Answers of Full mock exam OCA – 1Z0 -808

ME-Q1) Given the following definition of the classes Animal, Lion, and Jumpable, select the correct combinations of assignments of a variable that don't result in compilation errors or runtime exceptions (select 2 options).

```
interface Jumpable {
}

class Animal {
}

class Lion extends Animal implements Jumpable {
}
```

A Jumpable var1 = new Jumpable();

B **Animal var2 = new Animal();**

C Lion var3 = new Animal();

D Jumpable var4 = new Animal();

E **Jumpable var5 = new Lion();**

F Jumpable var6 = (Jumpable)(new Animal());

Answer: B, E

Explanation:

Option (a) is incorrect. An interface can't be instantiated.

Option (c) is incorrect. A reference variable of a derived class can't be used to refer to an object of its base class.

Option (d) is incorrect. A reference variable of type Jumpable can't be used to refer to an object of the class Animal because Animal doesn't implement the interface Jumpable.

Option (f) is incorrect. Although this line of code will compile successfully, it will throw a ClassCastException at runtime.

You can explicitly cast ***any*** object to an interface, even if it doesn't implement it to make the code compile. But if the object's class doesn't implement the interface, the code will throw a ClassCastException at runtime.

ME-Q2) Given the following code, which option, if used to replace /* INSERT CODE HERE */, will make the code print 1? (Select 1 option.)

```
try {
    String[][] names = {{"Andre", "Mike"}, null, {"Pedro"}};
    System.out.println(names[2][1].substring(0, 2));
} catch (/*INSERT CODE HERE*/) {
    System.out.println(1);
}
```

A IndexPositionException e

B NullPointerException e

C ArrayIndexOutOfBoundsException e

D ArrayOutOfBoundsException e

Answer: C

Explanation: Options (a) and (d) are incorrect because the Java API doesn't define any exception classes with these names.

Here's a list of the array values that are initialized by the code in this question:

names[0][0] = "Andre"

names[0][1] = "Mike"

names[1] = null

names[2][0] = "Pedro"

Because the array position [2][1] isn't defined, any attempt to access it will throw an ArrayIndexOutOfBoundsException.

An attempt to access any position of the second array—that is, names[1][0]—will throw a NullPointerException because names[1] is set to null.

ME-Q3) What is the output of the following code? (Select 1 option.)

```
public static void main(String[] args) {
    int a = 10;
    String name = null;
    try {
        a = name.length(); //line1
        a++; //line2
    } catch (NullPointerException e) {
        ++a;
        return;
    } catch (RuntimeException e) {
        a--;
        return;
    } finally {
        System.out.println(a);
    }
}
```

A 5

B 6

C 10

D 11

E 12

F Compilation error

G No output

H Runtime exception

Answer: D

Explanation: Because the variable name isn't assigned a value, you can't call an instance method (length()) using it. The following line of code will throw a Null-PointerException:

name.length();

When an exception is thrown, the control is transferred to the exception handler, skipping the execution of the remaining lines of code in the try block.

So the code (a++) doesn't execute at the comment marked with line2.

The code defines an exception handler for both NullPointerException and RuntimeException. When an exception is thrown, more than one exception handler won't execute. In this case, the exception handler for NullPointerException will execute because it's more specific and it's defined earlier than RuntimeException. The exception handler for NullPointerException includes the following code:

++a;

return;

The preceding code increments the value of the variable a by 1; and before it exits the method main, due to the call to the statement return, it executes the finally block, outputting the value 11. A finally block executes even if the catch block includes a return statement.

ME-Q4) Given the following class definition,

```
class Student {
    int marks = 10;
}
```

what is the output of the following code? (Select 1 option.)

```
class Result {
    public static void main(String... args) {
        Student s = new Student();
        switch (s.marks) {
            default:
                System.out.println("100");
            case 10:
                System.out.println("10");
            case 98:
                System.out.println("98");
        }
    }
```

A 100

10

98

B 10

98

C 100

D 10

Answer: B

Explanation: The default case executes only if no matching values are found. In this case, a matching value of 10 is found and the case label prints 10. Because a break statement doesn't terminate this case label, the code execution continues and executes the remaining statements within the switch block, until a break statement terminates it or it ends.

