Q1) Given

```
package pac;
public class Hello{
 public static void main(String[] args){
  Module module = Hello.class.getModule();
  System.out.println("Module: "+module);
  System.out.println("Name: "+module.getName());
  System.out.println("Descriptor: "+module.getDescriptor());
 }
}
```

Given the directory structure

```
\Test
| Hello.java
```

Given the commands to execute at the Test directory prompt:

```
Test>javac -d pac Hello.java
Test>java -cp pac pac.Hello
```

which statement is true?

A. Create an empty module-info.java file in the Test directory and on execution of the given commands the program prints:

```
Module: unnamed module @<</font><</font>hash code>>
Name: null
Descriptor: module-info
```

B. Execute java --module-path pac pac.Hello instead of java -cp pac pac.Hello execution the program prints:

```
Module : pac @<</font><</font>hash code>>
Name: pac.Test
Descriptor: null
```

C. On execution of the given commands, the program prints

```
Module: unnamed module @<</font><</font>hash code>>
Name: null
Descriptor: null
```

D. On execution of the given commands, the program prints:

```
Module: pac.Hello @<</font><</font>hash code>>
Name : unnamed
Descriptor: null
```

ANS : Option C

Q2) GIVEN :

```
import java.util.function.BiFunction;
public class Pair<T> {
final BiFunction<T, T, Boolean> validator;
T left = null;
T right = null;
private Pair() {
validator=null;

Pair (BiFunction<T, T, Boolean> v, T x, T y) {
```

```
validator = v;
set(x, y);

void set (T x, T y) {
if (!validator.apply(x, y)) throw new IllegalArgumentException () ;
setLeft (x);
setRight (y);

void setLeft (T x) {
left = x;

void setRight (T y) {
right = y;

final boolean isValid() {
return validator.apply(left, right);

}
}
```

It is required that if p instanceof Pair then p.isValid() returns true.
Which is the smallest set of visibility changes to insure this requirement is met?
A.setLeft and setRight must be protected.
B.left and right must be private.
C.isValid must be public.
D.left, right, setLeft, and setRight must be private.

ANS : Option B - left and right must be set private.

Q3) GIVEN :

```
public class App{
 var a = true; //line 1;
 {
  final var b = 10; // line 2
 }
 public static void main(String[] args){
  int var = 100; // line 3
  var b = "100";
  System.out.println(b);
 }
}
```

which statement is true?

A. the code fails to compile at line 3
B. the code prints 100
C. the code fails to compile at line 1
D. the code fails to compile at line 2
E. the code fails to compile at line 4
F. the code prints 10

Ans : Option C since var is not allowed in a field. (The line after var is instance initialiser)
If we remove line 1, then the code compiles and 100 is printed.

Q4) Which code fragment does a service use to load the service provider with a Print interface?

A. private Print print = new com.service.Provider.PrintImpl();
B. private Print print = com.service.Provider.getInstance();
C. private java.util.ServiceLoader<Print> loader = new java.util.ServiceLoader<>();
D. private java.util.ServiceLoader<Print> loader = ServiceLoader.load(Print.class);

ANS : Option D
EXPLANATION :

Option A: private Print print = new com.service.Provider.PrintImpl(); This line directly creates an instance of the PrintImpl class, which is not using Java's ServiceLoader for dynamic loading.
Option B: private Print print = com.service.Provider.getInstance(); This suggests a static method getInstance() which does not align with the typical usage of ServiceLoader for dynamic loading.
Option C: private java.util.ServiceLoader<Print> loader = new java.util.ServiceLoader<>(); This line initializes a ServiceLoader, but it's missing the step to actually load the service providers.
Option D: private java.util.ServiceLoader<Print> loader = ServiceLoader.load(Print.class); This line correctly uses the ServiceLoader to load service providers for the Print interface.

Q5) Which statement is true?

A. PrintWriter outputs and automatically flushes the stream
B. PrintStream outputs only bytes
C. System.exit() invokes the close() method for the InputStream/OutputStream resources.
D. Console.readPassword() method encrypts the text entered.

ANS : A. PrintWriter outputs characters and automatically flushes the stream.

Explanation: A. PrintWriter is a class in Java that outputs characters to an output stream and automatically flushes the stream when a newline character is written, when the println method is invoked, or when a carriage return character is written. B. System.exit() does not invoke the close() method for the InputStream/OutputStream resources. It terminates the currently running Java Virtual Machine (JVM). It's the responsibility of your code to close streams before exiting. C. Console.readPassword() does not encrypt the text entered. It simply disables echoing, so the password is not displayed on the console when it's entered. D. PrintStream does not output only bytes. It can output all primitive data types and strings using its print and println methods.

Q6) GIVEN :

```
/* line 1*/
A(){
 super("The Mandatory criteria yet to meet");
 }
}

15. public class TestCE{
16.  public static void main(String[] args) throws A{
17.   int a =10, b=13;
18.   try{
19.    if(a<b){
20.     throw new A();
21.     }
22.   }
```

```
23.   catch(Exception e){ System.out.println(e);}
24.   System.out.println("Continue..");
25.  }
26. }
```

You must define the A exception class. The program execution must be terminated if the condition at line 19 is true and an A exception is thrown at line 20.
Which code fragment at line n1 defines A as per the requirement?
A. class A extends Throwable
B. class A extends Exception
C. class A extends ArithmeticException
D. class A extends RuntimeException

ANS: Option A

Explanation :
A is correct for terminate program in line 20, others catch in exception clause an continue program

Q7) Given this declaration

```
@Target(TYPE)
@interface Resource{}
```

For which two kinds of declarations can the @Resource annotation be applied?

A. class declaration
B. method declaration
C. interface declaration
D. field declaration
E. local variable declaration

ANS: Option A, C

The @Target annotation is used to specify the kinds of program element to which an annotation type is applicable.
In this case, @Target(TYPE) means that the @Resource annotation can be applied to any type declaration. In Java, a type declaration is any class, interface, enum, or annotation type declaration. Therefore, the @Resource annotation can be applied to an class declaration (option A) and a interface declaration (option C). The @Resource annotation cannot be applied to a method declaration (option B), a field declaration (option D), or local variable declaration (option E) because these are not type declarations.

Q8) Given

```
 String s = "Oracle";
   Runnable r = ()-> {
       System.out.println(s);

   };
   s = "Java";
   Thread t = new Thread(r);
   t.start();
```

What is the result?
A. Java

B. Compilation error
C. Oracle
D. An exception is thrown at runtime

ANS: B. Compilation error

EXPLANATION:
Local variables declared in the enclosing method, including its formal parameters, can be accessed in a lambda expression provided and they are effectively final

```
String s = "Oracle";
Runnable r = () -> {
System.out.println(s);
};
//s = "Java";
Thread t = new Thread(r);
t.start()
```

will be ok since s="java" is commented here.

the error thrown is "variables in lambdas must be a final or effectively final". If s is modified then code not compile.

Q9) Given:

```
5. IntStream str = IntStream.of(2,3,4);
6. IntFunction<Integer> func = x -> y -> x*y;
7. str.map(func.apply(10)).forEach{System.out.println};
```

Which action will enable the code to compile?

A. replace line 6 with IntFuntion<UnaryOperator> func = x->y->x*y;
B. replace line 6 with Function<UnaryOperator> func = x->y->x*y;
C. replace line 6 with BiFunction<Integer> func = x->y->x*y;
D. replace line 6 with IntFunction<IntUnaryOperator> func = x->y->x*y;

ANS: Option D

EXPLANATION :
A. incompatible types : Intfunction is not a functional interface
B. requires two arguments, but only one is provided
C. requires three arguments, but only one is provided
D. The function.apply() expects IntFunction<IntUnaryOperator>

Q10) Given:

```
List original = new ArrayList<>(Arrays.asList(1,2,3,4,5));
```

Which two code fragmnets remove the elements from the "original" list?

A. Queue clq = new ConcurrentLinkedQueue<>(original);
for(Integer w : clq)
clq.remove(w);

B.
for(Integer w : al)
al.remove(w)

C. List cwa = new CopyOnWriteArrayList<>(original);
for(Integer w : cwa)
cwa.remove(w);

D. List sl = Collection.synchronizedList(original);
for(Integer w : sl)
sl.remove(w);

examtopic answer : AC

Q11) Given the code fragment from Box.java:

```
public class Box implements Serializable{
 private int boxId;
 private String size;
 private List items;
}
```

Given the code fragment from Item.java:

```
public class Item{
 private int id;
 private String name;
}
```

Given the information:

The classes Box and Item are encapsulated with getters and setters methods.
The classes Box and Item contains required constructors source code.

and the code fragment:

```
public static void main (String[] args) throws IOException {
List items1 = new ArrayList<>();
items1.add (new Item(1, "Pen") ) ;
items1.add (new Item(2, "Ruler") );
Box b1 = new Box (123, "s", items1);
try ( FileOutputStream fout = new FileOutputStream("boxser.txt") ;
ObjectOutputStream out = new ObjectOutputStream(fout) ;) {
out.writeObject (b1) ;
out. flush ();
out. close ();
} catch (Exception e) {
System.out.println ("Unable to Serialize") ;

}
}
```

Which action serializes the b1 object?

A. Handle NotSerializableException in the try clause or throw in the main() method definition.
B. Add SerialVersionUID to the Box and Item class.
C. Implement the Serializable interface in the Item class.
D. Override readObject() and writeObject() methods in the Book class.
E. Remove out.flush() method invocation.

