE " );
       String s = sb.toString() + (sb.append( "EXHALE " ));
       System. out .println(s.strip().length());
    }
}
```

What will be the result of compiling and executing Test class?

A. 21

B. 20

C. 28

D. 27

E. 24

F. 23

G. 18

H. 17

5.1.69 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       StringBuilder sb = new StringBuilder( "Hello" );
       List<StringBuilder> list = new ArrayList<>();
       list.add(sb);
       list.add( new StringBuilder( "Hello" ));
       list.add(sb);
       sb.append( "World!" );

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Hello, Hello, Hello]

B. [HelloWorld!, Hello, Hello]

C. [HelloWorld!, Hello, HelloWorld!]

D. [HelloWorld!, HelloWorld!, HelloWorld!]

5.1.70 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String word = "REBUS" ;
        /* INSERT */
        System. out .println(word);
    }
}
```

Following options are available to replace /*INSERT*/:

1.	word = word.substring(2);
2.	word = word.substring(2 , 4);
3.	word = word.substring(2 , 5);
4.	word = word.replace("Re" , "");
5.	word = word.substring(2 , 6);
6.	word = word.delete(0 , 2);

How many of the above options can be used to replace /*INSERT*/ (separately and not together) such that given command prints BUS on to the console?

A. One option only

B. Two options only

C. Three options only

D. Four options only

E. Five options only

F. All 6 options

5.1.71 Consider below code of Test.java file:

```
//Test.java
package com.udayankhattry.ocp1;

class Parent {
     int i = 10 ;
    Parent( int i) {
        super ();
        this . i = i;
    }
}

class Child extends Parent {
    int j = 20 ;

     Child( int j) {
        super ( 0 );
        this . j = j;
    }

     Child( int i, int j) {
        super (i);
        this (j);
    }

}

public class Test {
     public static void main(String[] args) {
        Child child = new Child( 1000 , 2000 );
        System. out .println(child. i + ":" + child. j );
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in Parent(int) constructor

B. Compilation error in Child(int) constructor

C. Compilation error in Child(int, int) Constructor

D. Compilation error in Test class

E. It executes successfully and prints 1000:2000 on to the console

F. It executes successfully and prints 1000:0 on to the console

5.1.72 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<String> list = new ArrayList<>();
       list.add( 0 , "I" );
       list.set( 0 , "CAN" );

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [I]

B. [CAN]

C. [I, CAN]

D. [CAN, I]

E. An exception is thrown at runtime

5.1.73 Which of the following array declarations and initializations is NOT legal?

A. char [] arr1 [] = new char[5][];

B. int [] arr2 = {1, 2, 3, 4, 5};

C. int [] arr3 = new int[3]{10, 20, 30};

D. byte [] val = new byte[10];

5.1.74 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Lock {
    public void open() {
       System. out .println( "LOCK-OPEN" );
    }
}

class Padlock extends Lock {
    public void open() {
       System. out .println( "PADLOCK-OPEN" );
    }
}

class DigitalPadlock extends Padlock {
    public void open() {
        /*INSERT*/
    }
}

public class Test {
    public static void main(String[] args) {
       Lock lock = new DigitalPadlock();
       lock.open();
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print LOCK-OPEN on to the console?

A. super.open();

B. super.super.open();

C. ((Lock)super).open();

D. (Lock)super.open();

E. None of the other options

5.1.75 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Base {
    static void print() { //Line n1
        System. out .println( "BASE" );
    }
}

class Derived extends Base {
    static void print() { //Line n2
        System. out .println( "DERIVED" );
    }
}

public class Test {
    public static void main(String[] args) {
       Base b = null ;
       Derived d = (Derived) b; //Line n3
        d. print (); //Line n4
    }
}
```

Which of the following statements is true for above code?

A. Line n2 causes compilation error

B. Line n3 causes compilation error

C. Line n4 causes compilation error

D. Code compiles successfully and on execution Line n3 throws an exception

E. Code compiles successfully and on execution prints BASE on to the console

F. Code compiles successfully and on execution prints DERIVED on to the console

5.1.76 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int grade = 85 ;
        if (grade >= 60 )
           System. out .println( "Congratulations" );
           System. out .println( "You passed" );
        else
```

```
            System. out .println( "You failed" );
        }
}
```

What will be the result of compiling and executing Test class?

A.	Congratulations	B.	You failed
C.	Compilation error	D.	Congratulations You passed

5.1.77 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String [] args) {
      List<Integer> list = new ArrayList<Integer>();

        list.add( 27 );
      list.add( 27 );

        list.add( 227 );
      list.add( 227 );

        System. out .println(list.get( 0 ) == list.get( 1 ));
      System. out .println(list. get( 2 ) == list.get( 3 ));
    }
}
```

What will be the result of compiling and executing Test class?

A.	false false	B.	false true
C.	true true	D.	true false

5.1.78 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       String[] arr = { "L" , "I" , "V" , "E" }; //Line n1
        int i = - 2 ;

          if (i++ == - 1 ) { //Line n2
            arr[-(--i)] = "F" ; //Line n3
        } else if (--i == - 2 ) { //Line n4
            arr[-++i] = "O" ; //Line n5
        }

          System. out .println(String. join ( "" , arr)); //Line n6
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. LIVE

D. LIFE

E. LIVO

F. LOVE

G. LIOE

5.1.79 Which of the following code segments, written inside main method will compile successfully?

Select ALL that apply.

A	. var [] arr1 = **new** String[2];

B	. **var** num;
	num = 10 ;

C	. **final var** str = "Hello" ;
D	. **var** arr2 = { 1 , 2 , 3 };

E	. **for** (**var** i = 0 ; i < 2 ; i++) {
	System. *out* .println(i);

```
}
```

F.

```
String [] arr = { "A" , "E" , "I" , "O" , "U" };
for ( var x : arr) {
    System. out .println(x);
}
```

G.

```
var msg = null ;
```

H.

```
private var y = 100 ;
```

5.1.80 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        var arr = new int [] { 0 , 1 , 2 , 3 , 4 ,
             5 , 6 , 7 , 8 , 9 }; //Line n1
        String str = process(arr, 3 , 8 ); //Line n2
        System. out .println(str);
    }

    /*INSERT*/
}
```

Line n2 causes compilation error as process method is not found.
Which of the following method definitions, if used to replace /*INSERT*/, will resolve the compilation error?

A.

```
private static int [] process( int [] arr,
                                              int start, int end) {
    return null ;
}
```

B.

```
private static String process( int [] arr,
                                              int start, int end) {
    return null ;
}
```

C.

```
private static int process( int [] arr,
                                              int start, int end) {
    return null ;
}
```

D.

```
private static String[] process( int [] arr,
                                          int start, int end) {
    return null ;
}
```

E.

```
private static String process(var arr,
                                          int start, int end) {
    return null ;
}
```

F.

```
private static var process( int [] arr,
                                          int start, int end) {
    return null ;
}
```

5.2 Answers of Practice Test - 5 with Explanation

5.1.1 Answer: A

Reason :
Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

As compiler is aware that given infinite loop has an exit point, hence it doesn't mark Line n2 as unreachable. Line n2 doesn't cause any compilation error as well .

There is no compilation error for 3rd and 6th statements as syntax of calling join(...) method is correct.

In java it is allowed to use same method name as the class name (though not a recommended practice), hence 1st, 2nd and 3rd statements represent overloaded 'Sport' methods.
4th, 5th and 6th statements represent overloaded constructors as no return type is specified.

All 6 statements can be inserted in Sport class.

5.1.2 Answer: B

Reason :
First add parenthesis (round brackets) to the given expression: -a++.
There are 2 operators involved. unary minus and Postfix operator. Let's start with expression and value of a.

-a++; [a = 1000].
-(a++); [a = 1000] Postfix operator has got higher precedence than unary operator.
-(1000); [a = 1001] Use the value of a (1000) in the expression and after that increase the value of a to 1001.
-1000; [a = 1001] -1000 is printed on to the console.

5.1.3 Answer: C

Reason :
Binary plus (+) has got higher precedence than != operator. Let us group the expression.
"Equals??? " + 10 != 5
= ("Equals??? " + 10) != 5
[!= is binary operator, so we have to evaluate the left side first. +

operator behaves as concatenation operator.]
= "Equals??? 10" != 5
Left side of above expression is String, and right side is int. But String can't be compared to int hence compilation error.

5.1.4 Answer: A,C,D,E

Reason :
switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums. In this case long and double are invalid values to be passed in switch expression. char uses 16 bits (2 Bytes) and its range is 0 to 65535 (no signed bit reserved) so it can easily store value 10.

Please note that the identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

5.1.5 Answer: B

Reason :
Module 'string.util' has two packages: 'com.udayankhattry.ocp1.util' and 'com.udayankhattry.utility' and its module descriptor file (module-info.java) has exports directive only for 'com.udayankhattry.ocp1.util' package.

Module descriptor of 'testing' module has requires directive for 'string.util' module. Hence, 'testing' module has access to all the public classes in 'com.udayankhattry.ocp1.util' package and not 'com.udayankhattry.utility' package.

Therefore, there is a compilation error in Test.java file. Below 2 statements in Test.java file cause compilation error:
import com.udayankhattry.utility.StringUtility;
and
System.out.println(StringUtility.reverse("NOON"));

This is one more layer of encapsulation provided by the Java Module System, in which not all the public classes are accessible to everyone, as this was the case with pre-module codes.
public classes defined in exported packages are accessible and not the un-exported packages.

5.1.6 Answer: B

Reason :
There is no compilation error in Medicines.java file.

By looking at the contents of module descriptor file (module-info.java), it can be said that:
- Name of the module is: medicines
- exports com.allopathic to elizabeth_hospital; => This is known as qualified export, in which package 'com.allopathic' is made available to only named module 'elizabeth_hospital'.
If the module name, to which package is exported to (elizabeth_hospital in this case), is not available, then compiler displays the warning but compiles the code .

After executing the given javac command, you will have below directory/file structure:

```
C:
+---out
|   \---medicines
|       |   module-info.class
|       |
|       \---com
|           \---allopathic
|                   Medicines.class
|
\---src
    \---medicines
        |   module-info.java
        |
        \---com
            \---allopathic
                    Medicines.java
```

5.1.7 Answer: D

Reason :
Derived class overrides method m1() of Base class.
Access modifier of method m1() in Base class is protected, so overriding method can use protected or public. But overriding method in this case used default modifier and hence there is compilation error.

5.1.8 Answer: B

Reason :
Lambda expression for Predicate is: s -> s.length() < 4. This means return true if passed string's length is < 4.
So first three array elements are printed.

5.1.9 Answer: C

Reason :
public abstract interface Printable {}: ✓
Valid, as interface in java is implicitly abstract, so using abstract keyword doesn't cause any error.

```
public interface Printable {
    protected void print();
}: ✗
```

abstract method of the interface are implicitly public and if you provide access modifier for the abstract method of the interface, then only 'public' is allowed. As 'protected' is used for print() method, hence it causes compilation error.

```
public interface Printable {
    int i;
    void print();
}: ✗
```

Variables declared inside interface are implicitly public, static and final and therefore compiler complains about un-initialized final variable i.

```
public interface Printable {
    void print();

    default void log() {
    }

    private default void log1() {
    }


    private static void log2() {
    }

    static int getNumLogs() {
        return -1;
    }
}: ✗
```

As per Java 8, default and static methods were added in the interface and as per Java 9, private methods were added in the interface. default modifier is not allowed with private method of the interface, hence method log1() causes compilation error. Methods print(), log(), log2() and getNumLogs() compile successfully.

```
@MarkerInterface
public interface Printable {}: ✗
```

@MarkerInterface annotation is not available in Java and hence it causes compilation error.

```
public interface Printable {
    public static final int x = 10;
    public final int y = 20;
    int z = 30;
}: ✓
```

Interfaces can define public, static and final variables and these modifiers are implicit.
Hence, for 'y' compiler adds static modifier and for 'z' compiler adds public, static and final modifiers.

Therefore, only 2 interface definitions are valid.

5.1.10 Answer: C

Reason :
Name of the module descriptor file must be: module-info.java, hence option number 6 is invalid.

package declarations are not allowed in module-info.java file, hence options 1, 2 and 5 are invalid.

import statements are allowed in module descriptor file. These import statements are useful when you use 'provides' directive for Services but these are not in the scope of 1Z0-815 exam. For this exam you should know that import statements are allowed in module-info.java file.

By default all the module implicitly requires java.base module (one of the system module)

To get a list of all the system modules, use below command:

```
java --list-modules
```

And to check details of specific module (e.g. java.base), use below command:

```
java --describe-module java.base
```

You will notice that it exports java.io, java.lang packages but not java.sql package. Option 3 is valid.

To have `import java.sql.*;` in the module-info.java file, `requires java.sql;` directive is needed, hence option 4 is invalid but option 7 is valid.

So, out of seven given options, only two options (number 3 and 7) are valid.

5.1.11 Answer: A

Reason :
Both the methods of Report class have same signature(name and parameters match).
Having just different return types don't overload the methods and therefore Java compiler complains about duplicate generateReport() methods in Report class.

5.1.12 Answer: B

Reason :
`new StringBuilder(20);` creates a StringBuilder instance, whose capacity (internal char array's length) is 20. This initial capacity is not fixed and changes on adding / removing characters to the StringBuilder object.
Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".

Line n2 successfully appends 25 A's to the StringBuilder object.
Line n3 prints 25 on to the console.

At Line n4, `sb.setLength(10);` sets the length of StringBuilder object referred by 'sb' to 10 (it is reducing the length of StringBuilder object from 25 to 10), hence 'sb' refers to StringBuilder object containing 10

A's (last 15 A's are gone).
Hence, Line n3 prints 10 on to the console.

5.1.13 Answer: E,F,H

Reason :
As System.out.println needs to be executed on executing the Test class, this means special main method should replace /*INSERT*/.
Special main method's name should be "main" (all characters in lower case), should be static, should have public access specifier and it accepts argument of String [] type (Varargs syntax String... can also be used). String [] argument can use any identifier name, even though in most of the cases you will see "args" is used. Position of static and public can be changed but return type 'void' must come just before the method name.

Let's check all the given options one by one:
public void static main(String [] args): Compilation error as return type 'void' must come just before the method name 'main'.
protected static void main(String [] args): Compiles successfully but as this method is not public, hence an Error regarding missing main method is thrown on execution.
public void main(String... args): Compiles successfully but as this method is not static, hence an Error regarding non-static main method is thrown on execution.
static public void Main(String [] args): Compiles successfully but as 'M' is capital in method 'Main', hence it is not special main method. An Error regarding missing main method is thrown on execution .
static public void main(String [] args): Valid definition, it compiles successfully and on execution prints "All is well" on to the console.
public static void main(String [] a): Valid definition, it compiles successfully and on execution prints "All is well" on to the console.
public static Void main(String [] args): Compilation error as Void is a final class in Java and in this case compiler expects main method to return a value of Void type. If you add `return null;` to the main method code will compile successfully but on execution an Error will be thrown mentioning that return type must be 'void' ('v' in lower-case).
public static void main(String... a): Valid definition, it compiles successfully and on execution prints "All is well" on to the console.

5.1.14 Answer: B

Reason :
Like any other method, main method can also be overloaded. But main method called by JVM is always with String [] parameter. Don't get confused with 10 as it is passed as "10".
Execute above class with any command line arguments or 0 command line argument, output will always be "String".

5.1.15 Answer: E

Reason :
To get a list of all the system modules, use below command:
java --list-modules

5.1.16 Answer: C,D

Reason :
First you should find out the reason for compilation error. Methods declared in Printable interface are implicitly abstract, no issues with Printable interface.

class Paper is declared abstract and it implements Printable interface, it overrides setMargin() method but setOrientation() method is still abstract. No issues with class Paper as it is an abstract class and can have 0 or more abstract methods.

class NewsPaper is concrete class and it extends Paper class (which is abstract). So class NewsPaper must override setOrientation() method OR it must be declared abstract.
Hence, there are 2 possible solution:
Insert at Line 14: public void setOrientation() {}
OR
Replace the code at Line 12 with: abstract class NewsPaper extends Paper {

Replacing Line 9 with 'public abstract void setOrientation();' is not necessary and it will not resolve the compilation error in NewsPaper class.

Replacing Line 7 with 'class Paper implements Printable {' will cause compilation failure of Paper class as it inherits abstract method 'setOrientation'.

5.1.17 Answer: C

Reason :
Variable 'i' is a local variable and it is used in the lambda expression.
So, it should either be final or effectively final.
The last statement inside main(String []) method, increments value of i, which means it is not effectively final and hence compilation error.

5.1.18 Answer: D

Reason :
Exception is a java class, so `e = null;` is a valid statement and compiles successfully.

If you comment Line 10, and simply throw e, then code would compile successfully as compiler is certain that 'e' would refer to an instance of SQLException only.

But the moment compiler finds `e = null;`, `throw e;` (Line 11) causes compilation error as at runtime 'e' may refer to any Exception type.

NOTE: No issues with Line 17 as method checkData() declares to throw SQLException and main(String []) method code correctly handles it.

5.1.19 Answer: B

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
   (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java .

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence, `java Test.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Test.java
java -cp <memory> Test
```

Variable arr is a class variable of int [] type, so by default it is initialized to null.
In if block, arr.length > 0 is checked first. Accessing length property on null reference throws NullPointerException.

Correct logical if block declaration should be:
if(arr != null && arr.length > 0)

First check for null and then access properties/methods.

5.1.20 Answer: E

Reason :
Test class is defined under 'com.udayankhattry.ocp1'. Though it is the best practice to keep the source code files under package directory structure, e.g.
'C:\src\com.udayan.test\com\udayankhattry\ocp1\Test.java' is the logical path for Test.java file but it can legally be placed at any path.
Hence, 'C:\src\com.udayan.test\Test.java' is also allowed.
Please note that generated class files must be under proper directory structure.

Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this case, module source path is 'src' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'cls' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path> :

Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
It is not used in this case as given module is not dependent upon other compiled/packaged application modules.

Given command will first create the module-directory 'com.udayan.test' based on the module name specified in module-info.java file and will create the structure 'com\udayankhattry\ocp1\Test.class' (based on package statement in Test.java file) under 'com.udayan.test' directory.

Directory/file structure after executing given javac command:

```
C:
+---src
|  \---com.udayan.test
|          module-info.java
|          Test.java
|
\---cls
    \---com.udayan.test
        |  module-info.class
        |
        \---com
            \---udayankhattry
                \---ocp1
                        Test.class
```

Hence, correct option is:
C:\cls\com.udayan.test\com\udayankhattry\ocp1\Test.class

5.1.21 Answer: B

Reason :
'delete' method accepts 2 parameters: delete(int start, int end), where start is inclusive and end is exclusive.
This method throws StringIndexOutOfBoundsException for following

scenarios:
A. start is negative
B. start is greater than sb.length()
C. start is greater than end

If end is greater than the length of StringBuilder object, then StringIndexOutOfBoundsException is not thrown and end is set to sb.length() .
So, in this case, `sb.delete(0, 100);` is equivalent to `sb.delete(0, sb.length());` and this deletes all the characters from the StringBuilder object.
Hence, `System.out.println(sb.length());` prints 0 on to the console.