ME-Q5) Given the following code, which code can be used to create and initialize an object of the class ColorPencil? (Select 2 options.)

```
class Pencil {}

class ColorPencil extends Pencil {
    String color;
    ColorPencil(String color) {
        this.color = color;
    }
}
```

A ColorPencil var1 = new ColorPencil();

B ColorPencil var2 = new ColorPencil(RED);

C ColorPencil var3 = new ColorPencil("RED");

D Pencil var4 = new ColorPencil("BLUE");

Answer: C, D

Explanation:

Option (a) is incorrect because new ColorPencil() tries to invoke the no-argument constructor of the class ColorPencil, which isn't defined in the class ColorPencil.

Option (b) is incorrect because new ColorPencil(RED) tries to pass a variable RED, which isn't defined in the code.

ME-Q6) What is the output of the following code? (Select 1 option.)

```
class Doctor {
    protected int age;

    protected void setAge(int val) {
        age = val;
    }

    protected int getAge() {
        return age;
    }
}

class Surgeon extends Doctor {
    Surgeon(String val) {
        specialization = val;
    }

    String specialization;

    String getSpecialization() {
        return specialization;
    }
}

class Hospital {
    public static void main(String args[]) {
        Surgeon s1 = new Surgeon("Liver");
        Surgeon s2 = new Surgeon("Heart");
        s1.age = 45;
        System.out.println(s1.age + s2.getSpecialization());
        System.out.println(s2.age + s1.getSpecialization());
    }
}
```

A **45Heart**

0Liver

B 45Liver

0Heart

C 45Liver

45Heart

D 45Heart

45Heart

E Class fails to compile.

Answer: A

Explanation: The constructor of the class Surgeon assigns the values "Liver" and "Heart" to the variable specialization of objects s1 and s2. The variable age is protected in the class Doctor. Also, the class Surgeon extends the class Doctor. Hence, the variable age is accessible to reference variables s1 and s2. The code assigns a value of 45 to the member variable age of reference variable s1. The variable age of reference variable s2 is initialized to the default value of an int, which is 0. Hence, the code prints the values mentioned in option (a).

ME-Q7) What is the output of the following code? (Select 1 option.)

```
class RocketScience {
    public static void main(String args[]) {
        int a = 0;
        while (a == a++) {
            a++;
            System.out.println(a);
        }
    }
}
```

A The while loop won't execute; nothing will be printed.

B The while loop will execute indefinitely, printing all numbers, starting from 1.

C The while loop will execute indefinitely, printing all even numbers, starting from 0.

D **The while loop will execute indefinitely, printing all even numbers, starting from 2.**

E The while loop will execute indefinitely, printing all odd numbers, starting from 1.

F The while loop will execute indefinitely, printing all odd numbers, starting from 3.

Answer: D

Explanation: The while loop will execute indefinitely because the condition a == a++ will always evaluate to true. The postfix unary operator will increment the value of the variable a after it's used in the comparison expression. a++ within the loop body will increment the value of a by 1. Hence, the value of a increments by 2 in a single loop.

ME-Q8) Given the following statements,

■ com.ejava is a package

■ class Person is defined in package com.ejava

■ class Course is defined in package com.ejava

which of the following options correctly import the classes Person and Course in the class MyEJava? (Select 3 options.)

A **import com.ejava.*;**
class MyEJava {}

B import com.ejava;
class MyEJava {}

C **import com.ejava.Person;**
import com.ejava.Course;
class MyEJava {}

D **import com.ejava.Person;**
import com.ejava.*;
class MyEJava {}

Answer: A, C, D

Explanation:

Option (a) is correct. The statement import com.ejava.*; imports all the public members of the package com.ejava in the class MyEJava.

Option (b) is incorrect. Because com.ejava is a package, to import all the classes defined in this package, the package name should be followed by .*:
import com.ejava.*;
Option (c) is correct. It uses two separate import statements to import each of the classes Person and Course individually, which is correct.
Option (d) is also correct. The first import statement imports only the class Person in MyClass. But the second import statement imports both the Person and Course classes from the package com.ejava. You can import the same class more than once in a Java class with no issues. This code is correct.
In Java, the import statement makes the imported class *visible* to the Java compiler, allowing it to be referred to by the class that's importing it. In Java, the import statement doesn't embed the imported class in the target class.