ANS : C

Q12) Given

```
String[] words = {"am","am","first","second", "mismatch"};
Map map = Arrays.stream(words)
    .collect(Collectors.groupingBy(x->x, Collectors.counting())
System.out.println(map);
```

Taking into account that the order of the elements is unpredictable, what is the output?

A. {mismatch=1, am=2, first=1, second= 1}
B. {1=mismatch, 2=am}
C. {am=2,first=1,mismatch=1,second=2}
D. {mismatch=2,am=2,first=1,second=1}

ANS: Option A

Q13) Given

```
public class Employee{
 private String name;
 private String neighbourhood;
 private LocalDate birthday;
 private int salary;
}
```

and

```
List roster = new ArrayList<>(...);
Map<> m = roster.stream()
//line 1
```

Which code fragment on line 1 makes the m map contain the employee with the highest salary for each neighbourhood?

Options
A) .collect(Collectors.groupingBy(Employee :: getNeighborhood,
Collectors.maxBy (Comparator.comparing (Employee :: getSalary))));

B) .collect (Collectors.maxBy((x, y) -> y.getSalary() - x.getSalary(),
Collectors.groupingBy(Employee :: getNeighborhood)));

C) .collect(Collectors.groupingBy(e -> e.getNeighborhood(),
Collectors.maxBy((x, y) -> y.getSalary() - x.getSalary())));

D) .collect (Collectors.maxBy(Employee :: getSalary,

Collectors.groupingBy(Comparator.comparing (e ->
e.getNeighborhood())));

ANS: Option A

EXPLANATION:
maxBy() expects Comparator as the input.

Q14) Which two statements are correct about modules in java?

Options
A. module-info.java cannot be empty
B. module-info.java can be placed in any folder inside module-path
C. By default, modules can access each other as
as they run in the same folder
D. A module must be declared in the module-info.java file
E. java.base exports all of the java platforms core packages.

ANS: Options D,E

Explanation:
A. wrong coz module-info.java file CAN BE EMPTY but its generally considered good practice to include atleast the module declaration(eg ; module my.module). An empty descriptor indicates that the module exports no packages and has no explicit dependencies.

B. module-info.java file must be located in the source root of the module typically the same directory where the package structure starts to ensure proper compilation and integration within module system

C. wrong coz since java 9, modules cannot access other modules by default even if they reside in the same folder unless explicit declaration of dependencies are specified in the module-info.java file or granted access through automatic modules or the requires transitive directive.

Q15) public class Resource implements AutoCloseable{
```
    public Resource(){
      System.out.print("A");
    }
    @Override
    public void close(){
      System.out.print("B");
    }
    public void printResource(){
      System.out.print("C");
    }
   }

  class Tester{
     public static void main(String[] args){
try(Resource r = new Resource()){
  r.printResource();
}
finally{
 System.out.print("D");
```

```
 }
    }
   }
```

Ans : ACBD (try with resource is first called, hence A is printed. then printResource() is called, thus C is printed, then close() method is called and B is printed. atlast Finally block is called thus D is printed.

Q16) Given

```
class Seperators{
 public static String seperator = "/";
 public static String pathSeperator = ":";
}
```

To secure this code, you want to make sure that the client code cannot modify the public static fields.

Which code will accompolish this?

Options:

A)
```
abstract class Separators {
   public static String separator = "/";
   public static String pathSeparator =":";
```

B)
```
enum Separators {
   separator,
   pathSeparator
}
```

C)
```
interface Separators {
   String separator = "/";
   String pathSeparator = ":";
}
```

D)
```
class Separators{
   private static String separator = "/";
   private static String pathSeparator = ":";
}
```

ANS: C

Explanation:
In Java, interface variables are implicitly considered as constants and are therefore effectively final. By default, all variables declared in an interface are implicitly marked as public, static, and final, regardless of whether the public, static, or final modifiers are explicitly specified.

Q17) Given the declaration

```
@interface Resource {
  String[] value();
}
```

Examine this code fragment:

```
/*loc 1*/ class ProcessOrders(...)
```

Which two annotations may be applied at loc1 in the code fragment?

A. @Resource
B. @Resource("customer1")
C. @Resource()
D. @Resource({"customer1", "customer2"});
E. @Resource(value={{}})

ANS: Options B, D

Q18) Given:

```
public class Test{
    public static void main(String[] args) {
        String[] towns = {"boston","paris","bangkok","oman"};
        Comparator ms = (a,b)-> b.compareTo(a);
        Arrays.sort(towns,ms);
        System.out.println(Arrays.binarySearch(towns, "oman", ms));
    }
}
```

what is the result?
Options
A. 1
B. -1
C. 2
D. -3

ANS: Option A

Explanation:
The array is sorted in descending order since the code is b.compareTo(a) not a.compareTo(b) thus oman will be the second element (index 1)

Q19) Given these declarations:

```
String eName = "SMITH";
String empId = "42";
```

and these two code fragments:

Fragment 1 :

```
Statement stmt = conn.createstatement ();
String sql = "INSERT INTO EMP VALUES ('" + eName + "', '" + empId + "') ";
stmt.executeUpdate (sql) ;
```

Fragment 2:

```
String sql = "INSERT INTO EMP VALUES (?, ?)";
Preparedstatement pStmt = conn.prepareStatement (sql) ;
pStmt.setobject(1, eName, JDBCType. VARCHAR) ;
pStmt.setobject (2, empId, JDBCType. VARCHAR) ;
```

pStmt.executeUpdate () ;

Which code fragment is preferred and why?

A. Fragment 1 because it is shorter.

B. Fragment 2 because it explicitly specifies the SQL types of the column values.

C. Fragment 1 because it is more performant.

D. Fragment 2 because it prevents SQL injection.

ANS: Option D
EXPLANATION:
Using a prepared statement prevents SQL Injection vulnerabilities

Q20) Given the code fragment:

```
Path currentFile = Paths.get(“/scratch/exam/temp.txt”);
Path outputFile = Paths get(“/scratch/exam/new.txt”);
Path directory = Paths.get(“/scratch/”);
Files.copy(currentFile, outputFile);
Files.copy(outputFile, directory);
Files.delete (outputFile);
```

The /scratch/exam/temp.txt file exists. The /scratch/exam/new.txt and /scratch/new.txt files do not exist.

What is the result?
Options :

A. /scratch/exam/new.txt and /scratch/new.txt are deleted.

B. The program throws a FileaAlreadyExistsException.

C. The program throws a NoSuchFileException.

D. A copy of /scratch/exam/new.txt exists in the /scratch directory and /scratch/exam/new.txt is deleted.

ANS: Option C

Q21) Given:

```
class Item {
public String name;
public int count;
public Item(String name, int count) {
this.name = name;
this. count = count;
}
}
```

and the code fragment:

```
public class Test {
public static void main (String[] args){
```

```
var item = List.of (new Item("A", 10), new Item("B", 2),new Item("C", 12), new Item("D", 5), new Item("E", 6)) ;
// line 1

System.out.println ("There is an item for which the variable count is below zero.") ;
}
}
```

You want to examine the items list if it contains an item for which the variable count is below zero.

Which code fragment at line 1 will accomplish this?

A. if(item.stream().allMatch(i -> i.count < 0)) {
B. if(item.stream().anyMatch(i-> i.count < 0)) {
C. if(item.stream().filter(i-> i.count < 0). findAny()) {
D. if(item.stream().filter(i-> i.count < 0).findFirst()) {

ANS: Option B

Explanation :

option a doesnt print the statement
option c and d give the error saying incompatible types optional<item> cannot be converted to a boolean.

Q22) Given:

```
public class Main {
public static void main (string[] args){

List<Player> players = List.of (new Player("Scott", 115), new Player ("John", 70), new Player ("Jelly", 105));
double average = // line 1
System.out.println ("The average is: " + average) ;

}

}

class Player {
public String name;
public int score;
public Player (String name, int score) {
this.name = name;
this.score = score;
}
}
```

You want to calculate the average of the Player's score.

Which statement inserted on line 1 will accomplish this?

A) players.stream().average().orElse(0.0);
B) players.stream().mapToInt(a-> a.score).average().orElse(0.0);
C) players.stream().mapToDouble(a-> a.score).average();
D) players.stream().map(a-> a.score).average();

ANS: Option B

EXPLANATION:

A says cannot find method average in stream Player. average() method: The problem arises here. In Java, the average() method is not directly available on a stream of custom objects like Player. The average() method is typically used on numerical streams (e.g., IntStream, DoubleStream) to calculate the average of numeric values in the stream.

C raises an error saying optionaldouble cannot be converted to double. The issue with option C is that it doesn't handle the case of an empty stream or extract the average value from the OptionalDouble

D says cannot find method average in stream Integer. Because map(a -> a.score) returns a stream of Integer objects, the average() method cannot be directly applied to it. The correct approach, if you want to calculate the average of scores, is to use mapToInt(a -> a.score) to convert the stream to an IntStream before calling average()

Q23) Given the code fragment:

```
int x = 0;
do {
x++;
if (x == 1) {
continue;
}
System.out.println (x) ;
} while (x < 1);
```

What is the result?

A. 0

B. It prints 1 in infinite loop.

C. 1

D. The program prints nothing.

ANS: Option D

Explanation:

D, nothing gets printed because continue is triggered and x is not smaller than 1

Q24) Given:

```
public class Person {
private String name = "Joe Bloggs";
public Person (String name) {
this.name = name;
}
```

```
public String toString () {
return name;
}
}
```

and

```
public class Tester {
public static void main (String[] args){
Person pl = new Person (); // line 1
System.out.println (p1);
}
}
```

What is the result?