5.1.22 Answer: D

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

At Line n1, variable 'i' infers to int type, so given loop executes 5 times from values of i from 5 to 1.

Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".

1st iteration: System.out.println("*".repeat(5)); => prints ***** on to the console.
2nd iteration: System.out.println("*".repeat(4)); => prints **** on to the console.
3rd iteration: System.out.println("*".repeat(3)); => prints *** on to the console.

4th iteration: System.out.println("*".repeat(2)); => prints ** on to the console.
5th iteration: System.out.println("*".repeat(1)); => prints * on to the console.
Hence, output is:

```
*****
****
***
**
*
```

5.1.23 Answer: D

Reason :
It is legal for the constructors to have throws clause.
Constructors are not inherited by the Child class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Child class's constructor:

```
Child() throws Exception {
    super(); //added by the compiler
    System.out.println("MATATA");
}
```

super(); invokes the constructor of Parent class (which declares to throw IOException), but as no-argument constructor of Child class declares to throw Exception (super class of IOException), hence IOException is also handled. There is no compilation error and output is: HAKUNAMATATA

5.1.24 Answer: E

Reason :
There is no compilation error in the given code.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,

```
var x = "Java"; //x infers to String
var m = 10; //m infers to int
```

Given statement:
var books = List.of(...); => books refers to a List of Book type.

Static List.of() overloaded methods were added in Java 9 and these return an unmodifiable list containing passed elements. So, above list object referred by 'books' is unmodifiable. It is not allowed to invoke add/remove/set methods on the list object returned by List.of() method (no compilation error but an exception is thrown at runtime on invoking add/remove/set methods on books reference) but 3 Book instances are modifiable .

Iterable<T> interface has forEach(Consumer) method. List<E> extends Collection<E> & Collection<E> extends Iterable<E>, therefore forEach(Consumer) can easily be invoked on reference variable of List<E> type.
As Consumer is a Functional Interface, hence a lambda expression can be passed as argument to forEach() method.
forEach(Consumer) method performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Lambda expression at Line n1: `b -> b.setPrice(b.getPrice() - 4` is the correct implementation of `void accept(Book t);` and hence can easily be assigned to target type Consumer<Book>. Purpose of this lambda expression is to reduce the book price by 4 units. Hence, statement at Line n1 will reduce the price of 3 Book instances available in the list.

Lambda expression at Line n2: `b -> System.out.println(b)` is also the correct implementation of `void accept(Book t);` and hence can easily be assigned to target type Consumer<Book>. Purpose of this lambda expression is to print the Book object on to the console. toString() method of Book object is invoked and it uses static join method of String class to construct the display String.

Statement at Line n2 will print the details of 3 Book instances available in the list.

Output will be:

Head First Java:Kathy Sierra:15.5
Java SE 11 Programmer I -1Z0-815 Practice Tests:Udayan Khattry:5.99
Java - The Complete Reference:Herbert Schildt:10.0

5.1.25 Answer: D

Reason :
Method m1() of Sub class correctly overrides the method m1() of Super class, so there is no compilation error in Sub class.

FileNotFoundException extends IOException and hence catch block of FileNotFoundException should appear before the catch block of IOException.
Therefore, class Test causes compilation error.

5.1.26 Answer: C

Reason :
Variable 'count' declared inside interface GetSetGo is implicitly public, static and final. Line n1 compiles successfully.
Line n2 creates one dimensional array of 5 elements of GetSetGo type and all 5 elements are initialized to null. Line n2 compiles successfully.
Though correct way to refer static variable is by using the type name, such as GetSetGo.count but it can also be invoked by using GetSetGo reference variable. Hence, obj.count at Line n3 correctly points to the count variable at Line n1. But as variable 'count' is implicitly final, therefore obj.count++ causes compilation error. Line n3 fails to compile.
Line n4 compiles successfully as variable 'count' is implicitly static and GetSetGo.count is the correct syntax to point to 'count' variable of interface GetSetGo.

5.1.27 Answer: D

Reason :
search(String) method declares to throw FileNotFoundException, which is a checked exception. It returns true if match is found otherwise it throws an instance of FileNotFoundException.

main(String[]) provides try-catch block around `search("virat.pdf")` and catch handler checks for FileNotFoundException. Given code compiles successfully.

There are 4 elements in 'names' array, so starting index is 0 and end index is 3, but given for loop goes till index number 4.
As search string is "virat.pdf" (not present in names array), hence for loop will execute for i = 0, 1, 2, 3, 4.
For i = 4, `names[i].equalsIgnoreCase(name)` throws ArrayIndexOutOfBoundsException (it is a RuntimeException).
main(String []) method doesn't provide handler for ArrayIndexOutOfBoundsException and therefore stack trace is printed on to the console and program terminates abruptly.

5.1.28 Answer: C,E,H,I

Reason :
Let's check all the options one by one:
horse.ride("EMMA"); ✗ Variable 'horse' is of Animal type and ride(String) method is not defined in Animal class, therefore it causes compilation error .

(Horse)horse.ride("EMMA"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast.
horse.ride("EMMA") returns void, hence it cannot be casted to Horse type. This would cause compilation error.

((Horse)horse).ride("Emma"); ✓ Variable 'horse' refers to an instance of Horse type and variable 'horse' is casted to Horse type. Horse class has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output. As, name.toUpperCase() method is invoked, hence it doesn't matter in what case you pass the name, in the output name will always be displayed in the upper case.

(Rideable)horse.ride("emma"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast.
horse.ride("EMMA") returns void, hence it cannot be casted to Rideable type. This would cause compilation error.

((Rideable)horse).ride("emma"); ✓ Variable 'horse' refers to an instance of Horse type and variable 'horse' is casted to Rideable type (super type of Horse). Rideable interface has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

(Rideable)(Horse)horse.ride("EMMA"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride("EMMA") returns void, hence it cannot be casted to Horse type. This would cause compilation error.

(Horse)(Rideable)horse.ride("EMMA"); ✗ horse.ride("EMMA") will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride("EMMA") returns void, hence it cannot be casted to Rideable type. This would cause compilation error.

((Rideable)(Horse)horse).ride("EMMA"); ✓ Variable 'horse' refers to an instance of Horse type, it is first casted to Horse type and then casted to Rideable type. Rideable interface has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

((Horse)(Rideable)horse).ride("emma"); ✓ Variable 'horse' refers to an instance of Horse type, it is first casted to Rideable type and then casted to Horse type. Horse class has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

5.1.29 Answer: D

Reason :
To execute Test class, you should specify the -classpath or -cp option with java command, to specify the search path for Test class. This search path should contain whole path of the Test class(com\udayankhattry\ocp1\Test.class).
In this case, the search path is 'F:\codes\classes\' but as command is executed from 'F:\codes\' directory, hence relative path should be 'classes\'.
And you should also use fully qualified name of the class, which is com.udayankhattry.ocp1.Test.
Hence, correct command to executed Test class is: java -cp classes\ com.udayankhattry.ocp1.Test

5.1.30 Answer: D

Reason :
Reference variable to which lambda expression is assigned is known as target type. Target type can be a static variable, instance variable, local

variable, method parameter or return type. Lambda expression doesn't work without target type and target type must be a functional interface. In this case, println(Object) method is invoked but Object is a class and not a functional interface, hence no lambda expression can be passed directly to println method.

But you can first assign lambda expression to target type and then pass the target type reference variable to println(Object) method:

```
Operator<String> operator = (s1, s2) -> s1 + s2;
System.out.println(operator);
```

Or you can typecast lambda expression to target type. e.g. following works:

```
System.out.println((Operator<String>)(String s1, String s2) -> s1 + s2);
System.out.println((Operator<String>)(s1, s2) -> s1 + s2);
System.out.println((Operator<String>)(s1, s2) -> { return s1 + s2; });
```

5.1.31 Answer: B

Reason :

Class Q correctly overrides the compute(String) method of P class and class R correctly overrides the compute(String) method of Q class. Keyword super is used to invoke the method of parent class.

Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {} `
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".

Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
    (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence, `java Test.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Test.java
java -cp <memory> Test
```

Hence, in this case given command executes main(String []) method.
At Line n1, reference variable 'p' refers to an instance of class R, hence p.compute("Go") invokes the compute(String) method of R class.
return super.compute(str.replace('o', 'O')); => return super.compute("Go".replace('o', 'O')); => return super.compute("GO");

It invokes the compute(String) method of Parent class, which is Q.
=> return super.compute(str.toLowerCase()); => return super.compute("GO".toLowerCase()); => return super.compute("go");

It invokes the compute(String) method of Parent class, which is P.
=> return str.repeat(3); => return "go".repeat(3); => return "gogogo";

Control goes back to compute(String) method of Q and to the compute(String) method of R, which returns "gogogo".
Line n2 prints gogogo on to the console.

5.1.32 Answer: C

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

As the name suggests, Local variable Type inference is applicable only for local variables and not for instance or class variables. Hence, `static var arr = new Boolean[1];` causes compilation error.

5.1.33 Answer: C

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.

If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
      (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java University.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java University.java` is equivalent to (but not exactly same as):

```
javac -d <memory> University.java
java -cp <memory> com.udayankhattry.ocp1.Teacher
```

Hence in this case, output is: Teacher.

Please note that source-file can contain package statement and multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

5.1.34 Answer: C

Reason :

while loop doesn't have curly bracket over here, so only System.out.println(x) belongs to while loop.

Above syntax can be written as follows:

```
int x = 5;
while (x < 10) {
    System.out.println(x);
}
x++;
```

As x++; is outside loop, hence value of x is always 5 within loop, 5 < 10 is true for all the iterations and hence infinite loop.

5.1.35 Answer: E

Reason :

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
(to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries.
For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

As given question doesn't say anything about setting the 'Main-Class' attribute, hence class name must be passed after the module name in -m option.
Correct -m option will be: -m com.db.connect/com.jdbc.utils.Connect

Correct -p option will be: -p third-party\jars or -p third-party\jars\conncetivity.jar

Let's check all the options one by one :
java -p third-party -m com.db.connect/com.jdbc.utils.Connect ✗ '-p third-party' is not valid, it should be '-p third-party\jars' or '-p third-party\jars\conncetivity.jar'.

java -p third-party\jars -m com.db.connect ✗ '-m com.db.connect' is not valid, it should be '-m com.db.connect/com.jdbc.utils.Connect'.

java -p third-party* -m com.db.connect/com.jdbc.utils.Connect ✗ '*' is not allowed in -p option.

java -p third-party\jars\conncetivity.jar -m com.db.connect ✗ '-m com.db.connect' is not valid, it should be '-m com.db.connect/com.jdbc.utils.Connect'.

java -p third-party\jars -m com.db.connect/com.jdbc.utils.Connect ✓ Both -p and -m options are correct.

java -m com.db.connect/com.jdbc.utils.Connect -p third-party\jars ✗ -m should be the last option.

5.1.36 Answer: B

Reason :
Logic in secret_one.jar is to take the source string "ADTETFAECNKD" and print the alternate characters starting at index 0, hence it prints ATTACK
and
Logic in secret_two.jar is to take the source string "ADTETFAECNKD" and print the alternate characters starting at index 1, hence it prints DEFEND

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
(to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries .
For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory,

exploded module 'singlemodule' and modular jar 'connector.jar'.

When multiple modules with the same name are in different jar files on the module path, first module is selected and rest of the modules with same name are ignored. As first modular jar file in module path is 'secret_two.jar', hence com.udayankhattry.ocp1.Secret class defined in secret_two.jar file executes, which prints DEFEND on to the console.

5.1.37 Answer: D

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

At Line n1, x infers to double type as 7.85 is double literal.
At Line n2, y infers to float type as 5.25f is a float literal.

Line n3:
var a = (int)x + (int)y;
var a = (int)7.85 + (int)5.25;
var a = 7 + 5;
var a = 12; //a infers to int type

Line n4:
var b = (int)(x + y);
var b = (int)(7.85 + 5.25);
var b = (int)(13.10);
var b = 13; //b infers to int type

Hence, `System.out.println(a + b);` prints 25 on to the console.

5.1.38 Answer: C

Reason :
sb --> {"HavePatience"}

sb.delete(4, 5) --> {"Haveatience"}
sb.insert(4, " P") -> {"Have Patience"}
sb.toString() -> Creates a new String object "Have Patience"
"Have Patience".toUpperCase() -> Creates another String object "HAVE PATIENCE" but the String object is not referred and used.

Method invocation on sb modifies the same object, so after insert(4, " P") method invocation, 'sb' refers to {"Have Patience"}.

StringBuilder class overrides toString() method, which returns the contents of StringBuilder object in the form of String. `System.out.println(sb);` invokes the sb.toString() method, which returns "Have Patience" and the same is printed on to the console.

5.1.39 Answer: D

Reason :
Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor
2. public Throwable(String message) {...} : Pass the detail message
3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause
4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)
1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)
2. public String toString() {} :
Returns a short description of this throwable. The result is the concatenation of:
the name of the class of this object
": " (a colon and a space)
the result of invoking this object's getLocalizedMessage() method

If getLocalizedMessage() returns null, then just the class name is returned .

In multi-catch statement, classes with multi-level hierarchical relationship can't be used.
RuntimeException is subclass of Exception, IllegalArgumentException is indirect subclass of Exception and IllegalArgumentException is subclass of RuntimeException, hence these pairs can't be used in multi-catch statement.

Only one option is left to replace Line 14 with 'catch(RuntimeException e) {'.

Commenting out Line 14, Line 15 and Line 16 will resolve the compilation error but it will print the whole stack trace rather than just printing the message.

5.1.40 Answer: C

Reason :
If choice is between implicit casting and variable arguments, then implicit casting takes precedence because variable arguments syntax was added in Java 5 version.
m('A'); is tagged to m(int) as 'A' is char literal and implicitly casted to int.
m('A', 'B'); is tagged to m(int, int) as 'A' and 'B' are char literals and implicitly casted to int.
m('A', 'B', 'C'); is tagged to m(char...)
m('A', 'B', 'C', 'D'); is tagged to m(char...)

There is no compilation error and on execution output is: 1233

5.1.41 Answer: E

Reason :
super(); is added by the compiler as the first statement in both the constructors:

```
Super(int i) {
    super();
    System.out.println(100);
}
```

and

```
Sub() {
    super();
    System.out.println(200);
}
```

Class Super extends from Object class and Object class has no-argument constructor, hence no issues with the constructor of Super class.

But no-argument constructor is not available in Super class, hence calling super(); from Sub class's constructor causes compilation error.

5.1.42 Answer: A

Reason :
Before you answer this, you must know that there are 5 different Employee object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Employee objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method has been overridden by the Employee class and equates the object based on their name and age.

3 matching Employee objects are found in the list and first matching list element is removed from the list. Remaining 3 list elements are printed in the insertion order.

5.1.43 Answer: A

Reason :
Cat class doesn't override the jump() method of Animal class, in fact jump(int) method is overloaded in Cat class.
Deer class overrides jump() method of Animal class.

Reference variable 'cat' is of Animal type, cat.jump() syntax is fine and as Cat doesn't override jump() method hence Animal version is invoked, which prints ANIMAL on to the console.

Even though reference variable 'deer' is of Animal type but at runtime `deer.jump();` invokes overriding method of Deer class, this prints DEER on to the console.

5.1.44 Answer: B

Reason :
Three StringBuilder objects [{"Sapphire"}, {"Emerald"}, {"Ruby"}] are added to the gemStones list .
StringBuilder class doesn't override equals(Object) method and hence `days.contains(new StringBuilder("Sapphire"))` returns false. Code inside if-block is not executed and days.size() returns 3.

5.1.45 Answer: B

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

At Line n1, list refers to an ArrayList of Object type, because Generic type is not defined on the right side.
Three String objects are added to the list.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerExceptio n
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Now if you look at the 2nd overloaded method, 2nd parameter is: 'Iterable<? extends CharSequence>', which means if you are passing List, then List<String>, List<StringBuilder> & List<StringBuffer> are possible arguments.
But raw List and List<Object> won't work. As ArrayList instance created at Line n1 is of Object type, hence Line n2 causes compilation error.

If you change Line n1 to: List<String> list = new ArrayList<>();
then output will be: TAKE.THE.RISK

5.1.46 Answer: C

Reason :
According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:
1. May not declare to throw any checked exception.
2. May declare to throw the same checked exception thrown by super class / interface method.

3. May declare to throw the sub class of the exception thrown by super class / interface method.
4. Cannot declare to throw the super class of the exception thrown by super class / interface method.
5. Cannot declare to throw unrelated checked exception.
6. May declare to throw any RuntimeException or Error.

default methods were added in Java 8 and FilePrinter class correctly overrides the default method print() of Printer interface. Line n1 compiles successfully.

At Line n2, 'p' is of Printer type (supertype) and it refers to an instance of FilePrinter class (subtype), this is polymorphism and allowed in Java. Line n2 compiles successfully.

At Line n3, method print() is invoked on 'p' reference (Printer type) and as print() method defined in Printer interface declares to throw FileNotFoundException (checked exception). As main(String []) method doesn't declare to throw FileNotFoundException and also there is not try-catch block available, therefore Line n3 causes compilation error.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

At Line n4, 'fp' infers to FilePrinter type.

At Line n5, method print() is invoked on 'fp' reference (FilePrinter type) and as print() method defined in FilePrinter class doesn't declare to throw any checked exception, hence Line n5 compiles successfully.

Only Line n3 causes compilation failure.

5.1.47 Answer: C,F

Reason :
If package is used, then it should be the first statement. But javadoc and developer comments are not considered as java statements, so a class

can have developer and javadoc comments before the package statement.
If import and package both are available, then correct order is package, import, class declaration.
Multiple package statements are not allowed.

5.1.48 Answer: E

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name, package name or loop's label but it cannot be used as a class or interface name.

At Line n1, variable 'var' infers to int type. Line n1 compiles successfully.
At Line n2, 'var' is used as for loop's label, no issues at Line n2 as well.
At Line n3, continue is used with label. break and continue can be used with labels, so no issue at line n3 as well.
As var infers to int type, hence at Line n4, `var++;` is a valid statement.
Given code compiles successfully.

5.1.49 Answer: A,C,E

Reason :
You can declare one-dimensional array either by using `short arr []` or `short [] arr`.

You can either create and assign array object in the same statement or two statements. `short arr [] = new short[2];` and `short [] arr; arr = new short[2];` both are correct.

Array size cannot be specified at the time of declaration, so `short [2] arr;` causes compilation error.

`short [] arr = {};` => arr refers to a short array object of 0 size. so arr[0] = 5; and arr[1] = 10; will throw

ArrayIndexOutOfBoundsException at runtime.

`short [] arr = new short[]{100, 100};` => arr refers to a short array object of size 2 and both array elements have value 100. arr[0] = 5; replaces 1st element value with 5 and arr[1] = 10; replaces 2nd element value with 10. So this is also a correct option.

`short [] arr = new short[2]{100, 100};` => Array's size can't be specified, if you use {} to assign values to array elements. Hence, this statement causes compilation error.

5.1.50 Answer: C

Reason :
print() is static method of class Test. So correct syntax to invoke method print() is Test.print();
but static methods can also be invoked using reference variable: obj.print(); Warning is displayed in this case.
Even though obj has null value, we don't get NullPointerException as objects are not needed to invoke static methods.

5.1.51 Answer: D

Reason :
Please note that Strings computed by concatenation at compile time are referred by String Pool. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc. This means the result of constant expression is calculated at compile time and later referred by String Pool.
Where as Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

'fName' is a constant variable and 'lName' is a non-constant variable.

`fName + lName` is not a constant expression and hence the expression is computed at run-time and the resultant String object "JamesGosling" is not referred by String Pool.
As 'fName' is constant variable and "Gosling" is String literal, hence the expression `fName + "Gosling"` is a constant expression, therefore expression is computed at compile-time and resultant String object "JamesGosling" is referred by String Pool.
So, 'name1' and 'name2' refer to different String object and that is why

`name1 == name2` returns false.

`"James" + "Gosling"` is also a constant expression and hence the expression is computed at compile-time and resultant String object "JamesGosling" also refers to the same String object created by 2nd expression.
So, 'name2' and 'name3' refer to same String object and that is why `name2 == name3` returns true.

5.1.52 Answer: E

Reason :
Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`
It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,
"A".repeat(3); returns "AAA".

Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
    (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence `java Test.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Test.java
java -cp <memory> com.udayankhattry.ocp1.Test
```

Please note that source-file can contain multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

`java --source 10 Test.java` instructs the java command to process the Test.java file as Java 10 source code. As repeat(int) method was added in Java 11, hence this command causes error.