ME-Q9) Given that the following classes Animal and Forest are defined in the same package, examine the code and select the correct statements (select 2 options).

```
line1> class Animal {
line2> public void printKing() {
line3> System.out.println("Lion");
line4> }
line5> }
line6> class Forest {
line7> public static void main(String... args) {
line8> Animal anAnimal = new Animal();
line9> anAnimal.printKing();
line10> }
line11> }
```

A The class Forest prints Lion.

B If the code on line 2 is changed as follows, the class Forest will print Lion:
private void printKing() {

C If the code on line 2 is changed as follows, the class Forest will print Lion:
void printKing() {

D If the code on line 2 is changed as follows, the class Forest will print Lion:
default void printKing() {

Answer: A, C

Explanation:
Option (a) is correct. The code will compile successfully and print Lion.

Option (b) is incorrect. The code won't compile if the access modifier of the method printKing is changed to private. private members of a class can't be accessed outside the class.

Option (c) is correct. The classes Animal and Forest are defined in the same package, so changing the access modifier of the method printKing to default access will still make it accessible in the class Forest. The class will compile successfully and print Lion.

Option (d) is incorrect. "default" isn't a valid access modifier or keyword in Java. In Java, the default accessibility is marked by the absence of any explicit access modifier.

This code will fail to compile.

ME-Q10) Given the following code,

```
class MainMethod {
    public static void main(String... args) {
        System.out.println(args[0] + ":" + args[2]);
    }
}
```

what is its output if it's executed using the following command? (Select 1 option.)

```
java MainMethod 1+2 2*3 4-3 5+1
```

A java:1+2

B java:3

C MainMethod:2*3

D MainMethod:6

E 1+2:2*3

F 3:3

G 6

H 1+2:4-3

I 3 1

J 4

Answer: H

Explanation: This question tests you on multiple points.

1 *The arguments that are passed on to the main method*—The keyword java and the name of the class (MainMethod) aren't passed as arguments to the main method. The arguments following the class name are passed to the main method. In this case, four method arguments are passed to the main method, as follows:

args[0]: 1+2

args[1]: 2*3

args[2]: 4-3

args[3]: 5+1

2 *The type of the arguments that are passed to the main method*—The main method accepts arguments of type String. All the numeric expressions—1+2, 2*3, 5+1, and 4-3—are passed as literal String values. These won't be evaluated when you try to print their values. Hence, args[0] won't be printed as 3. It will be printed as 1+2.

3 *+ operations with String array elements*—Because the array passed to the main method contains all the String values, using the + operand with its individual values will concatenate its values. It won't add the values, if they are numeric expressions. Hence, "1+2"+"4-3" won't evaluate to 31 or 4.

ME-Q11) What is the output of the following code? (Select 1 option.)

```
interface Moveable {
    int move(int distance);
}

class Person {
    static int MIN_DISTANCE = 5;
    int age;
    float height;
    boolean result;
    String name;
}

public class EJava {
    public static void main(String arguments[]) {
        Person person = new Person();
        Moveable moveable = (x) -> Person.MIN_DISTANCE + x;
        System.out.println(person.name + person.height + person.result
                + person.age + moveable.move(20));
    }
}
```

A   null0.0false025

B   null0false025

C   null0.0ffalse025

D   0.0false025

E 0false025

F 0.0ffalse025

G null0.0true025

H 0true025

I 0.0ftrue025

J Compilation error

K Runtime exception

Answer: A

Explanation: The instance variables of a class are all assigned default values if no explicit value is assigned to them.

Here are the default values of the primitive data types and the objects:

- char -> \u0000
- byte, short, int -> 0
- long -> 0L
- float-> 0.0f
- double -> 0.0d
- boolean -> false
- objects -> null

Moveable is a functional interface. The example code defines the code to execute for its functional method move by using the Lambda expression
(x) -> Person.MIN_DISTANCE + x.
Calling moveable.move(20) passes 20 as an argument to the method move. It returns 25 (the sum of Person.MIN_DISTANCE, which is 5, and the method argument 20).