A. Joe Bloggs

B. P1

C. null

D. The compilation fails due to an error in line 1.

Ans : option D

Explanation:
Contructor without parameter does not exist
Remember : if you create a explicit constructor with parameters, you must create the constructor without parameter too if you want to use it.

Q25) Given

```
public class A {
    private boolean checkValue(int val) {
        return true;
    }
}

public class B extends A {
    public int modifyVal(int val) {
        if (checkValue(val)) {
            return val;
        } else {
            return 0;
        }
    }

    public static void main(String[] args) {
        B b = new B();
        System.out.println(b.modifyVal(10));
    }
}
```

What is the result?

Options:
A) 10
B) nothing
C) a java.lang.illegalargumentexception is thrown at runtime
D) it fails to compile
E) 0

ANS: Option D

EXPLANATION:
Output:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - Erroneous sym type: oracle.B.checkValue
at oracle.B.modifyVal(B.java:14)
at oracle.B.main(B.java:24)

since the code fails to compile.

Q26) Given

```
import java.util.ArrayList;

import java.util.Arrays;

public class NewMain {

public static void main(String[] args){

String[] catNames = { "abyssinian", "oxicat", "korat", "laperm", "bengal", "sphynx" };

var cats = new ArrayList<>(Arrays.asList(catNames));
cats.sort((var a, var b) -> -a.compareTo(b));
cats.forEach(System.out::println);
```

ANS:

```
sphynx
oxicat
laperm
korat
bengal
abyssinian
```

Q) Given

```
public static void main(String[] args){

List even =  List.of();
even.add(0,-1);
even.add(0,-2);
even.add(0,-3);
    System.out.println(even);
}
```

ANS: Runtime exception is thrown

Q) Why would you choose to use a peek operation instead of a forEach operation on a Stream?

ANS: to process the current item and return a stream

REASON: The peek operation is an intermediate operation that allows you to perform an action on each element of a stream as they are consumed from the resulting stream. Unlike the forEach operation, which is a terminal operation that consumes the entire stream and returns void, the peek operation returns a new stream that is identical to the original stream, with the additional side effect of the provided action.

Q) Given

```
public class main {
    public static void main(String[] args) {
        var lst = List.of (1, 2.0f, "4.0");
 for (var c : lst) {
System.out.print(">" + c);
    }

System.out.println();
lst.add(2, 3);                          //line 1

for (int c=0;c<lst.size(); c++) {
    display(lst.get(c));
        }}
public static void display(var c){       //line 2
    System.out.print("> " + c);
}
}
```

ANS: A compile time error occurs at line n2.

REASON: var is not allowed in parameter type

Q) Given:

```
public interface API {  //line 1
    public void checkValue(Object value) throws IllegalArgumentException; //line 2
    public boolean isValueANumber(Object val){
     if(val instanceof Number){
      return true;
     } else{
         try{
             Double.parseDouble(val.toString());
             return true;
         } catch(NumberFormatException ex)
         {
          return false;
         }

    }
}
```

which two changes should be done to make the class compile

ANS:

Change line 1 to an abstract class:
public abstract class API

Change line 2 to an abstract method
public abstract void checkValue(Object value) throws IllegalArgumentException;

```
1. public class ResourceTest{
    public static void main(String[] args) {
        final MyResource res1 = new MyResource();
        MyResource res2 = new MyResource();
        try(res1;res2){
            //do something

        }catch(Exception e){}
    }
    static class MyResource implements AutoCloseable{
     public void close() throws Exception{
     }
    }
}
```

ANS: The code compiles successfully.

2. Given the code fragment

```
class NoMatchException extends RuntimeException{}

public class Test{
 public static void main(String[] args){
  try{
   if("oracle".equals("ORACLE".toLowerCase())){
 throw new NoMatchException();
   }
   }
   catch (NoMatchException | NullPointerException npe){
    System.out.println("Exception 1");
   }
   catch (RuntimeException e){
 System.out.println("Exception 2");
   }
   catch(Exception e){
        System.out.println("Exception 3");
   }
   finally {
    System.out.println("Finally block");
   }
}
}
```

How many lines of text does this program print?

ANS: two

Exception1
finally block

3. Given:

```
public class Person{
 private String name;
 public Person(String name){
  this.name = name;
}
 public String toString(){
  return name;
}
}
```

and

```
public class Tester {
 static Person p = null;
 public static void main(String[] args){
  p = checkPerson(p);
  System.out.println(p);
  Person p1 = new Person("Joe");
  p1 = checkPerson(p);
  System.out.println(p1);
}

public static Person checkPerson(Person p) {
 if(p == null){
  p = new Person("Mary");
}
return p;
}
}
```

ANS:
Mary
Mary

4. Given

```
public class Person {
 private String name;
 private Person child;
 public Person(String name, Person child){
  this(name);
  this.child = child;
}
public Person(String name){
 this.name = name;
}
```

```
public String toString(){
 return name
}
}
```

and

```
public class Tester{
 public static void main(String[] args){
  Person jane = new Person("Jane");
  Person john = new Person("John",jane);
  return jane;
}
public static Person createPerson(Person person){
 person = new Person("Jack", person);
 /* line 1*/
 person = createPerson(person);
 /* line 2*/
 String name = person.toString();
 System.out.println(name);
}
}
```

Options:
A. The memory allocated for Jack object can be reused in line 2.
B. The memory allocated for Jane object can be reused in line 1.
C. The memory allocated for Jane object can be reused in line 2.
D. The memory allocated for John object can be reused in line 1.

ANS:

5. Given the directory structure:

```
-continent
 | a.txt
 |- country
    |b.txt
    |-state
    | c.txt
                    | + country
```

and

```
BiPredicate pred = (path, fileAttrs) -> {
  return fileAttrs.isDirectory();
};
int depth = 1;
try(var stream = Files.find(Paths.get("/continent"),depth,pred)){
  stream.forEach(System.out::println);
} catch(IOException e){}
```

What is the result?
ANS:

31.

```
public class Test1 {
    public static void main(String[] args) {
        String a ="10";
        try{
         int x =10;
         x= Integer.parseInt(a,2);//line 1
            System.out.println("X is "+x);

        }catch(NumberFormatException e){
            System.out.println("Error parsing value of"+x);//line 2
        }
    }
}
```

ANS: compilation error occurs due to line 2

34.

```
public class Main {
    public static void greet(String[] args) {
        System.out.println("hello");
        for(String arg: args){
            System.out.println("arg");
        }
    }
    public static void main(String[] args) {
        Main c =  null;
        c.greet();
    }

}
```

ANS: Hello

dereferncing null pointer error and uncompilable source code at c.greet() line 44.

```
class Animal {}
class Dog extends Animal{}
class Petdog extends Dog{}

public class House<A extends Animal>{
 public House<? super Dog> build(A a ){
  // CODE
 }
    public static void main(String[] args) {
        House h =new House();
        Dog d= new Dog();
        h.build(d);
    }
}
```

ANS :

```
return new House<Animal>();
return new House<Dog>();
```

WRONG:

```
return new House<PetDog>();
return new House<A>();
return new House<?>();
```

47.

```
public class Foo{
 public void foo(Collection arg){
     System.out.println("bonjour le monde");
 }
}
```

and

```
public class Bar{
public void foo(List arg){
    System.out.println("hello world");
}
    public static void main(String[] args) {
        List<String> li = new ArrayList<>();
        Collection<String> co =li;
        Bar b = new Bar();
        b.foo(li);
        b.foo((List) co);
    }
}
```

ANS: hello world!
     hello world!

49.

```
public class Test{
    public static void main(String[] args) {

 ExecutorService es = Executors.newCachedThreadPool();
 es.execute(()->System.out.println("ping "));
 //line 1
        System.out.println(future.get());
        es.shutdown();
}}
```

Options:

```
Future<String> future = new Callable(){
  public String call() throws Exception{
   return "pong";
  }
```

```
 }.call(); // error saying incompatible types, string cannot be converted to future

Future<String> future = es.execute(()->"pong");
        System.out.println(future.get()); // here unreported exception error is thrown InterruptedException
must be caught.

Future<String> future = es.submit(()->"pong");
        System.out.println(future.get()); // here unreported exception error is thrown InterruptedException
must be caught.

Future<String> future = es.invokeAny(new Callable<String>(){
     public String call() throws Exception{
      return "pong";
     }
 });
```

50.

```
public enum Status {
BRONZE(5),SILVER(10),GOLD(15);
private int rate;
private Status (int rate){
 this.rate=rate;
}
public int getRate(){return rate;}
public Status addStatus(int rate){
 return new Status(20); // error here saying enum types may not be instantiated
}
}
```

and

```
public class Test {
    public static void main(String[] args) {
        Status silver = Status.SILVER;
        System.out.println(silver+silver.getRate());
        Status platinum  = Status.addStatus(20);
        System.out.println(platinum+platinum.getRate());
    }
}
```

ANS: The compilation fails

Q)

```
public class Person1 {
    private String name;
    private int age;
    public Person1(String name, int age){
        this.name=name;
        this.age=age;

    }
    public int getAge(){
     return age;
```

```
    }
    public static void main(String[] args) {
        var persons = Arrays.asList(new Person1("Max", 18),new Person1("peter", 23),new
Person1("pamela", 23),new Person1("david", 12));
     int num = persons.stream().mapToInt(Person1::getAge).filter(p->p<20).reduce(0,(a,b)->a+b);
        System.out.println(num);
    }}
```