If you use the command 'java --source 11 Test.java' or 'java Test.java', then false will be printed on to the console. repeat(count) method uses constructor of String class to construct the new Object and hence newly created String object is a non-pool object. "++" is a String literal, so it is a pool object. That is why `str.repeat(2) == "++"` would evaluate to false.

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

5.1.53 Answer: D

Reason :
In this case "if - else if" block is used and not "if - else" block.

90000 is assigned to variable 'price' but you can assign parameter value or call some method returning double value, such as:

`double price = currentTemp();`.

In these cases, compiler will not know the exact value until runtime, hence Java Compiler is not sure which boolean expression will be evaluated to true and so variable model may not be initialized.

Usage of LOCAL variable, 'model' without initialization causes compilation error. Hence, `System.out.println(model);` statement causes compilation error.

5.1.54 Answer: A

Reason :
Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods, private methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].
Interface ILog specifies two non-overriding abstract methods, hence lambda expression causes compilation error.

5.1.55 Answer: C

Reason :

Encapsulation is all about having private instance variable and providing public getter and setter methods.
Out of the given 4 options, below option provides a well encapsulated Clock class:

```
public class Clock {
    private String model;
    public String getModel() { return model; }
    public void setModel(String val) { model = val; }
}
```

5.1.56 Answer: E

Reason :
Implied readability is to specify a transitive dependency on another module such that other modules reading your module also read that dependency.
For example, if module C has `requires B;` (C reads B) and module B has `requires A;` (B reads A), then C doesn't read A.
To have the transitive readability, 'requires transitive' directive is used.
If module C has `requires B;` (C reads B) and module B has `requires transitive A;` (B reads A [transitive dependency]), then C will also read A.

5.1.57 Answer: D,E

Reason :
Line n8's syntax was added in JDK 5 and it compiles without any warnings.

Line n9's syntax was added in JDK 7, in which type parameter can be ignored from right side of the statement, it is inferred from left side, so Line n9 also compiles without any warning.

Type parameter can't be removed from declaration part, hence Line n7 causes compilation error.

Both Line n5 and Line n6 are mixing Generic type with Raw type and hence warning is given by the compiler.

5.1.58 Answer: D

Reason :
Both Runtime.getRuntime().gc(); and System.gc(); do the same thing,

these make a request to JVM to run Garbage Collector.
JVM makes the best effort to run Garbage Collector but nothing is guaranteed.

Setting the reference variable to null will make the object, eligible for Garbage Collection, if there are no other references to this object. But this doesn't force JVM to run the Garbage Collector. Garbage Collection cannot be forced.

5.1.59 Answer: B

Reason :
Range of char data type is from 0 to 65535 and hence it can be easily assigned to int type. println() method is overloaded to accept char type and int type both. If char type value is passed, it prints char value and if int type value is passed, it prints int value .
As i1 is of int type, hence corresponding int value, which is 97, is printed on to the console.

5.1.60 Answer: D

Reason :
Matching case block 'case 10:' is found, a *= 2; is executed, which means a = a * 2; => a = 5 * 2; => a = 10;
No break statement, hence it enters in fall-through.
a *= 3; is executed, which means a = a * 3; => a = 10 * 3; => a = 30;
a *= 4; is executed, which means a = a * 4; => a = 30 * 4; => a = 120;

5.1.61 Answer: G

Reason :
for is a keyword and hence can't be used as a label.
Java labels follow the identifier naming rules and one rule is that we can't use java keywords as identifier. Hence, Compilation error.

5.1.62 Answer: D

Reason :
If there is only one statement in the right-hand side of lambda expression, then semicolon inside the body, curly brackets and return keyword can be removed. But all 3 [return, {}, ;] must either be available or must be absent.

Based on above statement, `s -> {return true;}` is the correct option.

5.1.63 Answer: E

Reason :

No syntax error in the given code.
Initially, i = 1, j = 5 and k = 0.
1st iteration of A: i = 2.
    1st iteration of B: j = 4.
        1st iteration of C: k = k + i + j = 0 + 2 + 4 = 6. `i == j` evaluates to false and `i > j` also evaluates to false, hence else block gets executed. `continue B` takes the control to the loop B.
    2nd iteration of B: j = 3.
        1st iteration of C: k = k + i + j = 6 + 2 + 3 = 11. `i == j` evaluates to false and `i > j` also evaluates to false, hence else block gets executed. `continue B` takes the control to the loop B.
    3rd iteration of B: j = 2 .
        1st iteration of C: k = k + i + j = 11 + 2 + 2 = 15. `i == j` evaluates to true, control breaks out of the loop A.

`System.out.println(k);` prints 15 on to the console.

5.1.64 Answer: A

Reason :

ArrayList instance referred by 'list' stores 3 StringBuilder instances.

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.
StringBuilder class doesn't override equals(Object) method. So Object version is invoked which uses == operator, hence `sb.equals(new StringBuilder("AAA"))` would return false as all 4 StringBuilder instances have been created at four different memory locations.

None of the StringBuilder instances are removed from the list.

StringBuilder class overrides toString() method, which returns the containing String and that is why [AAA, BBB, AAA] will be printed on to the console.

5.1.65 Answer: B

Reason :

arr1 is of String[] type, where as arr2 and arr3 are of String type. As all three arr1, arr2 and arr3 are of reference type, hence null can be assigned to all these variables. Line n1 compiles successfully.

Statement at Line n2: arr2 = arr3 = arr1;
=> arr2 = (arr3 = arr1); //assignment operator is right to left associative.
arr3 is of String type and arr1 is of String [] type, hence (arr3 = arr1) causes compilation error.

Though you had to select one correct option, hence no need to look further but I am providing explanation for Line n3 as well.
Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C "
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Based on above points, `String.join("-", arr2)` compiles successfully as arr2 is of String type.

5.1.66 Answer: D

Reason :
java.io.FileNotFoundException exception is a checked exception.

Method declaring checked exception in its throws clause doesn't mean that it should have code to actually throw that type of Exceptions. So even though read() method of Class1 declares to throw FileNotFoundException but its body doesn't actually throw an instance of FileNotFoundException.

Variable and method name can be same as class name, so code of Class2 is also valid. Remember: Though you don't get any compilation error but it is not recommended to use the Class name for variable and method names .

LOCAL variable can be declared with final modifier only. msg variable inside print() method of Class3 is declared private and this causes compilation error.

5.1.67 Answer: A,B,C

Reason :
Javadoc of trim() method states that it returns a string whose value is this string, with all leading and trailing space removed, where space is defined as any character whose codepoint is less than or equal to 'U+0020' or '\u0020' (the space character).
As the developer comment in above code states that 'text' refers to String object containing multiple space characters, hence text.trim()

returns "BE YOURSELF!" after removing all leading and trailing space characters. Hence trim() is correct option.

stripLeading(), stripTrailing() and strip() were added in Java 11.
stripLeading() returns a string whose value is this string, with all leading white space removed.
stripTrailing() returns a string whose value is this string, with all trailing white space removed.
strip() returns a string whose value is this string, with all leading and trailing white space removed.

Point to note here is that trim() method works with space, where space is defined as any character whose codepoint is less than or equal to 'U+0020' or '\u0020' (the space character) but strip(), stripLeading() and stripTrailing() works with white space.
To find out what is white space character in Java, check Character.isWhitespace(int) method, you will find below:

A character is a Java whitespace character if and only if it satisfies one of the following criteria:
It is a Unicode space character (SPACE_SEPARATOR, LINE_SEPARATOR, or PARAGRAPH_SEPARATOR) but is not also a non-breaking space ('\u00A0', '\u2007', '\u202F').
It is '\t', U+0009 HORIZONTAL TABULATION.
It is '\n', U+000A LINE FEED.
It is '\u000B', U+000B VERTICAL TABULATION.
It is '\f', U+000C FORM FEED.
It is '\r', U+000D CARRIAGE RETURN.
It is '\u001C', U+001C FILE SEPARATOR.
It is '\u001D', U+001D GROUP SEPARATOR.
It is '\u001E', U+001E RECORD SEPARATOR.
It is '\u001F', U+001F UNIT SEPARATOR.

White space character in java includes space character along with above characters .
Hence, text.strip() will successfully remove all the leading and trailing white spaces and will return "BE YOURSELF!".

text.stripLeading().stripTrailing() will first remove leading white space and then trailing white space and will return "BE YOURSELF!".

String class doesn't have methods with the names: ltrim(), rtrim(), trimLeading(), trimTrailing() and trimBoth(), hence these will cause compilation error.

5.1.68 Answer: B

Reason :
Initially sb refers to {"INHALE "}
Given Statement:
String s = sb.toString() + (sb.append("EXHALE "));
String s = "INHALE " + (sb.append("EXHALE ")); //Left operand of + operator is evaluated first, sb --> {"INHALE "}
String s = "INHALE " + {"INHALE EXHALE "}; //Right operand of + operator is evaluated next, sb --> {"INHALE EXHALE "}
String s = "INHALE " + "INHALE EXHALE "; //As left operand is of String type, so + operator behaves as concatenation operator and that is why toString() method on right operand (which is StringBuilder object referred by 'sb') is invoked.
String s = "INHALE INHALE EXHALE "; //String object referred by 's' has 21 characters.

strip() method of String class (available since Java 11) returns a string whose value is this string, with all leading and trailing white space removed.
To find out what is white space character in Java, check Character.isWhitespace(int) method, you will find below:
A character is a Java whitespace character if and only if it satisfies one of the following criteria:
It is a Unicode space character (SPACE_SEPARATOR, LINE_SEPARATOR, or PARAGRAPH_SEPARATOR) but is not also a non-breaking space ('\u00A0', '\u2007', '\u202F').
It is '\t', U+0009 HORIZONTAL TABULATION.
It is '\n', U+000A LINE FEED.
It is '\u000B', U+000B VERTICAL TABULATION.
It is '\f', U+000C FORM FEED.
It is '\r', U+000D CARRIAGE RETURN.
It is '\u001C', U+001C FILE SEPARATOR.
It is '\u001D', U+001D GROUP SEPARATOR.
It is '\u001E', U+001E RECORD SEPARATOR.
It is '\u001F', U+001F UNIT SEPARATOR.

White space character in java includes space character along with above characters .

As s refers to "INHALE INHALE EXHALE " [contains 21 characters], hence s.strip() creates a new String object by trimming trailing space character: "INHALE INHALE EXHALE" [contains 20 characters].

Hence, `System.out.println(s.strip().length());` prints 20 on to the console.

5.1.69 Answer: C

Reason :
ArrayList's 1st and 3rd items are referring to same StringBuilder instance referred by sb [sb --> {Hello}] and 2nd item is referring to another instance of StringBuilder.

sb.append("World!"); means sb --> {HelloWorld!}, which means 1st and 3rd items of ArrayList now refers to StringBuilder instance containing HelloWorld!

In the output, [HelloWorld!, Hello, HelloWorld!] is printed.

5.1.70 Answer: C

Reason :
substring(int beginIndex, int endIndex) method of String class extracts the substring, which begins at the specified beginIndex and extends to the character at index endIndex - 1.
This method throws IndexOutOfBoundsException if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex. e.g.
"freeway".substring(4, 7) returns "way"
"freeway".substring(4, 8) throws IndexOutOfBoundsException

substring(int beginIndex) method of String class extracts the substring, which begins with the character at the specified index and extends to the end of this string.
This method throws IndexOutOfBoundsException if beginIndex is negative or larger than the length of this String object. e.g.
"freeway".substring(4) returns "way"
"freeway".substring(8) throws IndexOutOfBoundsException

replace(CharSequence target, CharSequence replacement) method of String class returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.
"Java".replace("a", "A") --> returns new String object "JAvA".

Let's check all the given options :
"REBUS".substring(2); [begin = 2, end = 4 (end of the string)], returns "BUS" and hence it is a correct option.
"REBUS".substring(2, 4); [begin = 2, end = 3 (endIndex - 1)], returns "BU" and hence it is incorrect option.
"REBUS".substring(2, 5); [begin = 2, end = 4 (endIndex - 1)], returns "BUS" and hence it is a correct option.
"REBUS".replace("RE", ""); It replaces "RE" with empty string "" and returns "BUS", so it is also a correct option.
"REBUS".substring(2, 6); Length of "REBUS" = 5 and endIndex = 6, which is greater than 5, hence it will thrown IndexOutOfBoundsException at runtime. Incorrect option
"REBUS".delete(0, 2); Compilation error as delete(...) method is not available in String class, it is part of StringBuilder class. Incorrect option.

So, total 3 options will replace /*INSERT*/ to print BUS on to the console.

5.1.71 Answer: C

Reason :
super(); inside Parent(int) constructor invokes the no-argument constructor of Object class and hence no compilation error in Parent(int) constructor.

super(0); inside Child(int) constructor invokes Parent(int) constructor, which is available and hence no issues.

Child(int, int) constructor has both super(i) and this(j) statements. A constructor should have super(...) or this(...) but not both. Hence Child(int, int) causes compilation failure.

As all the classes are defined in Test.java file under 'com.udayankhattry.ocp1' package, hence child.i and child.j don't cause compilation error. i and j are declared with package scope.

5.1.72 Answer: B

Reason :
list.add(0, "I"); means list --> [I],
list.set(0, "CAN"); means replace the current element at index 0 with the passed element "CAN". So after this operation, list --> [CAN].
[CAN] is printed on to the console.

5.1.73 Answer: C

Reason :
`char [] arr1 [] = new char[5][];`: Creates two-dimensional array object, whose 1st dimension is 5 but 2nd dimension is not yet defined. On the left side array symbols can be used before the reference variable or after the reference variable or can be mixed, hence `char [][] arr1`, `char [] arr1 []` and `char arr1[][]` all are valid. This statement compiles successfully.

`byte [] val = new byte[10];`: Creates one-dimensional byte array object and val refers to it. All the array elements are initialized to 0. This statement compiles without any error.

`int [] arr2 = {1, 2, 3, 4, 5};`: This syntax creates one-dimensional int array object of 5 elements and initializes these 5 elements as well. It initializes element at 0th index to 1, element at 1st index to 2, element at 2nd index to 3, element at 3rd index to 4 and element at 4th index to 5. 'arr2' refers to this one-dimensional array object. This statement also compiles fine.

`int [] arr3 = new int[3]{10, 20, 30};`: You can't specify size at the time of initializing with data, hence `new int[3]{10, 20, 30};` causes compilation error.
Correct syntax is: `int [] arr3 = new int[]{10, 20, 30};` OR `int [] arr3 = {10, 20, 30};`

5.1.74 Answer: E

Reason :
super.open(); => Using super keyword, you can access methods and variables of immediate parent class, hence if you replace /*INSERT*/ with `super.open();`, then open() method of Padlock class will be invoked.

super.super.open(); => super.super is not allowed in java, it causes compilation error.

((Lock)super).open(); => Not possible to cast super keyword in java, it causes compilation error.

(Lock)super.open(); => super.open(); will be evaluated first as dot (.) operator has higher precedence than cast. super.open(); returns void and hence it cannot be casted to Lock. It also causes compilation error.

In fact, it is not possible to directly reach to 2 levels, super keyword allows to access methods and variables of immediate parent class only (just 1 level up). Hence, correct answer is: 'None of the other options'

5.1.75 Answer: F

Reason :
print() method at Line n2 hides the method at Line n1. So, no compilation error at Line n2.

Reference variable 'b' is of type Base, so `(Derived) b` does not cause any compilation error. Moreover, at runtime it will not throw any ClassCastException as well because b is null. Had 'b' been referring to an instance of Base class [Base b = new Base();], `(Derived) b` would have thrown ClassCastException.

d.print(); doesn't cause any compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Derived.print(); is the preferred syntax.

d.print(); invokes the static print() method of Derived class and prints DERIVED on to the console.

5.1.76 Answer: C

Reason :
As there is no brackets after if, hence only one statement is part of if block and other is outside.
Above code can be written as below:

```
if(grade >= 60) {
    System.out.println("Congratulations");
}
```

```
System.out.println("You passed");
else
    System.out.println("You failed");
```

There should not be anything in between if-else block but in this case, System.out.println("You passed"); is in between if-else and thus Compilation error.

5.1.77 Answer: D

Reason :
This is bit tricky. Just remember this:
Two instances of following wrapper objects, created through auto-boxing will always be same, if their primitive values are same:
Boolean,
Byte ,
Character from \u0000 to \u007f (7f equals to 127),
Short and Integer from -128 to 127.

For 1st statement, list.add(27); => Auto-boxing creates an integer object for 27.
For 2nd statement, list.add(27); => Java compiler finds that there is already an Integer object in the memory with value 27, so it uses the same object.

That is why `System.out.println(list.get(0) == list.get(1));` returns true.