ME-Q12) Given the following code, which option, if used to replace /* INSERT CODE HERE */, will make the code print the value of the variable **pagesPerMin**? (Select 1 option.)

```
class Printer {
    int inkLevel;
}

class LaserPrinter extends Printer {
    int pagesPerMin;

    public static void main(String args[]) {
        Printer myPrinter = new LaserPrinter();
        System.out.println(/* INSERT CODE HERE */);
    }
}
```

A (LaserPrinter)myPrinter.pagesPerMin

B myPrinter.pagesPerMin

C LaserPrinter.myPrinter.pagesPerMin

D ((LaserPrinter)myPrinter).pagesPerMin

Answer: D

Explanation:

Option (a) is incorrect because (LaserPrinter) tries to cast myPrinter .pagesPerMin (variable of primitive type int) to LaserPrinter, which is incorrect. This code won't compile.

Option (b) is incorrect. The type of reference variable myPrinter is Printer.

myPrinter refers to an object of the class LaserPrinter, which extends the class Printer. A reference variable of the base class can't access the variables and methods defined in its subclass without an explicit cast.

Option (c) is incorrect. LaserPrinter.myPrinter treats LaserPrinter as a variable, although no variable with this name exists in the question's code. This code fails to compile.

ME-Q13) What is the output of the following code? (Select 1 option.)

```
interface Keys {
    String keypad(String region, int keys);
}

public class Handset {
    public static void main(String... args) {
        double price;
        String model;
        Keys varKeys = (region, keys) ->
        {
            if (keys >= 32)
                return region;
            else
                return "default";
        };
        System.out.println(model + price + varKeys.keypad("AB", 32));
    }
}
```

A null0AB

B null0.0AB

C null0default

D null0.0default

E 0

F 0.0

G Compilation error

Answer: G

Explanation: The local variables (variables that are declared within a method) aren't initialized with their default values. If you try to print the value of a local variable before initializing it, the code won't compile.
The Lambda expression used in the code is correct.

ME-Q14) What is the output of the following code? (Select 1 option.)

```
public class Sales {
    public static void main(String args[]) {
        int salesPhone = 1;
        System.out.println(salesPhone++ + ++salesPhone + ++salesPhone);
    }
}
```

A 5

B 6

C 8

D 9

Answer: C

Explanation: Understanding the following rules will enable you to answer this question correctly:

■ An arithmetic expression is evaluated from left to right.

■ When an expression uses the unary increment operator (++) in postfix notation, its value increments just *after* its original value is used in an expression.

■ When an expression uses the unary increment operator (++) in prefix notation, its value increments just *before* its value is used in an expression.

The initial value of the variable salesPhone is 1. Let's evaluate the result of the arithmetic expression salesPhone++ + ++salesPhone + ++salesPhone step by step:
1 The first occurrence of salesPhone uses ++ in postfix notation, so its value is used in the expression *before* it's incremented by 1. This means that the expression evaluates to
1 + ++salesPhone + ++salesPhone 2 Note that the previous usage of ++ in postfix increments has already incremented the value of salesPhone to 2. The second occurrence of salesPhone uses ++ in prefix notation, so its value is used in the expression *after* it's incremented by 1, to 3. This means that the expression evaluates to 1 + 3 + ++salesPhone 3 The third occurrence of salesPhone again uses ++ in prefix notation, so its value is used in the expression *after* it's incremented by 1, to 4. This means that the expression evaluates to
1 + 3 + 4 The preceding expression evaluates to 8.

ME-Q15) Which of the following options defines the correct structure of a Java class that compiles successfully? (Select 1 option.)

A
```
package com.ejava.guru;
package com.ejava.oracle;

class MyClass {
    int age = /* 25 */ 74;
}
```
B
```
import com.ejava.guru.*;
import com.ejava.oracle.*;
package com.ejava;

class MyClass {
    String name = "e" + "Ja /*va*/ v";
}
```
C
```
 class MyClass {
import com.ejava.guru .*;
}
```
D
```
 class MyClass {
int abc;
String course = //this is a comment
                "eJava";
}
```
E None of the above

Answer: D

Explanation: This question requires you to know

■ Correct syntax and usage of comments

■ Usage of import and package statements

None of the code fails to compile due to the end-of-line or multiline comments. All the following lines of code are valid:

int age = /* 25 */ 74;

String name = "e" + "Ja /*va*/ v";

String course = //this is a comment

"eJava";

In the preceding code, the variable age is assigned an integer value 74, the variable name is assigned a string value "eJa /*va*/ v", and course is assigned a string value "eJava". A multiline comment delimiter is ignored if put inside a string definition.