ANS: 30

Q) Given

```
public static void main(String[] args) throws InterruptedException {
        var c = new CopyOnWriteArrayList<>(List.of("1","2","3","4"));
        Runnable r = ()-> {
                try{
                    Thread.sleep(150);
                }
                catch(InterruptedException e){
                    System.out.println(e);
                }
                c.set(3,"four");
                System.out.println(c+" ");
        };
        Thread t = new Thread(r);
        t.start();
        for(var s:c){
            System.out.println(s+" ");
            Thread.sleep(100);
        }
    }
```

ANS: 1 2 [1, 2, 3, four] 3 4

Q) Given :

```
public interface InterfaceOne{
 public void methodA();
 public void methodB();
}

public interface InterfaceTwo extends AbstractClass{}

public abstract class AbstractClass implements InterfaceOne{
 public String origin = "Abstract Class";
 public void methodA(){
  System.out.println("A");
}
 public abstract void methodC();
}

public class ConcreteClass extends AbstractClass{
 public void methodC(String c){
  System.out.println(c);
}
```

}

Which three changes make this code compile?
A. implement methodC() in ConcreteClass
B. InterfaceTwo should no longer extend AbstractClass
C. Remove methodA() from AbstractClass
D. Add the keyword abstract to the methodA() and methodB() declarations in IterfaceOne
E. Implement methodA() in ConcreteClass
F. Remove methodA() from InterfaceOne
G. Implement methodB() in ConcreteClass

ANS: A, E, G

Q) Given

```
public class Reader {
    public static void main(String[] args) {
        char[] characters =  new char[100];
        try(FileReader reader = new FileReader("file_to_path");){
            reader.read(characters);
            System.out.println(String.valueOf(characters));
        } catch(IOException e){
         e.printStackTrace();
        }
    }
}
```

ANS: reader.read(characters);

Q) Given

```
List<String> states = List.of("NY","CA","WA","NC","CO");
states.forEach(s ->System.out.println(s)); //line 1
```

o/p:
NY
CA
WA
NC
CO

Which statement is equivalent to line 1?
q9

Q)

```
public interface ExampleInterface {
    static String origin = "Interface";
    void exampleMethod(String first);
}
public abstract class ExampleAbstractClass {
```

```
    static String origin = "Abstract class";
    abstract void exampleMethod(String first, String second);
}
public class ExampleClass  extends ExampleAbstractClass implements ExampleInterface{
    public void exampleMethod(String first){ }
    public void exampleMethod(String first, String second){ }
    public static void main(String[] args) {
        ExampleInterface theInstance = new ExampleClass();
       //line 1
    }
}
```

which option placed on line1 independently will cause compilation error?
ANS:

```
theInstance.exampleMethod(origin);
theInstance.exampleMethod(ExampleAbstractClass.origin, ExampleInterface.origin);
```

Q) Given

```
class Super{
 static String greeting() { return "Good Morning";}
 String name() {return "Harry";}
}

class Sub extends Super{
 static String greeting(){return "Good Morning";}
 String name(){return "Potter";}
}

public class Test {
    public static void main(String[] args) {
        Super s = new Sub();
        System.out.println(s.greeting() + ", "+s.name());
}
}
```

ANS: Good Morning, Potter

Q) Given:

```
import java.io.FileNotFoundException;

class ExSuper extends Exception {

private final int eCode;

public ExSuper (int eCode, Throwable cause) {
    super (cause);
    this.eCode = eCode;
}

public ExSuper (int eCode, String msg, Throwable cause) {

super (msg, cause);
```

```
this.eCode = eCode;
}
public String getMessage() {

return this.eCode+": "+super.getMessage()+"-"+this.getCause().getMessage();

}
}
public class ExSub extends ExSuper{

public ExSub(int eCode, String msg, Throwable cause) {
   super (eCode, msg, cause);
}

   public static void main(String[] args) {


try {

String param1 = "Oracle";

if (param1.equalsIgnoreCase("oracle")) {

throw new ExSub (9001, "APPLICATION ERROR-9001", new FileNotFoundException("MyFile.txt"));

}

throw new ExSuper (9001, new FileNotFoundException("MyFile.txt")); // Line 1

} catch (ExSuper ex) {

System.out.println(ex.getMessage());

}
}
}
```

ANS: 9001: APPLICATION ERROR-9001-MyFile.txt

Q) Given

```
public static void main(String[] args){
   File file1 = new File("filel.txt");

File file2 = new File("file2.txt");

try (BufferedReader reader = new BufferedReader(new FileReader(file1))) {

System.out.println(reader.readLine());
reader = new BufferedReader(new FileReader(file2));
System.out.println(reader.readLine());

} catch (IOException e) {
```

```
System.out.print("Error reading files");
}
   }
```

ANS: Error reading files in console is printed

Q Given:

```
public class X {
private Collection collection;
public void set (Collection collection) {
this.collection = collection;
}
}
```

and

```
public class Y extends X {
public void set (Map<string, string> map) {
super.set (map) ; // line 1
}
}
```

Which two lines can replace line 1 so that the Y class compiles? (Choose two.)

A. super.set(List map)
B. map.forEach((k, v) -> set(v)));
C. set(map.values());
D. set(map)
E. super.set(map.values());

ANS : Options C,E
EXPLANATION:
Option C: set(map.values()): This option retrieves the values of the map (map.values()) and passes them to the set method. Since map.values() returns a collection of values (Collection<String> in this case), it matches the parameter type Collection of the set method in class X.

Option E: super.set(map.values()): This option is similar to option C, but it explicitly calls the set method of the superclass (super.set) with the values of the map. Since map.values() returns a collection of values (Collection<String>), it matches the parameter type Collection of the set method in class X.

Q) Given:

```
var h = new HashMap () ;
String[] k = {"1", "2", null, "3" };
String[] v = { "a", "b", "c", null };
for (int i = 0; i < 4; i++) (
h.put(k[i], v[i]);
System.out.print(h.get(k[i]) + " ");
```

What is the result?

A. a b c followed by an exception

B. a b c null

C.abc

D. a b followed by an exception

ANS: Option B : a b c null

Q Given:

```
5. class A { }
6. class B extends A { }
7. class C extends B { }
8. public class Test {
9. public static void main (String args []) {
10. List <? extends A> listA = new ArrayList<> ();
11. List<B> listB = new ArrayList<B>();
12. List <? extends B> listC = new ArrayList<> ();
13. listA = listB;
14. listC = listB;
15. }
16. }
```

Which is true?

A. The program fails to compile on line 10.
B. The program fails to compile on line 13.
C. The program fails to compile on line 11.
D. The program compiles fine.

ANS: Option D
The program compiles fine

Q

Given:

```
abstract class Base {
abstract protected float getVal ();
}

public class Test extends Base {
public float getVal () { return Of; }
public long getVal () { return 2L; }
public static void main(String[] args) {
Test test = new Test ();
float f = test.getVal ();
System.out.println(f + test.getVal());
}
}
```

What is the output?

A. 2.0
B. 2
C. The compilation fails.

D. An exception is thrown at runtime.

ANS : Option C
Explanation:
The line
public long getVal () { return 2L; }

throws a error saying : method getVal() is already defined in class Test

same name and same signature is not allowed in Java even if they have different return types

it is possible to have different return type for a overriding method in child class, but child's return type should has co-variant type based on Liskov substitution principle.

Q

Given:

Your organization provides a cloud server to your customer to run their Java code. You are reviewing the changes for the next release and you see this change in one of
the config files:

old: JAVA_OPTS="$JAVA_OPTS-Xms8g -Xmx8g"
new: JAVA_OPTS="$JAVA_OPTS-Xms8g -Xmx8g -noverify"

Which is correct?

A. You accept the change because -noverify is necessary for your code to run with the latest version of Java.

B. You reject the change because -Xms8g -Xmx8g uses too much system memory.

C. You accept the change because -noverify is a standard option that has been supported since Java 1.0.

D. You reject the change because -noverify is a critical security risk.

ANS : Option D
EXPLANATION:
In Java, the -noverify option is used with the Java Virtual Machine (JVM) to skip bytecode verification during class loading. Bytecode verification is an integral part of Java's security model, as it ensures that the bytecode (compiled Java code) is safe to execute and adheres to certain rules defined by the Java Virtual Machine Specification.

Q
Given the code fragment:

```
public class App {
public static void main (String[] args) {
String str1 = "Java";
String str2 = new String ("java");
//line n1
{
System.out.println ("Equal") ;
} else {
System.out.println ("Not Equal") ;
}
}
}
```

Which code fragment, when inserted at line n1, enables the App class to print Equal?

A) Str 1. toLowerCase ();
if (str1 == str2)

B) if (str2. equals (str1. toLowerCase ()))

C) Str 1. toLowerCase ();
if (str1.equals (str1.toLowerCase ()))

D) if (str1.toLowerCase () == str2.toLowerCase ())

A. Option A

B. Option B

C. Option C

D. Option D

ANS: Option B
B is the correct option because == operator checks the same address and as str1 and str2 have different addresses but same content. whereas equals() checks the content, hence its right to use it.

Q)
Given:

```
public class Tester {
public static void main (String[] args) {
StringBuilder sb = new StringBuilder (5) ;
sb.append ("HOWDY") ;
sb.insert (0, ' ');
sb.replace (3, 5, "LL");
sb.insert (6, "COW");
```

```
sb.delete(2, 7);
System.out.println (sb. length ());
}
}
```

What is the result?