For 3rd statement, list.add(227); => Auto-boxing creates an integer object for 227.
For 4th statement, list.add(227); => As 227 is greater than 127, hence auto-boxing creates another integer object for 227.

As both the objects are created at different memory locations, hence `System.out.println(list.get(2) == list.get(3));` returns false.

5.1.78 Answer: F

Reason :
Line n1 creates a String [] object of 4 elements and arr refers to this array object. arr[0] = "L", arr[1] = "I", arr[2] = "V" and arr[3] = "E".
i = -2.
Boolean expression of Line n2: i++ == -1

=> (i++) == -1 //As Post-increment operator ++ has higher precedence over ==
=> -2 == -1 //i = -1, value of i is used in the expression and then incremented.
=> false and hence Line n3 is not executed.
But there is no issue with Line n3 and it compiles successfully.

Boolean expression of Line n4 is evaluated next:
--i == -2 //i = -1
=> (--i) == -2 //As Pre-decrement operator -- has higher precedence over ==
=> -2 == -2 //i = -2, value of i is decremented first and then used in the expression.
=> true and hence Line n5 is executed next.

Line n5:
arr[-++i] = "O"; //i = -2
=> arr[-(++i)] = "O"; //Unary minus '-' and pre-increment '++' operators have same precedence
=> arr[-(-1)] = "O"; //i = -1, value of i is incremented first and then used in the expression.
=> arr[1] = "O"; //2nd array element is changed to "O".
Hence after Line n5, arr refers to {"L", "O", "V", "E"}

Static overloaded method join(...) was added in JDK 1.8 and has below declarations :
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null"
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Hence, `System.out.println(String.join("", arr));` prints LOVE on to the console.

5.1.79 Answer: C,E,F

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to in t

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

But there are certain restrictions on using var type:
Declaration and initialization should be in one statement, hence `var num; num = 10;` won't compile.

var is not allowed as an element type of an array, hence `var [] arr1 = new String[2];` won't compile.

Explicit target-type is needed for the array initializer, if you use var type. Hence, `var arr2 = {1, 2, 3};` won't compile.
If you provide the target-type, `var arr2 = new int[]{1, 2, 3};`, then there is no compilation error and in this case var infers to int [] type.

If initializer is of null type, then it is not possible to infer the target type, hence `var msg = null;` won't compile.

Also note that, var type cannot be the target type of lambda expressions and method references.

`final var str = "Hello";` will compile successfully as 'str' infers to String type and final modifier is allowed for local variables.
`private var y = 100;` will not compile as private modifier is not allowed for local variables.

var can easily be used for enhanced for-loop indexes, and index variables declared in traditional for loops, hence both the loop code segments will compile successfully.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

5.1.80 Answer: B

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to in t

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

var type cannot be used as method parameters or method return type.

At Line n1, arr infers to int[] type.

It is clear from Line n2 that, method name should be process, it should be static method, it should accept 3 parameters (int[], int, int) and its return type must be String. Hence below definition is correct:

```
private static String process(int [] arr, int start, int end) {
    return null;
}
```

6 Practice Test-6

6.1 80 Questions covering all topics.

6.1.1 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
         short [] args = new short []{ 50 , 50 };
        args[ 0 ] = 5 ;
        args[ 1 ] = 10 ;
        System. out .println( "[" + args[ 0 ] + ", " + args[ 1 ] + "]" );
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. [50, 50]

D. [5, 10]

6.1.2 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        outer: for ( var i = 0 ; i < 3 ;
              System. out .print(i)) {
            i++;
            inner: for ( var j = 0 ; j < 3 ;
              System. out .print(j)) {
               if (i > ++j) {
                  break outer;
               }
            }
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Program terminates successfully but nothing is printed on to the console

C. Program terminates successfully after printing 1 on to the console

D. Program terminates successfully after printing 12 on to the console

E. Program terminates successfully after printing 123 on to the console

F. Program terminates successfully after printing 1231 on to the console

G. Program terminates successfully after printing 121 on to the console

H. Program terminates successfully after printing 0120 on to the console

6.1.3 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       StringBuilder sb =
              new StringBuilder( "B" ); //Line n1
        sb.append(sb.append( "A" )); //Line n2
        System. out .println(sb); //Line n3
    }
}
```

What will be the result of compiling and executing Test class?

A. B

B. BA

C. AB

D. BAB

E. ABA

F. ABAB

G. BABA

H. ABBA

I. Compilation error at Line n2

6.1.4 ______________ system module defines the foundational APIs of the Java SE Platform.

A. java.se

B. java.base

C. jdk.se

D. jdk.base

E. java

F. se

6.1.5 What will be the result of compiling and executing Test class?

```
//Test.java
package com.udayankhattry.ocp1;

class Point {
    static int x ;
    int y ;
}

public class Test {
    public static void main(String[] args) {
        Point p1 = new Point();
        Point p2 = new Point();
        p1. x = 17 ;
        p1. y = 35 ;
        p2. x = 19 ;
        p2. y = 40 ;

            System. out .println(p1. x + ":" + p1. y +
                ":" + p2. x + ":" + p2. y );
    }
}
```

A. 17:35:19:40

B. 19:35:19:40

C. 17:35:17:40

D. 17:35:19:35

E. 17:40:19:40

F. Compilation error

6.1.6 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;
import java.io.IOException;

abstract class Parent {
    public abstract void find() throws IOException;
}

class Child extends Parent {
    @Override
    public void find() throws IOException {
        throw new FileNotFoundException();
    }
}

public class Test {
    public static void main(String[] args) {
        Parent p = new Child();
        try {
            p.find();
        } catch (FileNotFoundException e) {
            System. out .print( "X" );
        } catch (IOException e) {
            System. out .print( "Y" );
        } finally {
            System. out .print( "Z" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. XZ

B. YZ

C. XYZ

D. Compilation Error

6.1.7 For the class Test, which options, if used to replace /*INSERT*/, will print "Lucky no. 7" on to the console? Select ALL that apply.

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        /*INSERT*/
        switch (var) {
            case 7 :
                System. out .println( "Lucky no. 7" );
                 break ;
            default :
                System. out .println( "DEFAULT" );
        }
    }
}
```

A. char var = '7';

B. char var = 7;

C. Integer var = 7;

D. Character var = '7';

E. Character var = 7;

6.1.8 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        System. out .println( "Result is: " + ( 10 != 5 ));
    }
}
```

What will be the result of compiling and executing Test class?

A. Result is: true

B. Result is: false

C. Compilation error

D. Result is: (10 != 5)

6.1.9 Consider below 4 code snippets:

A.	String [] arr1 = { **null** , **null** };
	System. ***out*** .println(**"1. "** + String. *join* (**"::"** , arr1));

B.	String [] arr2 = {};
	System. ***out*** .println(**"2. "** + String. *join* (**"-"** , arr2));

C.	String [] arr3 = **null** ; System. ***out*** .println(**"3. "** + String. *join* (**"."** , arr3));
D.	System. ***out*** .println(**"4. "** + String.join(**"."** , **null**));

Which of the following statements are correct about above snippets? Select ALL that apply.

A. Snippet 1 compiles successfully

B. Snippet 2 compiles successfully

C. Snippet 3 compiles successfully

D. Snippet 4 compiles successfully

E. Snippet 1 throws runtime exception

F. Snippet 2 throws runtime exception

G. Snippet 3 throws runtime exception

H. Snippet 4 throws runtime exception

6.1.10 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
        String res = "" ;
        loop: for ( var i = 1 ; i <= 5 ; i++) { //Line n1
            switch (i) {
                case 1 :
                    res += "UP " ;
                    break ;
                case 2 :
                    res += "TO " ;
                    break ;
                case 3 :
                    break ;
                case 4 :
                    res += "DATE" ;
                    break loop;
            }
        }
        System. out .println(String. join ( "-" ,
                    res.split( " " ))); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. UP

B. TO

C. DATE

D. UP-TO

E. TO-DATE

F. UP-TO-DATE

G. Compilation error at Line n1

H. Compilation error at Line n2

6.1.11 Which of the following options correctly import the Panda class from com.singaporezoo.animal package? Select 2 options.

A. Import com.singaporezoo.animal.Panda;

B. import com.singaporezoo.animal;

C. import com.singaporezoo.animal.*;

D. import com.singaporezoo.*;

E. import com.singaporezoo.animal.Panda;

6.1.12 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Paper {
    static String getType() { //Line n1
        return "GENERIC" ;
    }
}

class RuledPaper extends Paper {
    String getType() { //Line n2
        return "RULED" ;
    }
}

public class Test {
    public static void main(String[] args) {
        Paper paper = new RuledPaper(); //Line n3
        System. out .println(paper. getType ()); //Line n4
    }
}
```

Which of the following statements is true for above code?

A. Compilation error in RuledPaper class

B. Compilation error in Test class

C. Code compiles successfully and on execution prints GENERIC on to the console

D. Code compiles successfully and on execution prints RULED on to the console

6.1.13 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.IOException;

class Super {
    Super() throws RuntimeException {
        System. out .print( "CARPE " );
    }
}

class Sub extends Super {
    Sub() throws IOException {
        System. out .print( "DIEM " );
    }
}

public class Test {
    public static void main(String[] args)
                throws Exception {
         new Sub();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in both Super and Sub classes

B. Compilation error only in Super class

C. Compilation error only in Sub class

D. Test class executes successfully and prints CARPE DIEM on to the console

E. Test class executes successfully and prints DIEM CARPE on to the console

6.1.14 Which of the following can be used as a constructor for the class given below?

```
public class Planet {

}
```

A. public void Planet(){}

B. public void Planet(int x){}

C. public Planet(String str) {}

D. None of the other options

6.1.15 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface X1 {
    default void print() {
      System. out .println( "X1" );
    }
}

interface X2 extends X1 {
    void print();
}

interface X3 extends X2 {
    default void print() {
      System. out .println( "X3" );
    }
}

class X implements X3 {}

public class Test {
    public static void main(String[] args) {
      X1 obj = new X();
      obj.print();
    }
}
```

Which of the following statements is correct?

A. interface X1 fails to compile

B. interface X2 fails to compile

C. interface X3 fails to compile

D. class X fails to compile

E. class Test fails to compile

F. class Test compiles successfully and on execution prints X1 on to the console

G. class Test compiles successfully and on execution prints X3 on to the console

6.1.16 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder( 5 );
        sb.append( "0123456789" );
        sb.delete( 8 , 1000 );
        System. out .println(sb);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. 01234567

D. 89

6.1.17 Consider below statements:

A.	int x = 5_____0;
B.	int y = _____50;
C.	int z = 50_____;
D.	float f = 123.76_86f;
E.	double d = 1_2_3_4;

How many statements are legal?

A. One statement only

B. Two statements only

C. Three statements only

D. Four statements only

E. All 5 statements

6.1.18 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Character> list = new ArrayList<>();
        list.add( 0 , 'V' );
        list.add( 'T' );
        list.add( 1 , 'E' );
        list.add( 3 , 'O' );

        if (list.contains( 'O' )) {
            list.remove( 3 );
        }

        for ( char ch : list) {
            System. out .print(ch);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Runtime error

C. VET

D. VTE

E. VTEO

F. VETO

6.1.19 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int score = 60 ;
        switch (score) {
            default :
               System. out .println( "Not a valid score" );
            case score < 70 :
               System. out .println( "Failed" );
                break ;
            case score >= 70 :
               System. out .println( "Passed" );
                break ;
        }
    }
}
```

A.	Compilation error	B.	Failed
C.	Not a valid score Failed	D.	Passed

6.1.20 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Calculator {
     int calculate( int i1, int i2) {
        return i1 + i2;
    }

     double calculate( byte b1, byte b2) {
        return b1 % b2;
    }
}

public class Test {
     public static void main(String[] args) {
        byte b = 100 ;
        int i = 20 ;
       System. out .println( new Calculator().calculate(b, i));
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. 120

D. 120.0

E. 5

F. 5.0

6.1.21 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String str = "Game on" ; //Line n1
       StringBuilder sb = new StringBuilder(str); //Line n2

         System. out .println(str.contentEquals(sb)); //Line n3
       System. out .println(sb.contentEquals(str)); //Line n4
       System. out .println(sb.equals(str)); //Line n5
       System. out .println(str.equals(sb)); //Line n6
    }
}
```

Which of the following statements is correct?

A. Only one statement causes compilation error

B. Two statements cause compilation error

C. Three statements cause compilation error

D. Four statements cause compilation error

E. No compilation error

6.1.22 Code of Test.java file below:

```
private class Test {
    public static void main(String... args) {
      System. out .println(args[ 1 ]);
   }
}
```

What will be the result of compiling and executing Test class using below commands:

```
javac Test.java
java Test laugh out loud
```

A. Compilation Error

B. laugh

C. out

D. loud

6.1.23 Consider below code of main.java file:

```
package main;

public class main {
    static String main = "ONE" ;

     public main() {
        System. out .println( "TWO" );
    }

     public static void main(String [] args) {
         main ();
    }

     public static void main() {
        System. out .println( main );
    }
}
```

Also consider below statements:

1. Code doesn't compile
2. Code compiles successfully
3. Only ONE will be printed to the console
4. Only TWO will be printed to the console
5. Both ONE and TWO will be printed to the console

How many of the above statements is/are true?

A. One statement

B. Two statements

C. Three statements

6.1.24 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
   boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
       String [] arr = { "*" , "**" , "***" , "****" , "*****" };
       Predicate pr1 = s -> s.length() < 4 ;
        print (arr, pr1);
    }

     private static void print(String [] arr,
              Predicate<String> predicate) {
        for (String str : arr) {
           if (predicate.test(str)) {
              System. out .println(str);
           }
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation error	B.	* ** ***
C.	* ** *** ****	D.	* ** *** **** *****

6.1.25 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
      String str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;
      System. out .println(subtext(str, 3 , 8 )); //Line n1
    }

    /*INSERT*/
}
```

Line n1 causes compilation error as subtext method is not found. Which of the following method definitions, if used to replace /*INSERT*/, will resolve the compilation error?
Select ALL that apply.

A.
```
private static int [] subtext(
                        String text, int start, int end) {
    return null ;
}
```

B.
```
private static String subtext(
                        String text, int start, int end) {
    return null ;
}
```

C.
```
private static int subtext(
                        String text, int start, int end) {
    return null ;
}
```

D.
```
private static String[] subtext(
                        String text, int start, int end) {
    return null ;
}
```

E.
```
private static var subtext(
                        String text, int start, int end) {
    return null ;
}
```

F.
```
private static String subtext(
                          var text, int start, int end) {
```

```
    return null ;
}
```

6.1.26 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class PenDrive {
    int capacity ;
    PenDrive( int capacity) {
        this . capacity = capacity;
    }
}

class OTG extends PenDrive {
    String type ;
    OTG( int capacity, String type) {
        // TODO
    }
}

public class Test {
    public static void main(String[] args) {
        OTG obj = new OTG( 64 , "TYPE-C" );
        System. out .println(obj. capacity + ":" + obj. type );
    }
}
```

Currently above code causes compilation error. Which of the option can successfully replace //TODO statement such that on executing Test class, output is 64:TYPE-C?

A.	super(capacity);	B.	super.capacity = capacity;
			this.type = type;

C.	this.type = type;
	super(capacity);
		D.	super(capacity);
			this.type = type;

6.1.27 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
       var: while ( true ) { //Line n1
           i: for ( int i = 0 ; i <= 2 ; i++) {
               if (i == 2 ) {
                   /*INSERT*/
               }
           }
       }
       System. out .println( "THINK DIFFERENT" ); //Line n2
    }
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print THINK DIFFERENT on to the console?

A. continue;

B. continue i;

C. continue var;

D. break;

E. break i;

F. break var;

6.1.28 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    private static void div() {
       System. out .println( 1 / 0 );
   }

     public static void main(String[] args) {
        try {
            div ();
       } finally {
           System. out .println( "FINALLY" );
       }
    }
}
```

What will be the result of compiling and executing Test class?

A. FINALLY is printed to the console and program ends normally

B. FINALLY is printed to the console, stack trace is printed and then program ends normally

C. FINALLY is printed to the console, stack trace is printed and then program ends abruptly

D. Compilation error

6.1.29 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        String s = new String( "Hello" );
        List<String> list = new ArrayList<>();
        list.add(s);
        list.add( new String( "Hello" ));
        list.add(s);
        s.replace( "l" , "L" );

            System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [Hello, Hello, Hello]

B. [HeLLo, Hello, Hello]

C. [HeLLo, Hello, HeLLo]

D. [HeLLo, HeLLo, HeLLo]

6.1.30 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        try {
            play ();
            return ;
        } catch (Exception ex) {
            System. out .println(ex.getMessage());
            return ;
        } finally {
            System. out .println( "MATCH ABANDONED" );
        }
        System. out .println( "DONE" );
    }

    static void play() throws Exception {
        throw new Exception( "INJURED" );
    }
}
```

What will be the result of compiling and executing Test class?

A.	INJURED MATCH ABANDONED	B.	INJURED MATCH ABANDONED DONE
C.	MATCH ABANDONED	D.	INJURED
E.	INJURED DONE	F.	MATCH ABANDONED DONE
G.	Compilation error		

6.1.31 For the given code:

```
package com.udayankhattry.ocp1;

abstract class Greetings {
    abstract void greet(String s);
}

public class Test {
    public static void main(String[] args) {
        /*INSERT*/
        obj.greet( "SEE THE GOOD IN OTHERS" );
    }
}
```

Which of the following statements can be used to replace /*INSERT*/ such that there are no compilation errors?

A.	Greetings obj = (String s) -> {System.out.println(s.toUpperCase());};
B.	Greetings obj = s -> {System.out.println(s.toUpperCase());};
C.	Greetings obj = s -> System.out.println(s.toUpperCase());
D.	Lambda expression cannot be used in this case

6.1.32 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add( 100 );
        list.add( 200 );
        list.add( 100 );
        list.add( 200 );
        list.remove( 100 );

          System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [200, 100, 200]

B. [100, 200, 200]

C. [200, 200]

D. [200]

E. Compilation error

F. Exception is thrown at runtime

6.1.33 Given below the directory/file structure on Windows platform:

```
C:
+---src
|   \---holidays
|           module-info.java
|
\---out
```

Contents of C:\src\holidays\module-info.java file:

```
import java.lang.*;

module holidays {
}
```

And the command executed from C:\

```
javac -d out --module-source-path src -m holidays
```

What is the result?

A. javac command fails as module-info.java file is not valid

B. javac command fails as there are no source code files other than module-info.java

C. javac command completes with warning

D. javac command completes without any warning

6.1.34 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java. util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
      List<String> list = new ArrayList<>( 4 );
      list.add( 0 , "MOVE" );
      list.add( 2 , "ON" );

        System. out .println(list);
    }
}
```

What will be the result of compiling and executing Test class?