Let's see how all the options perform on the usage of package and import statements:

Option (a) is incorrect. A class can't define more than one package statement.

Option (b) is incorrect. Although a class can import multiple packages in a class, the package statement must be placed before the import statement.

Option (c) is incorrect. A class can't define an import statement within its class body. The import statement appears before the class body.

Option (d) is correct. In the absence of any package information, this class becomes part of the default package.

ME-Q16) What is the output of the following code? (Select 1 option.)

```
class OpPre {
    public static void main(String... args) {
        int x = 10;
        int y = 20;
        int z = 30;
        if (x + y % z > (x + (-y) * (-z))) {
            System.out.println(x + y + z);
        }
    }
}
```

A 60

B 59

C 61

D No output.

E The code fails to compile.

Answer: D

Explanation: x+y%z evaluates to 30; (x+(y%z))and (x+(-y)*(-z)) evaluate to 610. The if condition returns false and the line of code that prints the sum of x, y, and z doesn't execute. Hence, the code doesn't provide any output.

ME-Q17) Select the most appropriate definition of the variable name and the line number on which it should be declared so that the following code compiles successfully (choose 1 option).

```
class EJava {
// LINE 1
    public EJava() {
        System.out.println(name);
    }
    void calc() {
// LINE 2
        if (8 > 2) {
            System.out.println(name);
        }
    }
    public static void main(String... args) {
// LINE 3
        System.out.println(name);
    }
}
```

A   Define static String name; on line 1.

B   Define String name; on line 1.

C   Define String name; on line 2.

D   Define String name; on line 3.

Answer: A

Explanation: The variable name must be accessible in the instance method calc, the class constructor, and the static method main. A non-static variable can't be accessed by a static method. Hence, the only appropriate option is to define a static variable name that can be accessed by all: the constructor of the class EJava and the methods calc and main.

ME-Q18) Examine the following code and select the correct statement (choose 1 option).

```
line1>   class Emp {
line2>         Emp mgr = new Emp();
line3>       }
line4>   class Office {
line5>         public static void main(String args[]) {
line6>             Emp e = null;
line7>             e = new Emp();
line8>           e = null;
line9>         }
line10>    }
```

A The object referred to by object e is eligible for garbage collection on line 8.

B The object referred to by object e is eligible for garbage collection on line 9.

C The object referred to by object e isn't eligible for garbage collection because its member variable mgr isn't set to null.

D The code throws a runtime exception and the code execution never reaches line 8 or line 9.

Answer: D

Explanation: The code throws java.lang.StackOverflowError at runtime. Line 7 creates an instance of class Emp. Creation of an object of the class Emp requires the creation of an instance variable mgr and its initialization with an object of the same class.
As you can see, the Emp object creation calls itself recursively (without an exit condition), resulting in a java.lang.StackOverflowError.

ME-Q19) Given the following,

```
long result;
```

which options are correct declarations of methods that accept two String arguments and an int argument and whose return value can be assigned to the variable result? (Select 3 options.)

```
A    Short myMethod1(String str1, int str2, String str3)
B    Int myMethod2(String val1, int val2, String val3)
C    Byte myMethod3(String str1, str2, int a)
D    Float myMethod4(String val1, val2, int val3)
E    Long myMethod5(int str2, String str3, String str1)
F    Long myMethod6(String... val1, int val2)
G    Short myMethod7(int val1, String... val2)
```

Answer: A, E, G

Explanation: The placement of the type of method parameters and the name of the method parameters doesn't matter. You can accept two String variables and then an int variable or a String variable followed by int and again a String. The name of an int variable can be str2.