A. 5

B. 4

C. 3

D. An exception is thrown at runtime

ANS : Option B : 4
EXPLANATION:
HOWDY <-- sb.append("HOWDY");
HOWDY <-- sb.insert(0, "");
*HOLLY <-- sb.replace(3, 5, "LL");
*HOLLYCOW <-- sb.insert(6, "COW");
*HOW <-- sb.delete(2, 7);
4 <-- sb.length();

Q)
Given:

```
public class Foo{
private String a(){
return "Hello world!";
}
public string b() {
return a ();
}
}

public class Bar extends Foo {
protected String a (){
return "Bonjour le monde!";
}
}

public class Baz extends Bar {
public string b() {
return a ();
}
}
```

and

```
System.out.println(new Foo().b());
System.out.println(new Bar().b());
System.out.println(new Baz().b());
```

Options:

A. Hello world!
**NoSuchMethodError

B. Bonjour le monde!
Bonjour le monde!
Bonjour le monde!

C. Hello world!
Bonjour le monde!
Bonjour le monde!

D. Hello world!
Hello world!
Bonjour le monde!

E. Hello world!
Bonjour le monde!
Bonjour le monde!

ANS: Option D

Q)

Given:

```
LocalDate d1 = LocalDate.now() ;
d1.plusDays (1) ;
d1 = dl.minusMonths (2) ;
LocalDate d2 = d1.plusWeeks(3) ;
d2.minusDays (4) ;
d2 = null;
```

How many LocalDate objects are created in this example?

A. 2
B. 3
C. 4
D. 5

ANS: Option B
EXPLANATION:
OUTPUT:
2024-03-31
2024-01-31
2024-02-21
null

Q

Given the code fragment:

```
int i = 0;
for( ; i<10; i++) {
System.out.print (++i + " "); }
```

What is the result?

A. 2 4 6 8

B. 2 4 6 8 10

C. 1 3 5 7 9 11

D. 1 3 5 7 9

ANS: Option D

Q
Given the code fragment:

```
 var i =10;
        var j = 5;
        i+=(j*5+i)/j-2;
        System.out.println(i);
```

Whats the output?
ANS: 15

i+=(j*5+i)/j-2
i+=(5*5+i)/j-2
i+=(25+10)/j-2
i+=(35)/j-2
i+=35/5-2
i+=7-2
i+=5
i=i+5
i=10+5
i=15

Q

Given

```
class Scope{
static int myint=666;
public static void main(String[] args){
  int myint = myint;
  System.out.println(myint);
}
}
```

Which is true?

A. The code does not compile successfully
B. The code compiles but throws a runtime exception when run
C. The code compiles and runs successfully but with a wrong answer (i.e; a bug)
D. It prints 666

ANS: Option A
EXPLANATION:

A. local variable myint is not initialized.

Q

Given:

```
import java.sql.Timestamp;
public class Test{
    public static void main(String[] args) {
        Timestamp ts = new Timestamp(1);
    }

}
```

OUTPUT:
Test.class -> java.base
Test.class -> java.sql

Q

Which code fragment added to line 1 enables the code to compile and print Hello Joe?

A.

```
interface Greeting {
public default void greet (String name) {
System.out.println(greet+name);
}
}
```

B.

```
class Greeting {
public static void greet (String s) {
System.out.println("Hello "+ s);
}
}
```

C.

```
class Greeting {
private void greet (String name) {
System.out.println("Hello " + name);
}
}
```

D.

```
static class Greeting {
public void greet (String name) {
System.out.println("Hello " + name);
}
}
```

ANS: Option C

option b gives an error called modifier ’static’ is only allowed in constant variable declaration

Q) Given

```
String s1 = new String("Java");
String s2 = s1.intern();
StringBuilder sb1 = new StringBuilder("Java");
String s3 = sb1.toString();
System.out.println(s1 == s2);
System.out.println(s1.equals(sb1.toString()));
System.out.println(s2 == s3);
```

ANS: false true false

Q) Given:

```
import java.io .*;
public class Tester{
public static void main (String[] args){
try {
doA ();
doB ();
} catch(IOException e){
System.out.print("c");
return;
} finally{
System.out.print("d");
}
System.out.print("f");
}
private static void doA() {
System.out.print("a");
if (false){
throw new IndexOutOfBoundsException ();
}}
private static void doB() throws FileNotFoundException{
System.out.print("b");
if (true) {
throw new FileNotFoundException ();
}}}
```

ANS: abcd

Q) Given

```
int i = 10;
do {
for (int j = i/2; j> 0; j -- ) {
System.out.print(j + " ") ;
}
i -= 2;
} while (i> 0);
```

ANS: 5 4 3 2 1 4 3 2 1 3 2 1 2 1 1

Q) Given

```
List states =  new ArrayList(List.of("NY","CA","WA","NC","CO"));
//line 1
states.removeIf(function);
```

ANS: Predicate<String> function = s->s.contains("N");

Q) Given

```
public class Tester {
public static int reduce(int x) {
int y=4;
class Computer{
int reduce (int x) {
return x-y--;
}
}
Computer a = new Computer () ;
return a.reduce (x);
}
public static void main(String[] args) {
System.out.print(reduce(1));
}}
```

ANS: The compilation fails.

Q) Given:

```
public class Foo1{
        static void foo(){
        bar();
}
static void bar() throws FooException {
            throw new FooException();

}
public static void main(String[] args) {
try (StringReader r = new StringReader("how now brown cow")) {
bar();
System.out.println("A");
}
catch (FooException ex) {
foo();
System.out.println("B");
}
catch (Exception ex) {
System.out.println("C");
}
finally{
System.out.println("D");
}}}
```

ANS:

D
Exception in thread "main" oracle.FooException
 at oracle.Foo1.bar(Foo1.java:19)
 ... a stack trace is produced

Q) Given

```
public interface AdapterFirst{
 void showFirst();
}
```

ANS:

```
public abstract class MainClass implements AdaptorFirst{
    public void showFirst(){
        System.out.println("first");
    }
}

public class MainClass implements AdaptorFirst{
    public void showFirst(){
        System.out.println("first");
    }
}

public abstract class MainClass implements AdaptorFirst{
 public abstract void showFirst();
}
```

Q) Given:

```
public class LongFunctionTest {
public static void main (String[] args) {
LongFunction func = x -> x*x;
long test = func.apply(100);
System.out.println(test);
}}
```

ANS: The compilation fails

incompatible types, object cannot be converted to long

Q) Given:

```
public class Option {
public static void main (String[] args) {
System.out.println("Ans : " + convert("a").get ()) ;
}
private static Optional convert (String s) {
try {
return Optional.of (Integer.parseInt(s));
} catch (Exception e) {
```

```
return Optional.empty ();
}}}
```

ANS:

A java.util.NoSuchElementException is thrown at runtime.

Q) Given

Integer i=11;

ANS:

```
double d=i;
Double b = Double.valueOf(i);
```

Q)Given:

```
public class StrBldr {

    static StringBuilder sb1= new StringBuilder("yo ");
    StringBuilder sb2 = new StringBuilder("hi ");

    public static void main(String[] args) {
        sb1 = sb1.append(new StrBldr().foo(new StringBuilder("hey")));
        System.out.println(sb1);
                }
        StringBuilder foo(StringBuilder s) {
System.out.print(s + " oh " + sb2);
            return new StringBuilder("ey");
        }
}
```

ANS:

```
hey oh hi yo ey
```

Q) Given:

```
public class Worker {
private boolean ready = true;
public synchronized boolean isReady () {
return ready;
}
public synchronized void work (Worker other, Resource resource) {
while (ready) {
while (resource. owner != this) {
try {
wait (10);
}
catch (InterruptedException e) { }
}
if (other.isReady ()) {
resource.owner = other;
}
else {
// do work with resource
```

```
ready = false;
resource.owner = other;
}
}
}
public static void main(String[] args) {
Worker w1 = new Worker () ;
Worker w2 = new Worker ();
Resource r = new Resource () ;
Resource resource = new Resource();
resource.owner = w1;
new Thread ( () -> {
w1.work (w2, r);
} ) .start ();
new Thread ( () -> {
w2.work(w1, r) ;
} ) .start ();
}}
```

ANS: Livelock

Q) Given:

```
public class Test {
    void display(int i){
        System.out.println("one");
    }
    void display(long l){
        System.out.println("two");
    }
    public static void main(String[] args) {
        new Test().display(0B1010_0101_1001_0110);
    }
}
```

ANS: one

Q) Given:

```
enum Alphabet{
A,B,C;
}
```

Examine this code: System.out.println(Alphabet.getFirstLetter());

What code should be written at line 3 to make this compile?

ANS: static String getFirstLetter(){return A.toString();}

Q) Given:

```
public class Lines {
    public static void main(String[] args) {
    String fileName = "lines.txt";
    List list = new ArrayList<>();
```

```
   try(Stream stream = Files.lines(Paths.get(fileName))){
 list = stream
        .filter(line->!line.equalsIgnoreCase("JAVA"))
        .map(String::toUpperCase)
        .collect(Collectors.toList());
}
   catch(IOException e){ }
list.forEach(System.out::println);
}}
```

ANS:
C
C++
GO
KOTLIN

Q) Which declaration of an annotation type is legal?