A. [MOVE, ON]

B. [MOVE, null, ON, null]

C. [null, null, null, null]

D. []

E. Compilation error

F. An exception is thrown at runtime

6.1.35 Given code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
      System. out .println(args. length );
    }
}
```

What is the output if above program is run with the command line:
java Test.java --help

A. 0

B. 1

C. Above command causes error

D. None of the other options

6.1.36 Interface java.util.function.Consumer<T> declares below non-overriding abstract method:

```
   void accept(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.List;

class Division {
    int num ;
    int den ;
    int res ;

    Division( int num, int den) {
        this . num = num;
        this . den = den;
    }

    void divide() {
        try {
            res = num / den ;
        } catch (RuntimeException e) {}
    }

    public String toString() {
        if ( den == 0 )
            return num + "/" + den + " = " + "Infinity" ;
        else
            return num + "/" + den + " = " + res ;
    }
}

public class Test {
    public static void main(String[] args) {
        var list = List. of ( new Division( 100 , 4 ),
            new Division( 27 , 0 ), new Division( 25 , 5 ));
        list.forEach(d -> d.divide()); //Line n1
        list.forEach(d ->
                System. out .println(d)); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A.	100/4 = 25 27/0 = Infinity	B.	100/4 = 25 27/0 = 0

	25/5 = 5		25/5 = 5
C.	Compilation error	D.	On execution Test class throws an exception

6.1.37 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class X {
    void A() {
      System. out .print( "A" );
   }
}

class Y extends X {
    void A() {
      System. out .print( "A-" );
   }

     void B() {
      System. out .print( "B-" );
   }

     void C() {
      System. out .print( "C-" );
   }
}

public class Test {
    public static void main(String[] args) {
      X obj = new Y(); //Line n1
       obj.A(); //Line n2
       obj.B(); //Line n3
       obj.C(); //Line n4
    }
}
```

What will be the result of compiling and executing above code?

A. A-B-C-

B. AB-C-

C. Compilation error in class Y

D. Compilation error in class Test

6.1.38 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test extends String {
    @Override
    public String toString() {
        return "TEST" ;
    }

    public static void main(String[] args) {
        Test obj = new Test();
        System. out .println(obj);
    }
}
```

What will be the result of compiling and executing Test class?

A. TEST

B. Output string contains @ symbol

C. Exception is thrown at runtime

D. Compilation error

6.1.39 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        String str = "PANIC" ;
        StringBuilder sb = new StringBuilder( "THET" );
        System. out .println(str.replace( "N" , sb)); //Line n1
    }
}
```

What will be the result of compiling and executing Test class?

A. PANIC

B. PATHETIC

C. Line n1 causes compilation error

D. Line n1 throws error at runtime

6.1.40 Given code:

```
package com.udayankhattry.ocp1;

public class Pen {
    public static void main(String[] args) {
       Pen p1 = new Pen(); //Line n1
        Pen p2 = new Pen(); //Line n2
        p1 = p2; //Line n3
        p1 = null ; //Line n4
    }
}
```

When is the Pen object, created at Line n1, will be eligible for Garbage Collection?

A. After Line n2

B. After Line n3

C. After Line n4

D. At the end of main method

6.1.41 Given code of TestRectangle. java file:

```
package com.udayankhattry.ocp1;

class Rectangle {
    private int height ;
    private int width ;

     public Rectangle( int height, int width) {
        this . height = height;
        this . width = width;
    }

     public int getHeight() {
        return height ;
    }

     public int getWidth() {
        return width ;
    }
}

public class TestRectangle {
    public static void main(String[] args) {
        private int i = 100 ;
        private int j = 200 ;
       Rectangle rect = new Rectangle(i, j);
       System. out .println(rect.getHeight() +
                            ", " + rect.getWidth());
    }
}
```

What will be the result of compiling and executing above code?

A. 100, 200

B. 200, 100

C. Compilation Error

D. 0, 0

6.1.42 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.io.FileNotFoundException;

public class Test {
    public static void main(String[] args) {
        try {
           System. out .println(args[ 1 ].length());
       } catch (RuntimeException ex) {
           System. out .println( "ONE" );
       } catch (FileNotFoundException ex) {
           System. out .println( "TWO" );
       }
       System. out .println( "THREE" );
    }
}
```

What will be the result of compiling and executing Test class?

A.	ONE THREE
B.	TWO THREE
C.	THREE
D.	None of the System.out.println statements is executed
E.	Compilation error

6.1.43 Which of the following phases of software development are affected by modules?

A. Coding

B. Compilation

C. Testing

D. Packaging

E. Execution

F. All of the other options

6.1.44 Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
       System. out .println( "Welcome " + args[ 0 ] + "!" );
    }
}
```

And the commands:

```
javac Test.java
java Test "James Gosling" "Bill Joy"
```

What is the result?

A. Welcome James Gosling!

B. Welcome Bill Joy

C. Welcome "James Gosling!"

D. Welcome "Bill Joy"

E. Welcome James

F. Welcome Gosling

G. Welcome Bill

H. Welcome Joy

6.1.45 Given below the directory/file structure on Windows platform:

```
C:
├───src
│   └───<MODULE_DIRECTORY_NAME>
│       │  module-info.java
│       │
│       └───com
│           └───messages
│                 Test.java
│
└───out
```

Below is the code of Test.java file:

```
package com.messages;

public class Test {
    public static void main(String [] args) {
       System. out .println( "COMMIT OR QUIT" );
    }
}
```

And below is the code of module-info.java file:

```
module com.messages {
}
```

And below commands:

```
C:\>javac --module-source-path src -d out --module com.messages

C:\>java -p out -m com.messages/com.messages.Test
```

Which of the following options replaces <MODULE_DIRECTORY_NAME> such that output of the 2nd command is: COMMIT OR QUIT?

A. messages

B. com.messages

C. mymodule

D. messages-module

E. com.messages-module

F. com.messages.module

6.1.46 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
       List<Integer> list = new ArrayList<>();
        byte b = 10 ;
       list.add(b); //Line n1
        var mul = list.get( 0 ) * list.get( 0 ); //Line n2
        System. out .println(mul);
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. An exception is thrown at runtime

D. 10

E. 100

6.1.47 Below is the code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util. ArrayList;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        var list = new ArrayList<>(); //Line n1
        list.add( 7 );
       list.add( 14 );
       list.add( 21 );

         var sum = 0 ; //Line n2
        for ( int i : list){ //Line n3
            sum += i;
       }
       System. out .println(sum);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error at Line n1

B. Compilation error at Line n2

C. Compilation error at Line n3

D. 42

6.1.48 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Vehicle {
    public int getRegistrationNumber() {
        return 1 ;
    }
}

class Car {
    public double getRegistrationNumber() {
        return 2 ;
    }
}

public class Test {
    public static void main(String[] args) {
        Vehicle obj = new Car();
        System. out .println(obj.getRegistrationNumber());
    }
}
```

What will be the result of compiling and executing above code?

A. 1

B. 2

C. An exception is thrown at runtime

D. Compilation error in Vehicle class

E. Compilation error in Car class

F. Compilation error in Test class

6.1.49 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Currency {
    String notation = "-" ; //Line n1

    String getNotation() { //Line n2
        return notation ;
    }
}

class USDollar extends Currency {
    String notation = "$" ; //Line n3

    String getNotation() { //Line n4
        return notation ;
    }
}

class Euro extends Currency {
    protected String notation = "€" ; //Line n5

    protected String getNotation() { //Line n6
```

```
        return notation ;
    }
}

public class Test {
    public static void main(String[] args) {
       Currency c1 = new USDollar();
       System. out .println(c1. notation +
              ":" + c1.getNotation());

          Currency c2 = new Euro();
       System. out .println(c2. notation +
              ":" + c2.getNotation());
    }
}
```

What will be the result of compiling and executing above code?

A.	-:$ -:€
B.	-:- -:-
C.	$:$ €:€
D.	Compilation error in USDollar class
E.	Compilation error in Euro class

6.1.50 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        double [] arr = new int [ 2 ]; //Line n1
        System. out .println(arr[ 0 ]); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. 0

B. 0.0

C. Line n1 causes compilation error

D. Line n2 causes runtime exception

6.1.51 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface MyInterface {
    void calculate();
}

public class Test {
    public static void main(String[] args) {
       MyInterface obj = () ->  {
           int i = 10 ;
          i++;
          System. out .println(i);
           return ;
       };
       obj.calculate();
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. Exception is thrown at runtime

C. 10

D. 11

6.1.52 Which of the options of java command displays module resolution output?

A. --display-module-resolutio n

B. --module-resolution

C. --show-module-resolution

D. --list-modules

E. --list-module-resolution

6.1.53 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        final int i1 = 1 ;
        final Integer i2 = 1 ;
        final String s1 = ":ONE" ;

          String str1 = i1 + s1;
       String str2 = i2 + s1;

          System. out .println(str1 == "1:ONE" );
       System. out .println(str2 == "1:ONE" );
    }
}
```

What will be the result of compiling and executing Test class?

A.	true true	B.	true false
C.	false false	D.	false true

6.1.54 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

interface Blogger {
    default void blog() throws Exception {
        System. out .println( "GENERIC" );
    }
}

class TravelBlogger implements Blogger {
    public void blog() {
        System. out .println( "TRAVEL" );
    }
}

public class Test {
    public static void main(String[] args) {
        Blogger blogger = new TravelBlogger(); //Line n1
         ((TravelBlogger)blogger).blog(); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in TravelBlogger class

B. Compilation error in Test class

C. GENERIC is printed on to the console and program terminates successfully

D. TRAVEL is printed on to the console and program terminates successfully

E. An exception is thrown at runtime

6.1.55 Given below the directory/file structure on Windows platform:

```
C:
+---src
|  \---solarsystem
|      |  module-info.java
|      |
|      \---com
|          +---planet
|          |      Earth.java
|          |
|          \---satellite
\---out
```

Please note directory 'planet' contains Earth.java file, directory 'satellite' is empty and there is no 'artificial' directory under 'satellite' directory.

Contents of C:\src\solarsystem\com\planet\Earth.java file:

```
package com.planet;

public class Earth{
   //Lots of valid codes
}
```

Contents of C:\src\solarsystem\module-info.java file:

```
module solarsystem {
    exports com.planet; //Line n1
    exports com.satellite; //Line n2
    exports com.satellite.artificial; //Line n3
}
```

And below javac command is executed from C:\

```
javac -d out --module-source-path src -m solarsystem
```

What is the result?

A. Code compiles without any warning

B. Compilation error for Line n2 only

C. Compilation error for Line n3 only

D. Compilation error for both Line n2 and Line n3

E. Code compiles but with a warning for Line n2 only

F. Code compiles but with a warning for Line n3 only

G. Code compiles but with a warning for both Line n2 and Line n3

6.1.56 Consider below codes of 3 java files:

```
//Shrinkable.java
package com.udayankhattry.ocp1;

public interface Shrinkable {
    public static void shrinkPercentage() {
       System. out .println( "80%" );
    }
}
```

```
//AntMan.java
package com.udayankhattry.ocp1;

public class AntMan implements Shrinkable { }


//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        AntMan.shrinkPercentage();
    }
}
```

Which of the following statements is correct?

A. There is a compilation error in Shrinkable.java file

B. There is a compilation error in AntMan.java file

C. There is a compilation error in Test.java file

D. There is no compilation error and on execution, Test class prints 80% on to the console

6.1.57 In a modular application, is it possible to have empty module-info.java file?

A. Yes

B. No

6.1.58 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.sql.SQLException;

public class Test {
    private static void getReport() throws SQLException {
        try {
            throw new SQLException();
        } catch (Exception e) {
            throw null ; //Line 10
        }
    }

     public static void main(String[] args) {
        try {
            getReport (); //Line 16
        } catch (SQLException e) {
           System. out .println( "REPORT ERROR" );
        }
    }
}
```

What will be the result of compiling and executing Test class?

A. REPORT ERROR is printed on to the console and program terminates successfully

B. Program ends abruptly

C. Line 10 causes compilation failure

D. Line 16 causes compilation failure

6.1.59 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        var str = "Have Faith!" ; //Line n1
        System. out .println(str.length() +
                    " : " + str.charAt( 10 )); //Line n2
    }
}
```

What will be the result of compiling and executing Test class ?

A. 11 : !

B. 10 : !

C. An exception is thrown at runtime

D. 11 : h

E. 10 : h

F. Compilation error

6.1.60 What will be the result of compiling and executing Test class?

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
        int i = 2 ;
        boolean res = false ;
       res = i++ == 2 || --i == 2 && --i == 2 ;
       System. out .println(i);
    }
}
```

A. 2

B. 3

C. 1

D. Compilation error

6.1.61 Consider below interface:

```
interface ICalculator {
    int calc( int x);
}
```

Which of the following is the correct lambda expression for ICalculator?

A. ICalculator obj1 = x -> return x*x;

B. ICalculator obj2 = (x) -> return x*x;

C. ICalculator obj3 = x - > x*x;

D. ICalculator obj4 = x -> x*x;

6.1.62 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;
```

```
public class Test {
    public static void main(String[] args) {
        int i;
       outer:
        do {
           i = 5 ;
           inner:
            while ( true ) {
               System. out .println(i--);
                if (i == 4 ) {
                    break outer;
                }
            }
       } while ( true );
    }
}
```

What will be the result of compiling and executing Test class?

A.	Prints 5 in an infinite loop	B.	Prints 5 once
C.	Compilation error	D.	5 3 2 1

6.1.63 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        do {
           System. out .print( 100 );
       } while ( true ); //Line n1

           System. out .println( 200 ); //Line n2
    }
}
```

Which of the following statements is correct for above code?

A. Compiles successfully and on execution prints 200

B. Compiles successfully and on execution prints 100 in infinite loop

C. Unreachable code compilation error

D. Compiles successfully and on execution prints 100200

6.1.64 Consider below code of "Util.txt" file:

```
package com.udayankhattry.ocp1;

public class Util {
    public static void main(String [] args) {
       System. out .println(args[ 0 ] + args[ 1 ]);
    }
}
```

And the command:

```
java --source 11 Util.txt 10 20
```

What is the result?

A. Above command causes error

B. 30

C. 1020

D. 2010

6.1.65 Consider below code of Quotes.java file:

```
package com.udayankhattry.ocp1;

public class Quotes {
    String quote = null ;
    public Quotes() {
    }

    public Quotes(String str) {
       quote = str;
    }

    public void display() {
       System. out .println( quote );
    }

     public static void main(String [] args) {
       Quotes q1 = new Quotes();
       Quotes q2 = new Quotes( "NEVER GIVE UP!" );
       q1.display();
       q1.display();
    }
}
```

What will be the result of compiling and executing Quotes class?

A.	null NEVER GIVE UP!	B.	null null
C.	NEVER GIVE UP! null	D.	NEVER GIVE UP! NEVER GIVE UP!
E.	Compilation error		

6.1.66 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       var x = new int []{ 1 };
       var y = new int []{ 2 };
       var z = new int []{ 3 };
      System. out .println((x = y = z)[ 0 ] + y[ 0 ] + z[ 0 ]);
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error

B. An exception is thrown at runtime

C. 3

D. 6

E. 7

F. 8

G. 9

6.1.67 Consider below codes of 3 java files:

```
//Profitable1.java
package com.udayankhattry.ocp1;

public interface Profitable1 {
    default double profit() {
        return 12.5 ;
    }
}

//Profitable2.java
package com.udayankhattry.ocp1;

public interface Profitable2 {
    default double profit() {
        return 25.5 ;
    }
}

//Profit.java
package com.udayankhattry.ocp1;

public abstract class Profit implements
            Profitable1, Profitable2 {
    /*INSERT*/
}
```

Which of the following needs to be done so that there is no compilation error?

A.	No need for any modifications, code compiles as is

B.	Replace /*INSERT*/ with below code:
	double profit() {
	    return 50.0 ;
	}

C.	Replace /*INSERT*/ with below code:
	public default double profit() {
	    return 50.0 ;

	}

D.	Replace /*INSERT*/ with below code:
	protected double profit() {
	    return 50.0 ;
	}

E.	Replace /*INSERT*/ with below code:
	public double profit() {
	    return Profitable1.profit();
	}

F.	Replace /*INSERT*/ with below code:
	public double profit() {
	    return Profitable2. **super** .profit();
	}

6.1.68 Consider below codes of 3 java files:

```
//Sellable.java
package com.udayankhattry.ocp1;

public interface Sellable {
    double getPrice();

     default String symbol() {
        return "$" ;
    }
}


//Chair.java
package com.udayankhattry.ocp1;

public class Chair implements Sellable {
    public double getPrice() {
        return 35 ;
    }

     public String symbol() {
        return "£" ;
    }
}
```

```
//Test.java
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
       Sellable obj = new Chair(); //Line n1
        System. out .println(obj.symbol() +
             obj.getPrice()); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

A. Compilation error in Chair class

B. Compilation error in Test class

C. It compiles successfully and on execution prints $35 on to the console

D. It compiles successfully and on execution prints $35.0 on to the console

E. It compiles successfully and on execution prints $35.00 on to the console

F. It compiles successfully and on execution prints £35 on to the console

G. It compiles successfully and on execution prints £35.0 on to the console

H. It compiles successfully and on execution prints £35.00 on to the console

6.1.69 Given code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        try {
            try {
               System. out .println(args[ 1 ]); //Line n1
            } catch (RuntimeException e) {
               System. out .print( "INHALE-" ); //Line n2
                throw e; //Line n3
```

```
             } finally {
                System. out .print( "EXHALE-" ); //Line n4
             }
        } catch (RuntimeException e) {
            System. out .print( "INHALE-" ); //Line n5
         } finally {
            System. out .print( "EXHALE" ); //Line n6
         }
    }
}
```

And the command:

```
java Test.java
```

What is the result?

A. INHALE-EXHALE

B. INHALE-EXHALE-

C. INHALE-EXHALE-INHALE-

D. INHALE-EXHALE-EXHALE

E. INHALE-EXHALE-INHALE-EXHALE

6.1.70 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
        int grade = 60 ;
        if (grade = 60 )
            System. out .println( "You passed..." );
        else
             System. out .println( "You failed..." );
    }
}
```

What will be the result of compiling and executing Test class?

A. You passed...

B. You failed...

C. Compilation error

D. Produces no output

6.1.71 Consider the following class:

```
public class Employee {
   public int passportNo ; //line 2
}
```

Which of the following is the correct way to make the variable 'passportNo' read only for any other class?