As long as the names are valid identifiers, any name is acceptable. The return type of the method must be assignable to a variable of type long.
Option (a) is correct. The value of a Short instance can be assigned to a variable of the primitive type long.
Option (b) is incorrect. It won't compile because Int is not defined in the Java API.
The correct wrapper class name for the int data type is Integer.
Options (c) and (d) are incorrect. Unlike the declaration of multiple variables, which can be preceded by a single occurrence of their data type, each and every method argument *must* be preceded by its type. The declaration of the following variables is valid:
int aa, bb; But the declaration of the method parameters in the following declaration isn't:
Byte myMethod3(**String str1, str2,** int a) { /*code*/ }
Option (e) is correct. The value of a Long instance can be assigned to a variable of primitive type long.
Option (f) won't compile. If varargs is used to define method parameters, it must be the last one.
Option (g) is correct. The method parameter val2, a variable argument, can accept two String arguments. Also, the return value of method7(), a Short, can be assigned to a variable of type long.

ME-Q20) Which of the following will compile successfully? (Select 3 options.)

```
A   int eArr1[] = {10, 23, 10, 2};
B   int[] eArr2 = new int[10];
C   int[] eArr3 = new int[] {};
D   int[] eArr4 = new int[10] {};
E   int eArr5[] = new int[2] {10, 20};
```

Answer: A, B, C

Explanation: Option (d) is incorrect because it defines the size of the array while using {}, which isn't allowed. Both of the following lines of code are correct:
int[] eArr4 = new int[10];
int[] eArr4 = new int[]{};

Option (e) is incorrect because it's invalid to specify the size of the array within the square brackets when you're declaring, instantiating, and initializing an array in a single line of code.

ME-Q21) Assume that Oracle has asked you to create a method that returns the concatenated value of two String objects. Which of the following methods can accomplish this job? (Select 2 options.)

A
```java
public String add(String 1, String 2) {
    return str1 + str2;
}
```
B
```java
private String add(String s1, String s2) {
    return s1.concat(s2);
}
```
C
```java
 protected String add(String value1, String value2) {
        return value2.append(value2);
}
```
D
```java
String subtract(String first, String second) {
    return first.concat(second.substring(0));
}
```

Answer: B, D

Explanation: Option (a) is incorrect. This method defines method parameters with invalid identifier names. Identifiers can't start with a digit.

Option (b) is correct. The method requirements don't talk about the access modifier of the required method. It can have any accessibility.

Option (c) is incorrect because the class String doesn't define any append method.

Option (d) is correct. Even though the name of the method—subtract—isn't an appropriate name for a method that tries to concatenate two values, it does accomplish the required job.

ME-Q22) Given the following,

```java
        int ctr=10;
        char[]arrC1=new char[]{'P','a','u','l'};
        char[]arrC2={'H','a','r','r','y'};
//INSERT CODE HERE
        System.out.println(ctr);
```

which options, when inserted at //INSERT CODE HERE, will output 14? (Choose 2 options.)

A
```
    for (char c1 : arrC1) {
            for (char c2 : arrC2) {
            if (c2 == 'a')
             break;
            ++ctr;
        }
    }
```
B
```
    for (char c1 : arrC1)
    for (char c2 : arrC2) {
     if (c2 == 'a')
         break;
     ++ctr;
    }
```
C
```
    for (char c1 : arrC1)
        for (char c2 : arrC2)
            if (c2 == 'a')
                break;
++ctr;
```
D
```
for (char c1 : arrC1) {
    for (char c2 : arrC2) {
        if (c2 == 'a')
            continue;
        ++ctr;
    }
}
```

Answer: A, B

Explanation: Options (a) and (b) differ only in the usage of {} for the outer for construct. You can use {} to group the statements to execute for iteration constructs like do, do-while, and for. {} are also used with conditional constructs like switch and if-else.

The initial value of the variable ctr is 10. The size of array arrC1 is 4 and the size of array arrC2 is 5 with 'a' at the second position. The outer loop executes four times.

Because the second character referred to by arrC2 is 'a', the inner loop will increment the value of variable ctr for its first element. The inner loop won't execute ++ctr for its second element because c2=='a' returns true and the break statement, and exits the inner loop. The inner loop increments the value ctr four times, incrementing its value to 14.

Option (c) is incorrect. Because the inner for loop doesn't use {} to group the lines of code that must execute for it, the code ++ctr isn't a part of the inner for loop.
Option (d) is incorrect. The code ++ctr just executes once, after the completion of the outer and inner for loops, because it isn't a part of the looping constructs.