ANS:

```
@interface Author{
   String name() default "";
   String date();
}
```

Q) Given

```
interface AbilityA {
default void action () {
System.out.println("a action");
}
}

interface AbilityB {
void action ();
}

public class Test2 implements AbilityA, AbilityB { // line 1
public void action() {
System.out.println("ab action");
}
public static void main(String[] args) {
AbilityB x = new Test2();
x.action();
// line 2
}
}
```

ANS: ab action

Q) Given:

```
public static void main(String[] args) {
int x=0,y=6;
for( ; x<y;x++,y -- ) {//line 1
if (x%2 == 0) {
```

```
continue;
}
System.out.println(x+"-"+y);
}}
```

ANS: 1-5

Q) Given:

```
public class CreateArrayListExample {
 public static void main(String[] args) {
  List vegetables = new ArrayList<>();
  vegetables.add("Kale");
  vegetables.add(0, "Lettuce");
  System.out.println (vegetables);
  List fish = new ArrayList<>();
  fish.add("Salmon");
  fish.add(0, "Seabass");
  System.out.println(fish);
}}
```

ANS:
[Lettuce, Kale]
[Seabass, Salmon]

Q) Given

```
public class main{
public static void main(String[] args) {
String s1 = new String ("Java");
String s2 = s1.intern();
StringBuilder sb1 = new StringBuilder("Java");
String s3 = sb1.toString ();
System.out.println(s1 == s2);
System.out.println(s1.equals(sb1.toString()));
System.out.println(s2 == s3);
}
}
```

ANS: false true false

Q) Given:

```
public class StrBldr{
static StringBuilder sb1 = new StringBuilder ("yo ");
static StringBuilder sb2 = new StringBuilder("hi ");

public static void main (String[] args) {
sb1 = sb1.append(new StrBldr().foo (new StringBuilder("hey")));
System.out.println (sb1);
}
StringBuilder foo (StringBuilder s) {
sb2 = sb2. append (s + " oh ");
return sb2;
}}
```

ANS: yo hi hey oh

Q) Given:

```
public class Person {
private String name = "Green";
public void setName (String name) {
String title = "Mr. ";
this.name = title + name;
}
public String toString () {
return name;
}
}

public class Test {
public static void main(String args[]) {
Person p = new Person ();
p.setName ("Blue");
System.out.println(p);
}}
```

ANS: Mr. Blue

Q) Given:

```
public enum Season{
WINTER('w'), SPRING('s'),SUMMER('h'),FALL('f');
char c;
private Season(char c){
 this.c=c;
}
}
```

ANS: System.out.println(sA[1]);
    System.out.println(Season.valueOf("SPRING"));
    System.out.println(Season.SPRING);

Q) Given:

```
public class Main {
public static void main (String[] args) {
List<String> fruits = List.of("banana ", "orange ", "apple ", "lemon ");
Stream<String> s1 = fruits.stream ();
Stream<String> s2 = s1.peek(i -> System.out.print(i+""));
System.out.println(" ----- ");
Stream<String> s3 = s2.sorted ();
Stream s4 = s3.peek(i -> System.out.print(i +""));
System.out.println(" ----- ");
String strFruits = (String) s4.collect(Collectors.joining(","));
}}
```

ANS:

---
banana orange apple lemon apple banana lemon orange

Q) Given:

```
public interface APIInterface{

 public default void process() { System.out.println("Process() called 1.");}
}

public abstract class AbstractAPI{
 public abstract void process();
}

public class ApiImpl extends AbstractAPI implements APIInterface {
public void process(){
    System.out.println("Process() called 2.");
}
    public static void main(String[] args) {
        var impl = new ApiImpl();
        impl.process();
    }
}
```

ANS: Process() called 2.

Q) Given:

```
public class Test{
static void add (List l) {
l.add(4);
l.add(3.14f);
}
public static void main (String[] args) {
var x = new ArrayList () ;
x.add(3);
add (x);
for (Integer i : x) {
System.out.print(i +"");
}}}
```

ANS: 3 4 3.14

Q) Given:

```
interface OrderService {
default void place(int numItems, int minItems) { }
private void verify(int minItems) {
System.out.println("Verified");
}}

public class Order implements OrderService {
public static void main(String[] args) {
Order order = new Order ();
```

```
order.place(10,5);
}}
```

ANS:

Invoke the verify method from the place method.

Q) Given:

```
class MyPersistenceData {
String str;
private void methodA () {
System.out.println("methodA") ;
}
}
```

You want to implement the java.io.Serializable interface to the MyPersistenceData class.

Which method should be overridden?

A. The readExternal and writeExternal method

B. Nothing

C. The readExternal method

D. The writeExternal method

ANS: Nothing

Explanation: The correct answer is B. Nothing.

In Java, when a class implements the ‘java.io.Serializable‘ interface, it indicates that instances of that class can be serialized, i.e., converted into a stream of bytes that can be saved to a file, sent over a network, or stored in a database, and later deserialized, i.e., reconstructed back into an object.

The ‘java.io.Serializable‘ interface itself doesn’t have any methods that you must implement. It’s known as a marker interface, meaning it’s used to mark classes that support serialization without requiring any specific methods to be implemented. When a class implements ‘Serializable‘, it signifies to the Java runtime that instances of that class can be serialized.

So, for the ‘MyPersistenceData‘ class to implement ‘Serializable‘, you don’t need to override any methods. Just implementing the interface is sufficient.

Q) Which two assignments create Locale instances?

```
locale = new locale ("en", "GB");
```

```
locale = Locale.getDefault();
```

```
locale = Locale.getAvailableLocales();
```

```
locale = "en-UsA";
```

Locale = "fr_FR";

ANS:
locale = new locale ("en", "GB");
locale = Locale.getDefault();

EXPLANATION:

locale = new Locale("en", "GB");: This line creates a new Locale instance with the language code "en" (English) and the country code "GB" (United Kingdom), specifying a specific locale for English in the UK.

locale = Locale.getDefault();: This line retrieves the default locale set in the Java Virtual Machine (JVM) where the code is running. It creates a Locale instance based on the system's default locale settings, which might vary depending on the system's language and region settings.

The other options are incorrect:

locale = Locale.getAvailableLocales();: This method returns an array of all available locales, not a single Locale instance. It doesn't create a Locale instance directly.

locale = "en-UsA"; and Locale = "fr_FR";: These assignments are trying to assign strings to the Locale variable, which is not valid syntax for creating a Locale instance.

Q) Which two statements are correct about modules in java?

a) by default, modules can access each other as long as they run in the same folder
b) java.base exports all of the java platforms core packages
c) module-info.java cannot be empty
d) a module must be declared in module-info.java file
e) module-info.java can be placed in any folder inside module-path

ANS:

java.base exports all of the java platforms core packages
A module must be declared in module-info.java file

Option A is wrong coz since java 9, modules cannot access other modules by default even if they reside in the same folder unless explicit declaration of dependencies are specified in the module-info.java file or granted access through automatic modules or the requires transitive directive.
Option c is wrong , bcoz module-info.java file CAN BE EMPTY but its generally considered good practice to include atleast the module declaration(eg ; module my.module). An empty descriptor indicates that the module exports no packages and has no explicit dependencies.
Option e is incorrect coz module-info.java file must be located in the source root of the module typically the same directory where the package structure starts to ensure proper compilation and integration within module system.

Q) Which two statements are true about running code on the classpath and module path?

A. A non-modular JAR placed on the -classpath results in an unnamed module.
B. A non-modular JAR placed on the --module-path results in a named application module.
C. A modular JAR placed on the -classpath results in a named application module.
D. A modular JAR placed on the -classpath results in an automatic module.
E. A modular JAR placed on the --module-path results in a named application module.

ANS: A, E

A non-modular JAR placed on the -classpath results in an unnamed module.
A modular JAR placed on the --module-path results in a named application module.

EXPLANATION:

A. When you place a non-modular JAR on the classpath (-classpath), it results in an unnamed module. This is because the classpath doesn't have the concept of modules; it treats everything as part of the classpath without modularization.

B. This statement is incorrect because placing a non-modular JAR on the --module-path doesn't result in a named application module. The --module-path is used for modular JARs and directories containing modules, not for non-modular JARs.

C. Placing a modular JAR on the classpath doesn't result in a named application module. The classpath doesn't handle modules in the same way as the module path (--module-path).

D. Placing a modular JAR on the classpath doesn't result in an automatic module. Automatic modules are created when a modular JAR is placed on the module path (--module-path). Automatic modules are created when a modular JAR is placed on the module path (--module-path), not on the classpath. When a modular JAR is placed on the module path, the module system generates an automatic module name based on the JAR file's name, and it becomes an automatic module.

E. This statement is correct. When you place a modular JAR on the module path (--module-path), it results in a named application module. The module path is specifically designed to handle modular JARs and create named modules from them.

Q) Which three initialization statements are valid?

A. var loc = Set.of ("UK", "US", "UK") ;

B. var loc = List.of ("UK", null, "US");

C. var loc = ArrayList.of ("UK", "US");

D. var loc = List.of ("UK", "US");

E. var loc = Set.of("UK", "US");

F. var loc = Arrays.of ("UK", "US", "ES");

G. var loc = Map.of ("UK", 1, "US", 2);

ANS:

var loc = List.of ("UK", "US"); //creates an immutable list
var loc = Set.of("UK", "US"); //creates an immutable set
var loc = Map.of ("UK", 1, "US", 2); //creates an immutable map with keys "UK" and "US", and corresponding values 1 and 2

EXPLANATION:

- A. 'var loc = Set.of ("UK", "US", "UK");': This statement tries to create a set with duplicate elements ("UK" is repeated), which is not allowed in a set. The 'Set.of' method throws an

‘IllegalArgumentException‘ if duplicate elements are provided.