A. Make 'passportNo' private

B. Make 'passportNo' private and provide a public method getPassportNo() which will return its value

C. Make 'passportNo' static and provide a public static method getPassportNo() which will return its value

D. Remove 'public' from the 'passportNo' declaration. i.e., change line 2 to int passportNo;

6.1.72 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String [] args) {
       boolean flag = false ;
       System. out .println((flag = true ) |
             (flag = false ) || (flag = true ));
       System. out .println(flag);
    }
}
```

What will be the result of compiling and executing Test class?

A.	true false	B.	false true
C.	true true	D.	false false

E.	Compilation error		

6.1.73 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;
import java.util.List;

class Person {
    private String name ;
    private int age ;

    Person(String name, int age) {
        this . name = name;
        this . age = age;
    }

    public String toString() {
        return "Person[" + name + ", " + age + "]" ;
    }

    public boolean equals(Person obj) {
        if (obj instanceof Person) {
            Person person = (Person)obj;
            if ( this . name .equals(person. name ) &&
                            this . age == person. age ) {
                return true ;
            }
        }
        return false ;
    }
}

public class Test {
    public static void main(String[] args) {
        List<Person> persons = new ArrayList<>();
        persons.add( new Person( "Liam" , 25 ));
        persons.add( new Person( "Liam" , 27 ));
        persons.add( new Person( "Liam" , 25 ));
        persons.add( new Person( "Liam" , 25 ));
```

```
            persons.remove( new Person( "Liam" , 25 ));

            for (Person person : persons) {
             System. out .println(person);
        }
    }
}
```

What will be the result of compiling and executing Test class?

A.	Person[Liam, 27] Person[Liam, 25] Person[Liam, 25]	B.	Person[Liam, 25] Person[Liam, 27] Person[Liam, 25]
C.	Person[Liam, 27]	D.	Person[Liam, 25] Person[Liam, 27] Person[Liam, 25] Person[Liam, 25]

6.1.74 Given code of Test.java file:

```
package com.udayankhattry.ocp1;

class Super {
    final int NUM = - 1 ; //Line n1
}

class Sub extends Super {
     /*INSERT*/
}

public class Test {
     public static void main(String[] args) {
        Sub obj = new Sub();
        obj. NUM = 200 ; //Line n2
         System. out .println(obj. NUM ); //Line n3
     }
}
```

Above code causes compilation error, which modifications, done independently, enable the code to compile and on execution print 200 on to the console?

Select ALL that apply.

A. Remove final modifier from Line n1

B. Replace /*INSERT*/ with byte NUM;

C. Replace /*INSERT*/ with short NUM;

D. Replace /*INSERT*/ with int NUM;

E. Replace /*INSERT*/ with float NUM;

F. Replace /*INSERT*/ with double NUM;

G. Replace /*INSERT*/ with boolean NUM;

H. Replace /*INSERT*/ with Object NUM;

6.1.75 For the given code:

```
package com.udayankhattry.ocp1;

interface Formatter< T > {
    T format( T t);
}

public class Test {
    public static void main(String[] args) {
        System. out .println( formatString ( "hello" , /*INSERT*/ ));
        System. out .println( formatString ( "hELP" , /*INSERT*/ ));
    }

    private static String formatString(
            String str, Formatter<String> formatter) {
        return formatter.format(str);
    }
}
```

Which of the following options can replace /*INSERT*/ such that on executing Test class, passed Strings are formatted in Camel case? The output should be:

Hello
Help

Consider that passed string should be continuous, without any white-spaces in between. Select one option from below.

A.	s -> s.substring(0, 1).toUpperCase() + s.substring(1).toLowerCase()
B.	s -> s.substring(0, 1).toUpperCase() + s.substring(1, 5).toLowerCase()
C.	s -> s.toCamelCase()
D.	Other three options are not correct

6.1.76 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.ArrayList;

public class Test {
    private var place = "Unknown" ;  //Line n1
    public static final var DISTANCE = 200 ; //Line n2

     public static void main(String[] args) {
        var list1 = new ArrayList<>(); //Line n3

          var list2 = new ArrayList(); //Line n4

          var lambda1 = () ->
              System. out .println( "Hello" ); //Line n5

          var var = 100 ; //Line n6
     }
}
```

Which of the following statements are correct?
Select ALL that apply.

A. Line n1 compiles successfully

B. Line n2 compiles successfully

C. Line n3 compiles successfully

D. Line n4 compiles successfully

E. Line n5 compiles successfully

F. Line n6 compiles successfully

6.1.77 Interface java.util.function.Predicate<T> declares below non-overriding abstract method:

```
   boolean test(T t);
```

Given code of Test.java file:

```
package com.udayankhattry.ocp1;

import java.util.function.Predicate;

public class Test {
    public static void main(String[] args) {
      String [] arr = { "A" , "ab" , "bab" ,
            "Aa" , "bb" , "baba" , "aba" , "Abab" };

        Predicate<String> p = s ->
            s.toUpperCase().substring( 0 , 1 ).equals( "A" );

        processStringArray (arr, p);
   }

    private static void processStringArray(
            String [] arr, Predicate<String> predicate) {
       for (String str : arr) {
          if (predicate.test(str)) {
             System. out .println(str);
          }
       }
   }
}
```

What will be the result of compiling and executing Test class?

A.	Compilation error	B.	Runtime exception
C.	A Aa Abab	D.	ab aba
E.	A ab Aa aba Abab		

6.1.78 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

public class Test {
    public static void main(String[] args) {
      String [][] arr = {{ "1" , "2" , "3" },
            { "4" , "5" }, { "6" , "7" }};
       for ( int i = 0 ; i < arr. length ; i++) {
          for ( int j = 0 ; j < arr[i]. length ; j++) {
             System. out .print(arr[i][j] + " " );
              if (arr[i][j].equals( "2" )) {
                 break ;
             }
          }
       }
    }
}
```

What will be the result of compiling and executing Test class?

A. 1 2

B. 1 2 3 4 5 6 7

C. 1 2 4 5 6 7

D. 1 3

E. 1 3 4 5 6 7

F. Compilation fails

G. An exception is thrown at runtime

6.1.79 Consider below command executes successfully from C:\ on Windows environment:

```
java -p out -m com.teaching.util/com.teaching.util.GradeCalculator
```

**Directory 'out' or its subdirectories don't contain jar file.
Given command executes successfully.**

**Which of the following statements are correct?
Select ALL that apply .**

A. Module name is com.teaching.util.GradeCalculator

B. Module name is com.teaching.util

C. Source files must be available under subdirectories of out directory

D. Class files must be available under subdirectories of out directory

E. 'out' directory must contain a directory named 'com.teaching.util'

6.1.80 Consider below code of Test.java file:

```
package com.udayankhattry.ocp1;

class Super {
    Super() {
        System. out .print( "Reach" );
    }
}

class Sub extends Super {
    Sub() {
        Super();
        System. out .print( "Out" );
    }
}

public class Test {
     public static void main(String[] args) {
         new Sub();
    }
}
```

What will be the result of compiling and executing above code?

A. Compilation Error in Super class

B. Compilation Error in Sub class

C. Compilation Error in Test class

D. It prints ReachOut on to the console

E. It prints OutReach on to the console

6.2 Answers of Practice Test - 6 with Explanation

6.1.1 Answer: A

Reason :
main method's parameter variable name is "args" and it is LOCAL to main method. So, same name "args" can't be used directly within the curly brackets of main method. short [] args = new short[]{50, 50}; causes compilation error for using same name for local variable.

6.1.2 Answer: F

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name, package name or loop's label but it cannot be used as a class or interface name.

For the 1st loop variable 'i' infers to int type, so no issues for 1st loop and for the 2nd loop variable 'j' infers to int type, so no issues for 2nd loop as well.

Basic/Regular for loop has following form:
for ([ForInit] ; [Expression] ; [ForUpdate]) {...}
[ForInit] can be local variable initialization or the following expressions:
Assignment
PreIncrementExpression
PreDecrementExpression
PostIncrementExpression
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment
PreIncrementExpressio n
PreDecrementExpression
PostIncrementExpression
PostDecrementExpression
MethodInvocation
ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for both the loops, `System.out.print(...)` is used as [ForUpdate] expression, which is a MethodInvocation expression and hence a valid statement.
Given code compiles successfully.

Let's check the iterations:
1st iteration of outer: i = 0. i < 3 evaluates to true.
    i = 1.
    1st iteration of inner: j = 0. j < 3 evaluates to true as j = 0. Boolean expression `i > ++j` = `1 > 1` evaluates to false. j = 1.
    2nd iteration of inner: `System.out.print(j)` prints 1 to the console. j < 3 evaluates to true as j = 1. Boolean expression `i > ++j` = `1 > 2` evaluates to false. j = 2.
    3rd iteration of inner: `System.out.print(j)` prints 2 to the console. j < 3 evaluates to true as j = 2. Boolean expression `i > ++j` = `1 > 3` evaluates to false. j = 3.
    4th iteration of inner: `System.out.print(j)` prints 3 to the console. j < 3 evaluates to false as j = 3. Control goes out of inner for loop and to the [ForUpdate] expression of outer loop.
2nd iteration of outer: `System.out.print(i)` prints 1 to the console. i < 3 evaluates to true as i = 1.
    i = 2.
    1st iteration of inner: j = 0. j < 3 evaluates to true as j = 0. Boolean expression `i > ++j` = `2 > 1` evaluates to true. j = 1. ` break outer;` takes the control out of the outer for loop.

Program terminates successfully after printing 1231 on to the console.

6.1.3 Answer: G

Reason :
At Line n1:
sb --> {"B"}

append(...) method in StringBuilder class is overloaded to accept various arguments and 2 such arguments are String and CharSequence. It's return type is StringBuilder and as StringBuilder class implements CharSequence interface, hence 'sb.append("A")' can easily be passed as and argument to sb.append(...) method. Line n2 compiles successfully.
At Line n2:
sb.append(sb.append("A")); //sb --> {"B"}
sb.append({"BA"}); //sb --> {"BA"}
{"BABA"}

Hence, Line n3 prints BABA

6.1.4 Answer: B

Reason :
Functional APIs of the Java SE Platform are found in 'java.base' module. Please check the link:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/module-summary.html

java.se is an aggregator module, which collects and re-exports the content of other modules but adds no content of its own.

'jdk.se', 'jdk.base', 'java' and 'se' are not valid system modules.

6.1.5 Answer: B

Reason :
Variables x and y are declared with default scope, so can be accessed in same package.
There is only one copy of static variable for all the instances of the class. Variable x is shared by p1 and p2 both.
p1.x = 17; sets the value of variable x to 17, p2.x = 19; overrides the value of variable x to 19.
So in System.out.println statement both p1.x and p2.x prints 19 .

p1.x and p2.x don't cause any compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Point.x is the preferred syntax.

Each object has its own copy of instance variables, so p1.y is 35 and p2.y is 40.

Output of the given code is: 19:35:19:40

6.1.6 Answer: A

Reason :
Method find() of Child class correctly overrides the method find() of Parent class, so there is no compilation error in Child class.
Code in Test class doesn't cause any error as well.

java.io.FileNotFoundException extends java.io.IOException
and
java.io.IOException extends java.lang.Exception

Even though method find() declares to throw IOException but at runtime an instance of FileNotFoundException is thrown.
A catch handler for FileNotFoundException is available and hence X is printed on to the console.
After that finally block is executed, which prints Z to the console.

6.1.7 Answer: B,C,E

Reason :
switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums. In this case, all are valid values but only 3 executes "case 7:".
case is comparing integer value 7. NOTE: character seven, '7' is different from integer value seven, 7. So "char var = '7';" and "Character var = '7';" will print DEFAULT on to the console.

Please also note that the identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

6.1.8 Answer: A

Reason :
"Result is: " + (10 != 5) [Nothing to evaluate at left side, so let's evaluate the right side of +, 10 != 5 is true.]

= "Result is: " + true [+ operator behaves as concatenation operator]
= "Result is: true"

6.1.9 Answer: A,B,C,G

Reason :
Static overloaded method join(...) was added in JDK 1.8 and has below declarations:
1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", "A", "B", "C"); returns "A.B.C"
String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"
String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, "A", "B"); throws NullPointerException
String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,
String str = null; String.join("-", str); returns "null"
String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,
String.join(".", List.of("A", "B", "C")); returns "A.B.C"
String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,
String.join(null, List.of("HELLO")); throws NullPointerException
List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

List<String> list = new ArrayList<>(); list.add("A"); list.add(null);
String.join("::", list); returns "A::null "
Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Based on above points:
Snippet 1 compiles successfully and on execution prints "1. null::null" on to the console.
Snippet 2 compiles successfully and on execution prints "2. " on to the console.
Snippet 3 compiles successfully and on execution throws NullPointerException as elements refer to null
Snippet 4 causes compilation error as `String.join(".", null)` is an ambiguous call and compiler is not sure about which overloaded join(...) method should be tagged for this call.

6.1.10 Answer: F

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

At Line n1, variable 'i' infers to int type, so given loop executes 5 times from values of i from 5 to 1.

To know more about join and split methods of String class, please check the URLs:
https://udayankhattry.com/join-string/
https://udayankhattry.com/split-string/

Let's solve the given code:
Initially, res = ""
1st iteration: i = 1. code of 'case 1:' is executed, res = "" + "UP " = "UP

". break; statement breaks out of the switch-case block and not the loop.
2nd iteration: i = 2. code of 'case 2:' is executed, res = "UP " + "TO " = "UP TO ". break; statement breaks out of the switch-case block.
3rd iteration: i = 3. code of 'case 3:' is executed. break; statement breaks out of the switch-case block.
4th iteration: i = 4. code of 'case 4:' is executed. res = "UP TO " + "DATE" = "UP TO DATE". break loop; statement breaks out of the for loop .

Line n2 is executed:
String.join("-", res.split(" "))
= String.join("-", "UP TO DATE".split(" "))
= String.join("-", ["UP","TO","DATE"]) //Splits on the basis of single space " ".
= "UP-TO-DATE"

Line n2 prints UP-TO-DATE on to the console.

6.1.11 Answer: C,E

Reason :
Following import statements are correct:
import com.singaporezoo.animal.*;
import com.singaporezoo.animal.Panda;
NOTE: all small case letters in 'import' keyword.

6.1.12 Answer: A

Reason :
Instance method of subclass cannot override the static method of superclass.

Instance method at Line n2 tries to override the static method at Line n1 and hence Line n2 causes compilation error.

There is no issue with Line n3 as reference variable of superclass can refer to an instance of subclass.

At Line n4, paper.getType() doesn't cause compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Paper.getType() is the preferred syntax.

6.1.13 Answer: D

Reason :

It is legal for the constructors to have throws clause.
Constructors are not inherited by the Sub class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting .

Java compiler adds super(); as the first statement inside Sub class's constructor:

```
Sub() throws Exception {
    super(); //added by the compiler
    System.out.println("MATATA");
}
```

super(); invokes the constructor of Super class (which declares to throw RuntimeException), as RuntimeException is unchecked exception, therefore no handling is necessary in the constructor of Sub class.
Sub class's constructor declares to throw IOException but main(String []) method handles it.

There is no compilation error and output is: CARPE DIEM

6.1.14 Answer: C

Reason :

Constructor has the same name as the class, doesn't have return type and can accept parameters.
Out of the given 4 options, `public Planet(String str) {}` is the valid constructor declaration.

6.1.15 Answer: G

Reason :

As per Java 8, default methods were added in the interface. Interface X1 defines default method print(), there is no compilation error in interface X1. Method print() is implicitly public in X1.

interface X2 extends X1 and it overrides the default method print() of X1, overriding method in X2 is implicitly abstract and public. An interface in java can override the default method of super type with abstract modifier. interface X2 compiles successfully.

interface X3 extends X2 and it implements the abstract method print() of X2, overriding method in X3 is default and implicitly public. An interface in java can implement the abstract method of super type with default modifier. interface X3 compiles successfully.

class X implements X3 and therefore it inherits the default method print() defined in interface X3 .

`X1 obj = new X();` compiles successfully as X1 is of super type (X implements X3, X3 extends X2 and X2 extends X1).
`obj.print();` invokes the default method print() defined in interface X3 and hence X3 is printed on to the console.

6.1.16 Answer: C

Reason :
`new StringBuilder(5);` creates a StringBuilder instance, whose internal char array's length is 5 but the internal char array's length is adjusted when characters are added/removed from the StringBuilder instance.
`sb.append("0123456789");` successfully appends "0123456789" to the StringBuilder's instance referred by sb.
delete method accepts 2 parameters: delete(int start, int end), where start is inclusive and end is exclusive.
This method throws StringIndexOutOfBoundsException for following scenarios:
A. start is negative
B. start is greater than sb.length()
C. start is greater than end

If end is greater than the length of StringBuilder object, then StringIndexOutOfBoundsException is not thrown and end is set to sb.length().
So, in this case, `sb.delete(8, 1000);` is equivalent to `sb.delete(8, sb.length());` and this deletes characters at 8th index (8) and 9th index (9). So remaining characters are: "01234567".

StringBuilder class overrides toString() method, which prints the text stored in StringBuilder instance. Hence, `System.out.println(sb);` prints 01234567 on to the console.

6.1.17 Answer: C

Reason :
For readability purpose underscore (_) is used to separate numeric values. This is very useful in representing big numbers such as credit card numbers (1234_7654_9876_0987). Multiple underscores are also allowed within the digits. Hence, `int x = 5____0;` compiles successfully and variable x stores 50.
`float f = 123.76_86f;` compiles successfully.
1_2_3_4 is int literal 1234 and int can easily be assigned to double, hence `double d = 1_2_3_4;` compiles successfully.

____50 is a valid variable name, and as this variable is not available hence, int y = ____50; causes compilation error.

Underscores must be available within the digits. For the statement int z = 50____; as underscores are used after the digits, hence it causes compilation error.

6.1.18 Answer: C

Reason :
list.add(0, 'V'); => char 'V' is converted to Character object and stored as the first element in the list. list --> [V].
list.add('T'); => char 'T' is auto-boxed to Character object and stored at the end of the list. list --> [V,T].
list.add(1, 'E'); => char 'E' is auto-boxed to Character object and inserted at index 1 of the list, this shifts T to the right. list --> [V,E,T].
list.add(3, 'O'); => char 'O' is auto-boxed to Character object and added at index 3 of the list. list --> [V,E,T,O].
list.contains('O') => char 'O' is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove(3);.
list.remove(3); => Removes last element of the list. list --> [V,E,T].
for(char ch : list) => First list item is Character object, which is auto-unboxed and assigned to ch. This means in first iteration ch = 'V'; And after this it is simple enhanced for loop. Output is VET.

6.1.19 Answer: A

Reason :
case values must evaluate to the same/compatible type as the switch

expression can use.
switch expression can accept following:
char or Character,
byte or Byte,
short or Short,
int or Integer,
An enum only from Java 6,
A String expression only from Java 7.

In this case, switch expression [switch (score)] is of int type.
But case expressions, score < 70 and score >= 70 are of boolean type and hence compilation error.

6.1.20 Answer: C

Reason :
calculate method is correctly overloaded as both the methods have different signature: calculate(int, int) and calculate(byte, byte). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Calculator().calculate(b, i)` tags to `calculate(int, int)` as byte value is implicitly casted to int type.