ME-Q23) Given the following definitions of the class ChemistryBook, select the statements that are correct individually (choose 2 options).

```
import java.util.ArrayList;

class ChemistryBook {
    public void read() {
    } //METHOD1

    public String read() {
        return null;
    } //METHOD2

    ArrayList read(int a) {
        return null;
    } //METHOD3
}
```

A Methods marked with //METHOD1 and //METHOD2 are correctly overloaded methods.

B Methods marked with //METHOD2 and //METHOD3 are correctly overloaded methods.

C Methods marked with //METHOD1 and //METHOD3 are correctly overloaded methods.

D All the methods—methods marked with //METHOD1, //METHOD2, and //METHOD3— are correctly overloaded methods.

Answer: B, C

Explanation: Options (a) and (d) are incorrect because the methods read marked with //METHOD1 and //METHOD2 differ only in their return types, void and String. Overloaded methods can't be defined with only a change in their return types; hence, these methods don't qualify as correctly overloaded methods.
Note that the presence of methods marked with //METHOD1 and //METHOD2 together will cause a compilation error.

ME-Q24) Given the following,

```
final class Home {
    String name;
    int rooms;
//INSERT CONSTRUCTOR HERE
}
```

which options, when inserted at //INSERT CONSTRUCTOR HERE, will define valid overloaded constructors for the class Home? (Choose 3 options.)

A Home() {}

B Float Home() {}

C protected Home(int rooms) {}

D final Home() {}

E private Home(long name) {}

F float Home(int rooms, String name) {}

G static Home() {}

Answer: A, C, E

Explanation: A constructor must not define an explicit return type. It you use it to do so, it's no longer a constructor. A constructor can be defined using any access level—private, default, protected, and public—irrespective of the access level that's used to declare the class.

Options (b) and (f) are incorrect because they define explicit return types: Float or float. The code in these options defines a method with the name Home,is not a constructors.

Options (d) and (g) are incorrect. The code won't compile because a constructor can't be defined using non-access modifiers static, abstract, or final.

ME-Q25) Given the following code, which option, if used to replace //INSERT CODE HERE, will make the code print numbers that are completely divisible by 14? (Select 1 option.)

```
for(int ctr=2;ctr<=30;++ctr){
        if(ctr%7!=0)
//INSERT CODE HERE
        if(ctr%14==0)
        System.out.println(ctr);
}
```

A continue;

B exit;

C break;

D end;

Answer: A

Explanation: Options (b) and (d) are incorrect because exit and end aren't valid statements in Java.

Option (c) is incorrect. Using break will terminate the for loop for the first iteration of the for loop so that no output is printed.

ME-Q26) What is the output of the following code? (Select 1 option.)

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

public class MyCalendar {
    public static void main(String arguments[]) {
        Season season1 = new Season();
        season1.name = "Spring";
        Season season2 = new Season();
        season2.name = "Autumn";
        Predicate<String> aSeason = (s) -> s == "Summer" ?
season1.name : season2.name;
        season1 = season2;
        System.out.println(season1.name);
        System.out.println(season2.name);
        System.out.println(aSeason.test(new String("Summer")));
    }
}

class Season {
    String name;
}
```

A String
Autumn
false

B Spring
String
false

C Autumn
Autumn
false

D Autumn

String

true

E Compilation error

F Runtime exception

Answer: E

Explanation: The return type of the functional method test in the functional interface Predicate is boolean. The following Lambda expression is trying to return a String value and so the code fails compilation:

```
Predicate<String> aSeason = (s) -> s == "Summer" ? season1.name : season2.name;
```

This question also covers another important topic: multiple variable references can refer to the same instances. Let's assume that you modify the preceding Lambda expression as follows:

```
Predicate<String> aSeason = (s) -> s == "Summer";
```

In this case, the code will output the following:

```
Autumn
Autumn
false
```

This is because multiple variable references can point to the same object. The following lines of code define a reference variable season1, which refers to an object that has the value of its instance variable (name) set to Spring:

```
Season season1 = new Season();
season1.name = "Spring";
```

The following lines of code define a reference variable season2, which refers to an object that has the value of its instance variable (name) set to Autumn:

```
Season season2 = new Season();
season2.name = "Autumn";
```

The following line of code reinitializes the reference variable season1 and assigns it to the object referred to by the variable season2:

```
season1 = season2;
```

Now the variable season1 refers to the object that's also referred to by the variable season2. Both of these variables refer to the same object—the one that has the value of the instance variable set to Autumn. Hence, the output of the modified code is as follows:

```
Autumn
Autumn
false
```

A quick reminder for the reason for the preceding output: a String object with the value "Summer", created with the new operator, is never pooled by the JVM. Here, such an instance is compared to the pooled "Summer" instance through the == operator.

ME-Q27) What is true about the following code? (Select 1 option.)