- B. ‘var loc = List.of ("UK", null, "US");‘: This statement tries to create a list with a ‘null‘ element, which is not allowed in a list created by ‘List.of‘. The ‘List.of‘ method throws a ‘NullPointerException‘ if a ‘null‘ element is provided.

- C. ‘var loc = ArrayList.of ("UK", "US");‘: There is no ‘ArrayList.of‘ method in Java standard libraries. To create an ‘ArrayList‘ with initial elements, you would typically use the ‘ArrayList‘ constructor and then add elements using ‘add‘ method calls.

- F. ‘var loc = Arrays.of ("UK", "US", "ES");‘: There is no ‘Arrays.of‘ method in Java standard libraries. To create an array or a list from elements, you would typically use ‘Arrays.asList‘ for arrays or ‘List.of‘ for lists.

Q) Given:

```
List<String> list1 = new ArrayList<>();
list1.add("A");
list1.add("B");
List<String> list2 = Collections.unmodifiableList(list1);
list1.add("C");
System.out.println(list1);
System.out.println(list2);
```

OUTPUT:

```
[A,B,C]
[A,B,C]
```

EXPLANATION:

It’s because Collections.unmodifiableList(list1) creates an unmodifiable view of list1, not a separate copy of the list. Therefore, changes made to list1 after creating list2 are reflected in both lists (list1 and list2).

Here’s what’s happening step by step:

You create an ArrayList named list1 and add elements "A" and "B" to it.
You create list2 using Collections.unmodifiableList(list1), which creates an unmodifiable view of list1. Both list1 and list2 are referring to the same underlying list.
You add "C" to list1.
When you print list1, it contains "A", "B", and "C" because you added "C" to it.
When you print list2, it also contains "A", "B", and "C" because list2 is an unmodifiable view of list1, so any changes made to list1 are reflected in list2.

Q) Given :

```
String s1 = new String ("Java") ;
String s2 = s1.intern();
StringBuilder sb1 = new StringBuilder ("Java");
String s3 = sb1.toString ();
System.out.println(s1 == s2);
System.out.println(s1.equals(sb1.toString()));
System.out.println(s2 == s3);
```

What is the output?

ANS:

false
true
false

EXPLANATION:

String s1 = new String("Java");: This creates a new String object with the value "Java" and assigns it to s1.

String s2 = s1.intern();: This calls the intern method on s1, which checks if there is already an equivalent String object in the string pool. If yes, it returns that object; otherwise, it adds s1 to the string pool and returns s1. In this case, "Java" already exists in the string pool, so s2 refers to the same object as s1 (which is in the heap, not the string pool).

StringBuilder sb1 = new StringBuilder("Java");: This creates a StringBuilder object with the value "Java" and assigns it to sb1.

String s3 = sb1.toString();: This converts the contents of sb1 to a String using toString() and assigns it to s3.

Now, let's analyze the output statements:

System.out.println(s1 == s2);: This compares s1 and s2 for reference equality. Since s2 refers to the same object as s1 (both "Java" literals in this case), the output is false. This is because == checks if the two variables refer to the exact same object in memory, not just if they have the same value.

System.out.println(s1.equals(sb1.toString()));: This compares the content of s1 (which is "Java") with the content of sb1.toString() (also "Java"). Since equals() compares the actual contents of the strings, the output is true.

System.out.println(s2 == s3);: This compares s2 (which refers to the "Java" literal in the string pool) with s3 (which refers to a new String object created by sb1.toString()). These are two different objects, so the output is false.

Q) Given:

```
public class StrBldr {
static StringBuilder sbl = new StringBuilder ("yo ") ;
static StringBuilder sb2 = new StringBuilder("hi ");

public static void main (String[] args) {
sbl = sbl.append(new StrBldr().foo (new StringBuilder("hey")));
System.out.println (sbl) ;
}

StringBuilder foo (StringBuilder s) {
sb2 = sb2.append (s + " oh ");
return sb2;
}
}
```

OUTPUT:

yo hi hey oh

Q) Given:

```
@Target ({TYPE, METHOD})
@interface Resource {}

/* Loc1 */ class Manager extends /* Loc2 */ Person {
/* Loc3 */ Manager() ( ... }
/* Loc4 */ String getDepartmentName () { ... }
/* Loc5 */ String departmentName;
```

In which two locations is it legal to apply the @Resource annotation?

ANS: Loc 1, Loc 4
Since TYPE can be applied to class,enum,interface,annotations. METHOD element type can be applied to method declarations.

Q) Given:

```
@interface Resource {
String value() default "Customer1";
```

Examine this code fragment:

```
/* Loc1 */ class ProcessOrders { ... }
```

Which two annotations may be applied at Loc1 in the code fragment?

```
@Resource

@Resource ("Customer2")

@Resource({"Customer2"})

@Resource (Value="Customer2")

@Resource(val="Customer2")
```

ANS: @Resource , @Resource("Customer2")

Q) Given:

13. Given:

```
class MyType<T> {
private T value;
public T getValue () {
return value;
}
public void setValue (T value) {
this.value = value;
}
}
```

```
public class Test {
public static void main (String ... args) {
MyType<String> strType = new MyType<>();
MyType <? extends Number> type = new MyType<>();
strType.setValue("test");
type.setValue(null);
System.out.println(strType.getValue() + ":" + type.getValue());
}
}
```

What's the result?

A. test:null
B. null:null
C. An exception is thrown at runtime
D. test:0
E. The compilation fails

ANS:
test:null

EXPLANATION:

In the Test class's main method:

MyType<String> strType = new MyType<>(); creates an instance of MyType with the type parameter String and assigns it to strType.

MyType<? extends Number> type = new MyType<>(); creates an instance of MyType with a wildcard that extends Number, allowing it to hold any subtype of Number, and assigns it to type.

strType.setValue("test"); sets the value of strType to "test".

type.setValue(null); sets the value of type to null.

System.out.println(strType.getValue() + ":" + type.getValue()); prints the values of strType and type, which are "test" and null, respectively.
Therefore, the output is "test:null".

There's no exception thrown because setting a value to null is allowed for any reference type in Java, including generic types like MyType<T>.

Q) Given:

```
public interface ExampleInterface{ }
```

Which two statements are valid to be written in this interface?

A.
```
public void methodF() {
  System.out.printin("F");
```

B. public int x;

C. public String methodD();

D. public abstract void methodB();

E. private abstract void methodC();

F. final void methodE();

G. final void methodG()
   System.out.println("G");

ANS:
public String methodD();
public abstract void methodB();

Q) Given:

public interface A {
abstract void x();
public default void y() { }
}
and

public abstract class B {
public abstract void z ();
}

and

public class C extends B implements A {
/* insert code here */
}

What code inserted into class c would allow it to compile?

A.
public void x() { }
protected void y () { super. y(); }
public void z() { }

B.
void x() { }
public void z() { }

C.
void x() { super.y(); }
public void z(){ }

D.
public void x(){ }
public void z(){ }

E.
void x(){ }
public void y(){ }

public void z(){ }

ANS:
public void x(){ }
public void z(){ }

EXPLANATION:

All the abstract methods in the interface and abstract class must be implemented in the concrete class in order for the class to compile.

Q) Given:

```
IntStream.range(1,4)
            .peek(System.out::print)
            .peek(i->{
               if(i==3)
                  throw new RuntimeException("Exception thrown");
            });
```

ANS:
The program prints nothing

EXPLANATION:

The code you provided doesn't print anything because you're missing a terminal operation on the IntStream. In Java streams, intermediate operations like peek are lazy and only get executed when a terminal operation is present.
To make your code print something, you need to add a terminal operation such as forEach to consume the stream elements.

If forEach() operation is included then output is 1 2 3 Runtime exception, bcoz

IntStream.range(1, 4) creates a stream of integers from 1 to 3.
.peek(System.out::print) uses peek to print each element of the stream. So, it prints "123".
.peek(i -> { if (i == 3) throw new RuntimeException("Exception thrown"); }) uses another peek to check if i is equal to 3. When i is 3, it throws a RuntimeException.
.forEach(i -> {}); is a terminal operation that consumes the stream elements but does nothing with them.
Since there's no handling for the RuntimeException within the stream pipeline, it propagates out of the stream and causes the program to terminate with an unhandled exception.

Q) Given:

```
interface MyInterface1 {
public int method() throws Exception;
private void pMethod() { /* an implementation of pMethod */ }

interface MyInterface2 {
public static void sMethod() { /* an implementation of sMethod */ }
public boolean equals ();

interface MyInterface3 {
public void method ();
```

```
public void method (String str);

interface MyInterface4 {
public void dMethod() { /* an implementation of dMethod */ }
public void method ();

interface MyInterface5 {
public static void sMethod ();
public void method (String str) ;
```

Which two interfaces can be used in lambda expressions?

ANS: MyInterface1, MyInterface2

EXPLANATION:
Only functional interfaces can be used by a lambda expression. a functional interface has onluy one abstract method in it. only interface 1,2 satisfy that condition.

Q) Which two var declarations are correct?

A. var a;

B. var var = "hello";

C. var y = null;

D. var _ = 100;

E. var names = new ArrayList<>();

ANS: var var = "hello";
var names = new ArrayList<>();

EXPLANATION:

var should be initialised when declared.
var cannot be initialised to null since java cannot infer the type from null value
underscore(_) is a reserved identifier in Java, hence compiler error is thrown.