Given code compiles successfully and on execution prints 120 on to the console.

6.1.21 Answer: A

Reason :
There are no issues with Line n1 and Line n2, both the statements compile successfully.

String class contains contentEquals(CharSequence) method. Please note that String, StringBuilder and StringBuffer classes implement CharSequence interface, hence contentEquals(CharSequence) method defined in String class cab be invoked with the argument of either String or StringBuilder or StringBuffer.
At Line n3, `str.contentEquals(sb)` is invoked with StringBuilder argument and hence it compiles fine. On execution it would compare the contents of String object and the passed StringBuilder object. As both the String object and StringBuilder object contains same content "Game on", hence on execution, Line n3 will print true.

contentEquals method is not available in StringBuilder class and hence Line n4 causes compilation error.

equals method declared in Object class has the declaration: `public boolean equals(Object)`. Generally, equals method is used to compare different instances of same class but if you pass any other object, there is no compilation error. Parameter type is Object so it can accept any Java object.

`str.equals(sb)` => It compiles fine, String class overrides equals(Object) method but as 'sb' is of StringBuilder type so `str.equals(sb)` would return false at runtime.

`sb.equals(str)` => It also compiles fine, StringBuilder class doesn't override equals(Object) method. So Object version is invoked which uses == operator, hence `sb.equals(str)` would return false as well at runtime.

6.1.22 Answer: A

Reason :
Top level class can have two access modifiers: public and default. Over here Test class has private modifier and hence compilation error.

6.1.23 Answer: B

Reason :
Though given code looks strange but it is possible in java to provide same name to package, class (and constructor), variable and method. Above code compiles successfully and on execution prints ONE on to the console. Constructor is not invoked as 'new' keyword is not used and that is why TWO will not be printed to the console.
In real world coding, you would not see such code and that is why it is a good question for the certification exam.

6.1.24 Answer: A

Reason :
Though Predicate is a generic interface but raw type is also allowed. Type of the variable in lambda expression is inferred by the generic type of Predicate<T> interface.

In this case, Predicate pr1 = s -> s.length() < 4; Predicate is considered of Object type so variable 's' is of Object type and Object class doesn't have length() method. So, `s.length()` causes compilation error.

6.1.25 Answer: A,B,D

Reason :
It is clear from Line n1 that, method name should be subtext, it should be static method, it should accept 3 parameters (String, int, int).
As subtext(str, 3, 8) is passed as an argument of System.out.println method, hence subtext method's return type can be anything apart from void. println method is overloaded to accept all primitive types, char [], String type and Object type. int[] are String [] are of Object type.

In the given options, method specifying int as return type cannot return null as null can't be assigned to primitive type. int subtext(...) would give compilation error.

Local variable Type inference was added in JDK 10 .
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.
var type cannot be used as method parameters or method return type.

6.1.26 Answer: D

Reason :
Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.
Compiler adds super(); as the first line in OTG's constructor: OTG(int capacity, String type) { super(); } but PenDrive class doesn't have a no-argument constructor and that is why OTG's constructor causes compilation error.
To correct the compilation error, parent class constructor should be invoked by using super(capacity); This would resolve compilation error.
super(capacity); will only assign value to 'capacity' property, to assign value to 'type' property, another statement is needed.

`this.type = type;` must be the 2nd statement.

Remember: Constructor call using this(...) or super(...) must be the first statement inside the constructor.

6.1.27 Answer: F

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name, package name or loop's label but it cannot be used as a class or interface name.
At Line n1, 'var' is used as while loop's label, so no issues at Line n1 .

while(true) is and infinite loop and compiler is aware of that, so unless and until you provide the logic to break out of the loop, compiler considers Line n2 as unreachable code, which causes compilation failure.
continue statements don't break out of the loop so even if you use `continue;`, `continue i;` or `continue var;` to replace /*INSERT*/, unreachable code error will still be there.

`break;` and `break i;` statements are same and will break out to the inner for loop, Line n2 will still be unreachable.

`break var;` will successfully break out of the outer while loop.

On execution, control will come out of the while loop, when i equals to 2 and THINK DIFFERENT will be printed on to the console.

6.1.28 Answer: C

Reason :
As method div() doesn't declare to throw any Checked Exception, hence main(String []) method is not suppose to handle it, try-finally without catch is valid here. There is no compilation error in the code.

Method div() throws an instance of ArithmeticException and method

div() doesn't handle it, so it forwards the exception to calling method main(String []).
Method main(String []) doesn't handle ArithmeticException so it forwards it to JVM, but just before that, finally block is executed. This prints FINALLY on to the console.
After that JVM prints the stack trace and terminates the program abruptly.

6.1.29 Answer: A

Reason :
ArrayList's 1st and 3rd items are referring to same String instance referred by 's' [s --> "Hello"] and 2nd item is referring to another instance of String.

String is immutable, which means s.replace("l", "L"); creates another String instance "HeLLo" but 's' still refers to "Hello" [s --> "Hello"].

[Hello, Hello, Hello] is printed in the output.

6.1.30 Answer: G

Reason :
Both try and catch blocks have return; statement, which means either of the return statements will definitely get executed. Hence, compiler tags `System.out.println("DONE");` as unreachable and this causes compilation error.

6.1.31 Answer: D

Reason :
Reference variable to which lambda expression is assigned is known as target type. Target type can be a static variable, instance variable, local variable, method parameter or return type. Lambda expression doesn't work without target type and target type must be a functional interface. Functional interface was added in JDK 8 and it contains one non-overriding abstract method.
As Greetings is abstract class, so lambda expression cannot be used in this case.

6.1.32 Answer: F

Reason :
List cannot accept primitives, it can accept objects only. So, when 100

and 200 are added to the list, then auto-boxing feature converts these to wrapper objects of Integer type.
So, 4 items gets added to the list. One can expect the same behavior with remove method as well that 100 will be auto-boxed to Integer object.
But remove method is overloaded in List interface: remove(int) => Removes the element from the specified position in this list.
and remove(Object) => Removes the first occurrence of the specified element from the list.
As remove(int) version is available, which perfectly matches with the call remove(100); hence compiler does not do auto-boxing in this case.
But at runtime remove(100) tries to remove the element at 100th index and this throws IndexOutOfBoundsException.

6.1.33 Answer: D

Reason :
import statements are allowed in module descriptor file. These import statements are useful when you use 'provides' directive for Services but these are not in the scope of 1Z0-815 exam. For this exam you should know that import statements are allowed in module-info.java file .

By default all the module implicitly requires java.base module (one of the system module)

To get a list of all the system modules, use below command:
java --list-modules
And to check details of specific module (e.g. java.base), use below command:
java --describe-module java.base
You will notice that it exports java.lang package and hence module-info.java file is valid.

Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this case, module source path is 'src' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'out' directory. Please note generated class files will be arranged in package-wise directory structure.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
It is not used in this case as given module is not dependent upon other compiled/packaged application modules.

Module name is 'holidays' (from module-info.java file) and it is correctly placed under the module directory.
All the options of given javac command are valid, there is no need of any other java files. Given javac command completes successfully without any warning.

After compilation, you will get below directory/file structure:

```
C:
+---src
|   \---holidays
|           module-info.java
|
\---out
    \---holidays
            module-info.class
```

6.1.34 Answer: F

Reason :
ArrayList are different than arrays, though behind the scene ArrayList

uses Object[] to store its elements.

There are 2 things related to ArrayList, one is capacity and another is actual elements stored in the list, returned by size() method. If you don't pass anything to the ArrayList constructor, then default capacity is 10 but this doesn't mean that an ArrayList instance will be created containing 10 elements and all will be initialized to null.

In fact, size() method will still return 0 for this list. This list still doesn't contain even a single element. You need to use add method or its overloaded counterpart to add items to the list. Even if you want to add null values, you should still invoke some methods, nothing happens automatically.

In this question, new ArrayList<>(4); creates an ArrayList instance which can initially store 4 elements but currently it doesn't store any data.

Another point you should remember for the certification exam: Addition of elements in ArrayList should be continuous. If you are using add(index, Element) method to add items to the list, then index should be continuous, you simply can't skip any index.

In this case, list.add(0, "MOVE"); adds "MOVE" to 0th index. so after this operation list --> [MOVE]. You can now add at 0th index (existing elements will be shifted right) or you can add at index 1 but not at index 2. list.add(2, "ON"); throws an instance of java.lang.IndexOutOfBoundsException.

6.1.35 Answer: B

Reason :
Starting with JDK 11, it is possible to launch single-file source-code Programs.
If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args ]
     (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed.
Hence, `java Test.java --help` is equivalent to (but not exactly same as):
javac -d <memory> Test.java
java -cp <memory> Test --help

"--help" is considered as command-line argument, and on execution args[0] refers to "--help".
`System.out.println(args.length);` prints 1 on to the console.

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

6.1.36 Answer: A

Reason :
There is no compilation error in the given code.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

Given statement:
var list = List.of(new Division(100, 4), new Division(27, 0), new Division(25, 5)); => list refers to a List of Division type.

Static List.of() overloaded methods were added in Java 9 and these return an unmodifiable list containing passed elements. So, above list object referred by 'list' is unmodifiable. It is not allowed to invoke add/remove/set methods on the list object returned by List.of() method (no compilation error but an exception is thrown at runtime on invoking add/remove/set methods on list reference) but 3 Division instances are modifiable.

Iterable<T> interface has forEach(Consumer) method. List<E> extends Collection<E> & Collection<E> extends Iterable<E>, therefore forEach(Consumer) can easily be invoked on reference variable of List<E> type .

As Consumer is a Functional Interface, hence a lambda expression can be passed as argument to forEach() method.
forEach(Consumer) method performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Lambda expression at Line n1: `d -> d.divide()` is the correct implementation of `void accept(Division t);` and hence can easily be assigned to target type Consumer<Division>. Purpose of this lambda expression is to divide 'num' by 'den' and store the result in 'res' variable. Hence, statement at Line n1 will populate 'res' variable of the three Division objects.
Please note that for the 2nd Division object [num = 27, den = 0], divide() method throws a RuntimeException but it is handled. If you don't handle it, then stack trace will be printed and program will terminate abruptly.

Lambda expression at Line n2: `d -> System.out.println(d)` is also the correct implementation of `void accept(Division t);` and hence can easily be assigned to target type Consumer<Division>. Purpose of this lambda expression is to print the Division object on to the console. toString() method of Division object is invoked which prints the state of Division object, in case if den == 0, then "Infinity" is used in place of the value of variable 'res'.

Statement at Line n2 will print the details of 3 Division instances available in the list.

Output will be:

```
100/4 = 25
27/0 = Infinity
25/5 = 5
```

6.1.37 Answer: D

Reason :
Class Y correctly extends class X and it overrides method A() and provides two new methods B() and C().
At Line n1, obj is of X type and therefore obj.B(); and obj.C(); cause compilation error as these methods are not defined in class X.

6.1.38 Answer: D

Reason :
String is a final class so it cannot be extended.

6.1.39 Answer: B

Reason :
String class has following two overloaded replace methods:
1. public String replace(char oldChar, char newChar) {}:
Returns a string resulting from replacing all occurrences of oldChar in this string with newChar. If no replacement is done, then source String object is returned. e.g.
"Java".replace('a', 'A') --> returns new String object "JAvA".
"Java".replace('a', 'a') --> returns the source String object "Java" (no change).
"Java".replace('m', "M") --> returns the source String object "Java" (no change).

2. public String replace(CharSequence target, CharSequence replacement) {}:
Returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.
"Java".replace("a", "A") --> returns new String object "JAvA".
"Java".replace("a", "a") --> returns new String object "Java" (it replaces "a" with "a").
"Java".replace("m", "M") --> returns the source String object "Java" (no change).

As String, StringBuilder and StringBuffer all implement CharSequence, hence instances of these classes can be passed to replace method. Line n1 compiles successfully and on execution replaces "N" with "THET", and hence Line n1 prints PATHETIC on to the console.

6.1.40 Answer: B

Reason :
At Line n3, p1 starts referring to the object referred by p2(Created at Line n2).
So, after Line n3, object created at Line n1 becomes unreachable and thus eligible for Garbage Collection.

6.1.41 Answer: C

Reason :
i and j cannot be declared private as i and j are local variables.
Only final modifier can be used with local variables.

6.1.42 Answer: E

Reason :
java.io.FileNotFoundException exception is a checked exception.

Java doesn't allow to catch specific checked exceptions if these are not thrown by the statements inside try block.
catch(FileNotFoundException ex) {} causes compilation error in this case as System.out.println(1); will never throw FileNotFoundException.

NOTE: Java allows to catch Exception type. catch(Exception ex) {} will never cause compilation error.

6.1.43 Answer: F

Reason :
Modules affect all phases of development: coding, compiling, testing, packaging, deploying, running.

Coding: Developers need to provide module-info.java file with all the necessary directive.
Compilation: Options were added in javac command to support modules
Testing: Testing tools have been updated to test the modules
Packaging: Concept of modular jar and non-modular jar is already there
Execution: Options were added in java command to support modules

Hence modules affect all the phases of software development and in fact, it starts with designing itself.

6.1.44 Answer: A

Reason :
Please note, if passed command line arguments contain space(s) in between, then it is a common practice to enclosed within double quotes.
In this case "James Gosling" is passed as one String object and "Bill Joy" is also passed as one String object.
java Test "James Gosling" "Bill Joy" passes new String [] {"James Gosling", "Bill Joy"} to args of main method. args[0] refers to "James

Gosling" and args[1] refers to "Bill Joy".
Hence, Welcome James Gosling! is printed on to the console. While printing the String object, enclosing quotes are not shown.

To use quotes as part of the String, you can escape those using backslash, such as:
java Test "\"James Gosling"\" "\"Bill Joy"\"
Above command will print Welcome "James Gosling"! on to the console.

6.1.45 Answer: B

Reason :
Format of javac command is:
javac <options> <source files>

Below are the important options (related to modules) of javac command:
--module-source-path <module-source-path>:
Specify where to find input source files for multiple modules. In this case, module source path is 'src' directory.

--module <module-name>, -m <module-name>:
Compile only the specified module. This will compile all *.java file specified in the module. Multiple module names are separated by comma.

-d <directory>:
Specify where to place generated class files. In this case, it is 'out' directory.

--module-path <path>, -p <path>:
Specify where to find compiled/packaged application modules. It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries. For example, javac -p mods;singlemodule;connector.jar represents a module path which contains all the modules inside 'mods' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.
It is not used in this case as given module is not dependent upon other compiled/packaged application modules.

Format of the java command related to module is:
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
(to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries .
For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar'.

Given javac command:
C:\>javac --module-source-path src -d out --module com.messages => This is multi-module compilation command and for this command to work, module name and module directory name must be same.
Module name defined in module-info.java file is: 'com.messages'. Hence only com.messages can replace <MODULE_DIRECTORY_NAME>.

Please note that module names live in a global namespace, separate from other namespaces in Java. Hence, module name can use the name of the package, class or interface.

6.1.46 Answer: A

Reason :
list is of Integer type and variable 'b' is of byte type.
At Line n1, b is auto-boxed to Byte and not Integer and List<Integer> can't store Byte objects, therefore Line n1 causes compilation error.

Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

list.get(0) returns Integer and `list.get(0) * list.get(0)` is evaluated to int, therefore variable 'mul' infers to int. Line n2 compiles successfully.

6.1.47 Answer: C

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to in t

At Line n1, list refers to an ArrayList of Object type, because Generic type is not defined on the right side.
Three Integer objects are added to the list.

At Line n2, 'sum' infers to int.

At Line n3, as list is of ArrayList of Object type, hence each element of the list is considered as Object type and Object cannot be assigned to int.
Line n3 causes compilation error.

6.1.48 Answer: F

Reason :
class Car doesn't extend from Vehicle class, this means Vehicle is not super type of Car.
Hence, Vehicle obj = new Car(); causes compilation error.
As Vehicle and Car classes are not related, hence these don't cause any compilation error.

6.1.49 Answer: A

Reason :
Subclass overrides the methods of superclass but it hides the variables

of superclass.

Line n3 hides the variable created at Line n1 and Line n4 overrides the getNotation() method of Line n2. There is no compilation error for USDollar class as it correctly overrides getNotation() method.
Similarly, Line n5 hides the variable created at Line n1 and Line n6 overrides the getNotation() method of Line n2. There is no compilation error for Euro class as it correctly overrides getNotation() method as well.

'c1' is of Currency type, hence c1.notation refers to "-" and c1.getNotation() invokes overriding method of USDollar class and it returns "$".
Similarly, c2.notation refers to "-" and c2.getNotation() invokes overriding method of Euro class and it returns "€".

6.1.50 Answer: C

Reason :
int variable can easily be assigned to double type but double [] and int [] are not compatible. Hence, int [] object cannot be assigned to double [] type. Line n1 causes compilation error.

6.1.51 Answer: D

Reason :
No issues with lambda syntax: curly brackets and semicolon are available.
As calculate()) method of MyInterface interface returns void, hence `return;` statement works even though it is not mandatory.
Variable 'i' is declared within the body of lambda expression so don't confuse it with local variable of main method. i is declared and initialized to 10, i is incremented by 1 (i becomes 11) and finally value of 'i' is printed.

6.1.52 Answer: C

Reason :
--show-module-resolution shows module resolution output during startup.

On Windows platform, if you have modular jar file (C:\third-party\test.jar) for 'com.udayankhattry.modularity' module and class to

invoke is: com.udayankhattry.ocp1.Main,
then below command shows module resolution output:

```
C:\>java -p third-party --show-module-resolution -m
com.udayankhattry.modularity/com.udayankhattry.ocp1.Main
root com.udayankhattry.modularity file:///C:/third-party/test.jar
java.base binds jdk.localedata jrt:/jdk.localedata
java.base binds jdk.security.jgss jrt:/jdk.security.jgss
.
.
.
```

Please note, com.udayankhattry.modularity is the root module and as it implicitly requires java.base so you have other entries as well.

6.1.53 Answer: B

Reason :
Please note that Strings computed by concatenation at compile time are referred by String Pool. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc. This means the result of constant expression is calculated at compile time and later referred by String Pool.
Where as Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

`i1 + s1` is a constant expression which is computed at compile-time and results in a String object "1:ONE" referred by String Pool .

`i2 + s1` is not a constant expression because i2 is neither of primitive type nor of String type, hence it is computed at run-time and returns a non pool String object "1:ONE".

As, str1 refers to String Pool object, hence `str1 == "1:ONE"` returns true, where as str2 refers to non-Pool String object and hence `str2 == "1:ONE"` returns false.

6.1.54 Answer: D

Reason :
According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:

1. May not declare to throw any checked exception.
2. May declare to throw the same checked exception thrown by super class / interface method.
3. May declare to throw the sub class of the exception thrown by super class / interface method.
4. Cannot declare to throw the super class of the exception thrown by super class / interface method.
5. Cannot declare to throw unrelated checked exception.
6. May declare to throw any RuntimeException or Error.

default methods were added in Java 8 and TravelBlogger class correctly overrides the default method blog() of Blogger interface. Blogger class compiles successfully.

At Line n1, 'blogger' is of Blogger type (supertype) and it refers to an instance of TravelBlogger class (subtype), this is polymorphism and allowed in Java. Line n1 compiles successfully.
At Line n2, blog() method is being invoked on typecasting 'blogger' to TravelBlogger and as TravelBlogger class doesn't declare to throw any checked exception, hence Line n2 compiles successfully.

As instance is of TravelBlogger type, therefore on execution, Line n2 invokes blog() method of TravelBlogger instance, which prints TRAVEL on to the console.

6.1.55 Answer: D

Reason :
In the module descriptor file, if package specified in the exports directive is empty or doesn't exist, then compiler complains about it.
As mentioned in the question, directory 'satellite' is empty and directory 'artificial' doesn't exist and also there are no other java files under 'com.satellite' and 'com.satellite.artificial' packages, hence Line n2 and Line n3 cause compilation error.

6.1.56 Answer: C

Reason :
As per Java 8, default and static methods were added in the interface and as per Java 9, private methods were added in the interface. There is no issue in Shrinkable.java file.
class AntMan implements Shrinkable interface but as there is no

abstract method in Shrinkable interface, hence AntMan class is not needed to implement any method. AntMan.java file compiles successfully.
static method of Shrinkable interface can only be accessed by using Shrinkable.shrinkPercentage(). `AntMan.shrinkPercentage();` causes compilation error.

6.1.57 Answer: B

Reason :
module-info.java file is the module descriptor file, which is used to define the module name, its dependencies on other modules, its packages etc.
It cannot be empty, it must at least specify the module name. E.g.

```
module mymodule{
}
```

It is allowed to have empty module body (there is nothing between curly brackets {}). Please note all the java modules implicitly require java.base module.

6.1.58 Answer: B

Reason :
Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 10 compiles successfully.
No issues with Line 16 as method getReport() declares to throw SQLException and main(String []) method code correctly handles it .

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.

6.1.59 Answer: A

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and

index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

In Line n1, str infers to String type.

String class has length() method, which returns number of characters in the String object. So length() method returns 11.
String class has charAt(int index) method, which returns character at passed index. str.charAt(10) looks for character at index 10. index starts with 0. ! sign is at index 10.
Hence LIne n2 prints 11 : ! on to the console.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

6.1.60 Answer: B

Reason :
i++ == 2 || --i == 2 && --i == 2; [Given expression].
(i++) == 2 || --i == 2 && --i == 2; [Postfix has got higher precedence than other operators].
(i++) == 2 || (--i) == 2 && (--i) == 2; [After postfix, precedence is given to prefix].
((i++) == 2) || ((--i) == 2) && ((--i) == 2); [== has higher precedence over && and ||].
((i++) == 2) || (((--i) == 2) && ((--i) == 2)); [&& has higher precedence over ||].
Let's start solving it:
((i++) == 2) || (((--i) == 2) && ((--i) == 2)); [i=2, res=false].
(2 == 2) || (((--i) == 2) && ((--i) == 2)); [i=3, res=false].
true || (((--i) == 2) && ((--i) == 2)); [i=3, res=false]. || is a short-circuit operator, hence no need to evaluate expression on the right.
res is true and i is 3.

6.1.61 Answer: D

Reason :
If there is only one parameter in left part, then parentheses or round brackets '()' can be removed. If there is only one statement in the right-hand side then semicolon inside the body, curly brackets and return keyword(if available) can be removed.

Let's check all the options one by one:
ICalculator obj1 = x -> return x*x; ✗ If return keyword is used, then curly brackets and semicolon after the closing curly bracket must be used.

ICalculator obj2 = (x) -> return x*x; ✗ If return keyword is used, then curly brackets and semicolon after the closing curly bracket must be used.

ICalculator obj3 = x - > x*x; ✗ There should not be space between - and >.

ICalculator obj4 = x -> x*x; ✓ As there is only one parameter, hence parentheses or round brackets '()' can be removed. On right-hand side, there is only one statement, therefore, curly brackets, return keyword and semicolon inside the body are removed.

6.1.62 Answer: B

Reason :
"outer" and "inner" are valid label names.
On execution, control enters main method and creates int variable i.
On encountering do-while loop, control goes inside and initializes variable i to 5.
Then it executes while loop and it's boolean expression is always true.
System.out.println(i--); prints 5 to the console first, and then decrements the value of i by 1. So, i becomes 4.
Boolean expression of if(i == 4) evaluates to true. break outer; statement executes and takes the control out of do-while loop.
main method ends and program terminates successfully.
So, 5 gets printed only once.

6.1.63 Answer: C

Reason :
Boolean expression of do-while loop uses literal true (compile-time constant), hence Java compiler knows that this loop is an infinite loop. It also knows that once at runtime Java Control enters an infinite loop, none of the statements after loop block will get executed.
Hence it marks all the codes after the infinite do-while loop as Unreachable Code, which results in compilation error.

If boolean variable was used instead of boolean literal, then this program would have compiled and executed successfully.

```
public class Test {
    public static void main(String[] args) {
        boolean flag = true;
        do {
            System.out.println(100);
        } while (flag);

        System.out.println(200);
    }
}
```

Above program prints 100 in infinite loop and 200 never gets printed.

6.1.64 Answer: C

Reason :

Starting with JDK 11, it is possible to launch single-file source-code Programs.

If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
    (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

If you pass a file name with .java extension to java command, then java command consider the passed file as 'single-file source-code program'. But if any other file name is passed, then it considers the passed file as the class name. So, in this case, if you run the command `java Util.txt 10 20`, then java command searches for the class with the name 'Util.txt' and would thrown an error.

```
F:\>java Util.txt 10 20
Error: Could not find or load main class Util.txt
Caused by: java.lang.ClassNotFoundException: Util.txt
```

If the file does not have the .java extension, the --source option must be used to force source-file mode.

`java --source 11 Util.txt 10 20` instructs the java command to process the Util.txt as Java 11 source code. Two command line arguments ("10" and "20") are passed. args[0] refers to "10" and args[1] refers to "20" hence the expression args[0] + args[1] evaluates to "1020" .

Please note that source-file can contain package statement and multiple classes (even public) but first class found must have special main method 'public static void main(String [])'.

For more information on single-file source-code program please check the URL: https://openjdk.java.net/jeps/330

6.1.65 Answer: B

Reason :
Quotes class contains overloaded constructors: a no-argument constructor Quotes() and a parameterized constructor Quotes(String).
Quotes q1 = new Quotes(); invokes no-argument constructor. null is assigned to the property 'quote' of object referred by q1.
Quotes q2 = new Quotes("NEVER GIVE UP!"); invokes parameterized constructor, which assigns "NEVER GIVE UP!" to 'quote' of object referred by q2.
q1.display(); prints null.
Again we have same call q1.display(); which prints null.
NOTE: We haven't called display() on object referred by q2.

6.1.66 Answer: G

Reason :
`var x = new int[]{1};`: Right side expression creates an int[] object of one element and 1 is assigned to that element. Target-type (int[]) is specified on the right side, hence this statement compiles successfully. x infers to int[] type.
Similarly y and z also infer to int[] type.

System.out.println((x = y = z)[0] + y[0] + z[0]); //x --> {1}, y --> {2} and z --> {3}
=> System.out.println((x = y)[0] + y[0] + z[0]); //x --> {1}, y --> {3} and z --> {3}
=> System.out.println(x[0] + y[0] + z[0]); //x --> {3}, y --> {3} and z --> {3}
=> System.out.println(3 + 3 + z[0]);
=> System.out.println(6 + 3);

=> System.out.println(9);

9 is printed on to the console.

6.1.67 Answer: F

Reason :
Profit class causes compilation error as it complains about duplicate default methods: Profitable1.profit() and Profitable2.profit(). To rectify this error abstract class Profit must override the profit() method .

default keyword for method is allowed only inside the interface and default methods are implicitly public. So overriding method should use public modifier and shouldn't use default keyword.
If you want to invoke the default method implementation from the overriding method, then the correct syntax is: [Interface_name].super.[default_method_name].
Hence, `Profitable1.super.profit();` will invoke the default method of Profitable1 interface and `Profitable2.super.profit();` will invoke the default method of Profitable2 interface.

Based on above points, let's check all the options one by one:
No need for any modifications, code compiles as is: ✗

Replace /*INSERT*/ with below code:
double profit() {
    return 50.0;
}: ✗
profit() method must be declared with public access modifier.

Replace /*INSERT*/ with below code:
public default double profit() {
    return 50.0;
}: ✗
default keyword for method is allowed only inside the interface.

Replace /*INSERT*/ with below code:
protected double profit() {
    return 50.0;
}: ✗
profit() method must be declared with public access modifier.

Replace /*INSERT*/ with below code:

```
public double profit() {
    return Profitable1.profit();
}: ✗
```

Profitable1.profit(); causes compilation error as correct syntax is: Profitable1.super.profit();

Replace /*INSERT*/ with below code:

```
public double profit() {
    return Profitable2.super.profit();
}: ✓
```

It compiles successfully.

6.1.68 Answer: G

Reason :

default methods were added in Java 8. Class Chair correctly implements Sellable interface and it also overrides the default symbol() method of Sellable interface.

At Line n1, 'obj' refers to an instance of Chair class, so obj.symbol() and obj.getPrice() invoke the overriding methods of Chair class only. obj.symbol() returns "£" and obj.getPrice() returns 35.0

At Line n2, '+' operator behaves as concatenation operator and Line n2 prints £35.0 on to the console.

6.1.69 Answer: E

Reason :

Starting with JDK 11, it is possible to launch single-file source-code Programs.

If you execute 'java --help' command, you would find below option was added for Java 11:

```
java [options] <sourcefile> [args]
    (to execute a single source-file program)
```

Single-file program is the file where the whole program fits in a single source file and it is very useful in the early stages of learning Java.

The effect of `java Test.java` command is that the source file is compiled into memory and the first class found in the source file is executed. Hence, `java Test.java` is equivalent to (but not exactly same as):

```
javac -d <memory> Test.java
java -cp <memory> Test
```

As command-line argument is not passed, hence Line n1 throws ArrayIndexOutOfBoundsException (subclass of RuntimeException), handler is available in inner catch block, it executes Line n1 and prints INHALE- on to the console.
throw e; re-throws the exception.

But before exception instance is forwarded to outer catch-block, inner finally-block gets executed and prints EXHALE- on to the console.
In outer try-catch block, handler for RuntimeException is available, so outer catch-block gets executed and prints INHALE- on to the console.
After that outer finally-block gets executed and prints EXHALE- on to the console .

Hence, the output is: INHALE-EXHALE-INHALE-EXHALE

6.1.70 Answer: C

Reason :
Following are allowed in boolean expression of if statement:
1. Any expression whose result is either true or false. e.g. age > 20
2. A boolean variable. e.g. flag
3. A boolean literal: true or false
4. A boolean assignment. e.g. flag = true

boolean expression in this case is: (grade = 60), which is an int assignment and not boolean assignment. Hence Compilation error.

6.1.71 Answer: B

Reason :
'passportNo' should be read-only for any other class.
This means make 'passportNo' private and provide public getter method. Don't provide public setter as then 'passportNo' will be read-write property.
If passportNo is declared with default scope, then other classes in the same package will be able to access passportNo for read-write operation.

6.1.72 Answer: A

Reason :

Given statement:
System.out.println((flag = true) | (flag = false) || (flag = true)); //flag = false
System.out.println(((flag = true) | (flag = false)) || (flag = true)); //bitwise inclusive OR | has higher precedence over logical OR ||. flag = false
|| has two operands, Left: ((flag = true) | (flag = false)) and Right: (flag = true). Left operand needs to be evaluated first.
System.out.println((true | (flag = false)) || (flag = true)); //flag = true
System.out.println((true | false) || (flag = true)); //flag = false
System.out.println(true || (flag = true)); //flag = false
|| is a short-circuit operator and as left operand evaluates to true, hence right operand is not evaluated.
Above statement prints true on to the console.
And
System.out.println(flag); prints false on to the console as flag variable is false.

6.1.73 Answer: D

Reason :

Before you answer this, you must know that there are 5 different Person object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Person objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method has NOT been overridden by the Person class. In fact, equals(Person) is overloaded. But overloaded version is not invoked while equating the Person objects.

equals(Object) method defined in Object class is invoked and equals(Object) method defined in Object class uses == operator to check the equality and in this case as all the Person objects are stored at different memory location, hence not equal.

Nothing is removed from the persons list, all the 4 Person objects are printed in the insertion order.

6.1.74 Answer: A,C,D,H

Reason :
Variable NUM is declared in Super class and class Sub extends Super, hence NUM can be accessed by using obj.NUM.
But as NUM Is final, hence it cannot be reassigned, therefore Line n2 causes compilation error. Let's check all the options one by one:

Remove final modifier from Line n1 => ✓ Valid option and in this case output is 200.
Replace /*INSERT*/ with byte NUM; => ✗ In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as byte range is from -128 to 127 and 200 is out of range value.
Replace /*INSERT*/ with short NUM; => ✓ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to short type. In this case output is 200.
Replace /*INSERT*/ with int NUM; => In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to int type. In this case output is 200.
Replace /*INSERT*/ with float NUM; => ✗ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to float type. But output in this case will be 200.0 and not 200.
Replace /*INSERT*/ with double NUM; => ✗ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to double type. But output in this case will be 200.0 and not 200 .
Replace /*INSERT*/ with boolean NUM; => ✗ In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as boolean type in java allows 2 values true and false. 200 is not compatible with boolean type.
Replace /*INSERT*/ with Object NUM; => ✓ In this case, class Sub hides the variable NUM of Super class and at Line n2, value 200 is boxed to Integer, which is then assigned to obj.NUM. So, obj.NUM refers to an instance of Integer class. Line n3 invokes toString() method of Integer class and hence 200 is printed on to the console.

6.1.75 Answer: A

Reason :
Passed string's first character should be converted to upper case and rest of the characters should be converted to lower case. Combination of substring method with toUpperCase and toLowerCase method does the

trick. s.substring(1, 5) will throw StringIndexOutOfBoundsException for "hELP". There is no toCamelCase() method defined in String class.

6.1.76 Answer: C,D,F

Reason :
Local variable Type inference was added in JDK 10.
Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,
var x = "Java"; //x infers to String
var m = 10; //m infers to int

As the name suggests, Local variable Type inference is applicable only for local variables and not for instance or class variables. Hence, Line n1 and Line n2 cause compilation error.
For Line n3, 'list1' infers to ArrayList<Object> type and for Line n4, 'list2' infers to ArrayList. Both the lines n3 and n4 compile successfully.
var type cannot be the target type of lambda expressions and method references. Hence, Line n5 causes compilation error.
The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name. So, Line n6 compiles successfully.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

6.1.77 Answer: E

Reason :
Let us suppose test string is "aba".
Lambda expression s.toUpperCase().substring(0,1).equals("A"); means:
"aba".toUpperCase().substring(0,1).equals("A"); =>
"ABA".substring(0,1).equals("A"); => "A".equals("A"); => true.

This lambda expression returns true for any string starting with a (in lower or upper case). Based on the lambda expression, 5 array elements passes the Predicate's test and are printed on to the console.

6.1.78 Answer: C

Reason :
arr.length is 3, so outer loop executes 3 times. In 1st iteration of outer

loop, i=0. For 1st iteration of inner loop, i=0, j=0 and arr[0].length = 3. Inner loop runs for two iterations and breaks after printing "1 2 ". break; statement takes the control out of inner loop.
Control goes to step expression (i++) of outer loop, i becomes 1. Inner loop prints "4 5 " on to the console. Outer loop executes one more time and inner loop prints "6 7 " on to the console. Hence output is "1 2 4 5 6 7 ".

6.1.79 Answer: B,D

Reason :
Format of the java command related to module is:

```
java [options] -m <module>[/<mainclass>] [args...]
java [options] --module <module>[/<mainclass>] [args...]
```

(to execute the main class in a module)

--module or -m: It corresponds to module name. -m should be the last option used in the command. -m is followed by module-name, then by optional mainclass and any command-line arguments. If module is packaged in the jar and 'Main-Class' attribute is set, then passing classname after -m <module> is optional.

And below is the important option (related to modules) of java command:
--module-path or -p: It represents the list of directories containing modules or paths to individual modules. A platform dependent path separator (; on Windows and : on Linux/Mac) is used for multiple entries.
For example, java -p out;singlemodule;connector.jar represents a module path which contains all the modules inside 'out' directory, exploded module 'singlemodule' and modular jar 'connector.jar' .

Given command:

```
java -p out -m com.teaching.util/com.teaching.util.GradeCalculator
```

By looking at the -m option, it can be easily said that module name is 'com.teaching.util' and it contains a package 'com.teaching.util' and this package contains a class GradeCalculator.

As 'out' directory doesn't contain jar file, this means exploded module is available under 'out'. Hence, all the class files must be available under subdirectories of out directory

Hence below two statements are correct:
Module name is com.teaching.util
Class files must be available under subdirectories of out directory

Let's check other statements:
Module name is com.teaching.util.GradeCalculator ✗ It is a fully qualified class name and not the module name.

Source files must be available under subdirectories of out directory ✗ It is not a compulsion, source files can reside in other unrelated directory.

'out' directory must contain a directory named 'com.teaching.util' ✗ Not necessary if module was compiled using single-module mode.
For example,
Let's check below code structure:

```
C:
+---src
|   \---mymodule
|       |   module-info.java
|       |
|       \---com
|           \---teaching
|               \---util
|                       GradeCalculator.java
|
\---out
    \---mymodule
```

And below command successfully compiles the files:

```
C:\>javac -d out\mymodule
src\mymodule\com\teaching\util\GradeCalculator.java
src\mymodule\module-info.jav a
```

Structure after compilation:

```
C:
+---src
|   \---mymodule
|       |  module-info.java
|       |
|       \---com
|           \---teaching
|               \---util
|                       GradeCalculator.java
|
\---out
    \---mymodule
        |  module-info.class
        |
        \---com
            \---teaching
                \---util
                        GradeCalculator.class
```

Given java command executes fine:
C:\>java -p out -m com.teaching.util/com.teaching.util.GradeCalculator
Grade Calculator [It is because of the System.out.println("Grade Calculator"); statement inside main(String...) method of GradeCalculator class]

Clearly, Source files are not available under 'out' and 'out' directory doesn't contain 'com.teaching.util' directory.

6.1.80 Answer: B

Reason :
Parent (Super) class constructor is invoked by `super();` (all letters in lowercase) from within the constructor of subclass.
First statement inside no-argument constructor of Sub class is: `Super();` (Letter 'S' is in uppercase) and hence it causes compilation error.