Q) Given:

```
public class Tester {
private int x;
private static int y;
public static void main (String[] args) {
Tester t1 = new Tester ();
t1.x = 2;
Tester.y = 3;
Tester t2 = new Tester () ;
t2.x = 4;
t2.y = 5;
System.out.printin(t1.x+","+t1.y);
System.out.printin (t2.x+", "+Tester.y) ;
System.out.printin(t2.x+","+t1.y);
```

ANS:

2,5
4,5
4,5

EXPLANATION:

Since variable y is declared static, the value of y should be constant throughout the execution. Since the last set value of y is 5, it is followed throughout the code.

Q) Given:

```
public class Main {
    public static void greet(String... args) {
        System.out.print("Hello ");
        for (String arg : args) {
            System.out.println(arg);
        }
    }

    public static void main(String[] args) {
        Main c = null;
        c.greet();
    }
}
```

What will be the output or behavior of the following Java code snippet?

A. A compilation error occurs.
B. "Hello" is printed.
C. NullPointerException is thrown at line 4.
D. NullPointerException is thrown at line 10.

ANS: Hello is printed

EXPLANATION:

Despite the fact that c is null, Java allows calling static methods using a null reference. So, when c.greet(); is executed, it calls the greet method, which prints "Hello" on the console because no arguments are passed to the method.

NullPointerException is thrown at line 4:
This is incorrect because the NullPointerException would occur if a method was called on a null object reference, but in this case, a static method is called using a null reference, which does not result in a NullPointerException.

NullPointerException is thrown at line 10:
Calling a static method using a null reference does not result in a NullPointerException.

Q) Which three initialization statements are correct?

```
int[][][] e = {{1, 1, 1}, {2, 2, 2}} ;
```

```
int x = 12_34;

short sh = (short) ’A’;

float x = 1f;

byte b = 10;
char c = b;

String contact# = "(+2) (999) (232)";

boolean false = (4 != 4);
```

ANS:

```
int x = 12_34;
short sh = (short) ’A’;
float x = 1f;
```

Q) Given:

```
public class Tester {
public static void main(String[] args) {
float x =2,y=4,z=4;
float a = y / x, b = y / z;
if (a > b) {
System.out.println(a + b);
}
}
}
```

What is the result?

An exception is thrown at runtime.
The program prints nothing.
1.0
2.0
3.0

ANS:
3.0

Q) Given:

```
8. public class Test{
9.  private final int x = 1;
10.  static final int y;
11.  public Test(){
12.   System.out.print(x);
13.   System.out.print(y);
14.  }
15.  public static void main(String[] args){
16.   new Test();
17.  }
```

18. }

What is the result?
A. The compilation fails at line 16.
B. The compilation fails at line 9.
C. The compilation fails at line 13.
D. 10
E. 1

ANS:
The compilation fails at line 13.

Q) Given:

```
public class Person {
private String name;
private int age;
public Person(String name, int age) {
this.name = name;
this.age = age;
}
public int getAge () {
return age;
}
public static void main (String args []) {
var persons = Arrays.asList (new Person ("Max", 18),
new Person ("Peter", 23),
new Person ("Pamela", 23),
new Person("David", 12));

int num = persons.stream()
   .mapToInt(Person::getAge)
   .filter(p -> p < 20)
   .reduce(0, (a,b) -> a + b) ;
System.out.println (num) ;
}
}
```

ANS: 30

Q) Given:

```
public class Person {
private String name = "Green";
public void setName(String name) {
String title = "Mr. ";
this.name = title + name;
}
public String toString() {
return name;
}
}
```

and

```
public class Test {
public static void main (String args[]) {
Person p = new Person();
p.setName("Blue");
System.out.println(p);
}
}
```

What is the result?

Mr. Blue
Green
An exception is thrown at runtime.
Mr. Green

ANS: Mr. Blue

EXPLANATION:

An instance of the Person class is created in the Test class's main method using Person p = new Person();
The setName method of the Person class is called with the argument "Blue" using p.setName("Blue");.
Inside the setName method, the title variable is initialized with the value "Mr. ".
Then, the name instance variable of the Person object (this.name) is set to title + name, which is "Mr. " + "Blue" = "Mr. Blue".
Finally, the toString method of the Person class is implicitly called when System.out.println(p); is executed, resulting in the output "Mr. Blue".

Q) Given:

```
List<Integer> numbers = List.of(2, 3,0,8,1,9,5,7,6,4);
int sum = numbers.stream().reduce(0, (n, m) ->n+m); // line 1
```

You want to make the reduction operation parallelized.
Which two modifications will accomplish this?

A. Replace line 1 with int sum = numbers.stream().iterate(0, a -> a+1).reduce (0, (n, m) -> n + m) ;

B. Replace line 1 with int sum = numbers.parallelStream().reduce (0, (n, m) -> n + m);

C. Replace line 1 with int sum = numbers.stream().flatMap(a -> a).reduce(0, (n, m) -> n + m) ;

D. Replace line 1 with int sum = numbers.stream().parallel().reduce(0, (n, m) -> n + m);

E. Replace line 1 with int sum = numbers.parallel().stream().reduce(0, (n, m) -> n + m) ;

ANS:

Replace line 1 with int sum = numbers.parallelStream().reduce (0, (n, m) -> n + m);

Replace line 1 with int sum = numbers.stream().parallel().reduce(0, (n, m) -> n + m);

Q) Given:

```
package test.t1;
public class A {
public int x = 42;
  protected A() {}              // line 1
}
```

and

```
package test.t2;
import test.t1 .*;
public class B extends A {
  int x = 17;                   // line 2
  public B() { super (); }      // line 3
}
```

and

```
package test;
import test.t1 .*;
import test.t2 .*;
public class Tester {
public static void main (String[] args) {
A obj = new B();  //line 4
System.out.printin (obj.x);     // line 5
}
}
```

What is the result?

A. 17
B. The compilation fails due to an error in line 1.
C. The compilation fails due to an error in line 3.
D. The compilation fails due to an error in line 5.
E. 42
F. The compilation fails due to an error in line 2.
G. The compilation fails due to an error in line 4.

ANS:
42

Q) Which statement is true?

A. Console.readPassword() method encrypts the text entered.

B. PrintWriter outputs characters and automatically flushes the stream.

C. PrintStream outputs only bytes.

D. System.exit () invokes the close () method for the InputStream/OutputStream resources.

ANS:
PrintWriter outputs characters and automatically flushes the stream.

Q) Given:

```
ExecutorService es = Executors.newCachedThreadPool ();
es.execute(() -> System.out.print ("Ping "));
// line 1
System.out.println(future.get()); // line 2
es. shutdown () ;
```

Which statement at line 1 will print Ping Pong?

A.
```
Future<String> future = es.execute (() -> "Pong") ;
```

B.
```
Future<String> future = es.submit (() -> "Pong") ;
```

C.
```
Future<String> future - new Callable () {
   public String call() throws Exception {
   return "Pong";
   }
 }.call();
```

D.
```
Future<String> future = es.invokeAny (new Callable<String> () {
     public String call () throws Exception;
     return "Pong";
   }
  });
```

ANS:
```
Future<String> future = es.submit (() -> "Pong");
```

Q) Given:

```
public class Person {
private String name;
public Person (String name) {
this.name = name;
}
public String tostring () {
return name;
}
}
```

and

```
public class Tester {
static Person p = null;
public static void main (String[] args) {
p = checkPerson (p) ;
System.out.println(p);
Person p1 = new Person("Joe");
p1 = checkPerson(p);
System.out.println(p1);
}
public static Person checkPerson (Person p) {
if (p == null) {
p = new Person ("Mary") ;
}
return p;
```

```
}
}
```

OUTPUT:

```
Mary
Mary
```

Q) Given:

```
public class App {
// line 1
public static void main(String[] args) {
new App().new Greeting().greet("Joe");
}
}
```

Which code fragment added to line 1 enables the code to compile and print Hello Joe?

A.
```
class Greeting {
public static void greet (String s) {
System.out.println("Hello "+ s);
}}
```

B.
```
interface Greeting {
public default void greet (String name) {
System.out.println (greet+name) ;
}}
```

C.
```
class Greeting {
private void greet(String name) {
System.out.println ("Hello " + name) ;
}}
```

D.
```
static class Greeting{
public void greet(String name){
System.out.println("Hello "+ name);
}}
```

ANS:

```
class Greeting {
private void greet(String name) {
System.out.println ("Hello " + name) ;
}}
```

Q)
```
class MyType<T>{
 private T value;
 public T getValue(){
```

```
  return value;
 }
  public void setValue(T value){
   this.value=value;
  }
}
public class Test2 {
    public static void main(String[] args) {
        MyType<String> strType = new MyType<>();
        MyType<? extends Number> type = new MyType<>();
        strType.setValue("test");
        type.setValue(null);
        System.out.println(strType.getValue()+":"+type.getValue());
    }
}
```

ANS:- test:null

Q) Given:

```
public class Color {
String hue;
int value;
public Color(String hue, int value) {
this.hue = hue;
this.value = value;
}
public String toString() {
return this.hue + ":" + this.value;
}
public static void main(String[] args) {
List clrs = List.of(new Color("Red", 100),
new Color("Yellow", 50),
new Color("Red", 75),
new Color("Yellow",75));

Comparator hueSrtr = (h1, h2) -> h1.hue.compareTo(h2.hue) ;
Comparator valueSrtr = (h1, h2) -> { if (h1.value >= h2.value) {
return 1;
} else {
return -1;
}};
clrs.sort (hueSrtr.thenComparing(valueSrtr));
System.out.println(clrs);
}}
```

What’s the result?

ANS: Runtime exception is thrown.