```java
class Shoe {
}
class Boot extends Shoe {
}
class ShoeFactory {
    ShoeFactory(Boot val) {
        System.out.println("boot");
    }
    ShoeFactory(Shoe val) {
        System.out.println("shoe");
    }
}
```

A The class ShoeFactory has a total of two overloaded constructors.

B The class ShoeFactory has three overloaded constructors, two user-defined constructors, and one default constructor.

C The class ShoeFactory will fail to compile.

D The addition of the following constructor will increment the number of constructors of the class ShoeFactory to 3:

private ShoeFactory (Shoe arg) {}

Answer: A

Explanation: Java accepts changes in the objects of base-derived classes as a sole criterion to define overloaded constructors and methods.

Option (b) is incorrect because Java doesn't generate a default constructor for a class that has already defined a constructor.

Option (c) is incorrect. All classes defined for this example compile successfully.
Option (d) is incorrect. The class ShoeFactory already defines a constructor that accepts a method argument of type Shoe. You can't overload a constructor with a mere change in its access modifier.

ME-Q28) Given the following definitions of the classes ColorPencil and TestColor which option, if used to replace //INSERT CODE HERE, will initialize the instance variable color of the reference variable myPencil with the String literal value "RED"? (Select 1 option.)

```
class ColorPencil {
    String color;
    ColorPencil(String color) {
//INSERT CODE HERE
    }
}
class TestColor {
    ColorPencil myPencil = new ColorPencil("RED");
}
```

A **this.color = color;**

B color = color;

C color = RED;

D this.color = RED;

Answer: A

Explanation: Option (b) is incorrect. This line of code will assign the value of the method parameter to itself. The constructor of the class ColorPencil defines a method parameter with the same name as its instance variable, color. To access an instance variable in the constructor, it must be prefixed with the keyword this, or it will refer to the method parameter color.
Options (c) and (d) are incorrect. They try to access the value of the variable RED, which isn't defined in the code.

ME-Q29) What is the output of the following code? (Select 1 option.)

```
class EJavaCourse {
    String courseName = "Java";
}

class University {
    public static void main(String args[]) {
     EJavaCourse courses[] = {new EJavaCourse(), new EJavaCourse()};
```

```
                courses[0].courseName = "OCA";
                for (EJavaCourse c : courses)
                        c = new EJavaCourse();
                for (EJavaCourse c : courses)
        System.out.println(c.courseName);
            }
        }
```

A Java

Java

B OCA

Java

C OCA

OCA

D None of the above

Answer: B

Explanation: This question tests you on multiple concepts: how to read from and write to object fields, how to use arrays, the enhanced for loop, and assigning a value to a loop variable.

The code defines an array of the class EJavaCourse with two elements. The default value of the variable courseName—Java—is assigned to each of these two elements.

courses[0].courseName = "OCA" changes the value courseName for the object stored at array position 0. c = new EJavaCourse() assigns a new object to the loop variable c.

This assignment doesn't reassign new objects to the array reference variables.

System.out.println(c.courseName) prints the name of the courseName of the objects initially stored by the array, using the loop variable c.

The loop variable in the enhanced for loop refers to a copy of the array or list element.

If you modify the state of the loop variable, the modified object state will be reflected in the array. But if you assign a new object to the loop variable, it won't be reflected in the list or the array that's being iterated. You can compare this behavior of the enhanced for loop variable with the behavior of object references passed as arguments to a method.

ME-Q30) What is the output of the following code? (Select 1 option.)

```
        class Phone {
            static void call() {
                System.out.println("Call-Phone");
            }
```

```
}
class SmartPhone extends Phone {
    static void call() {
        System.out.println("Call-SmartPhone");
    }
}
class TestPhones {
    public static void main(String... args) {
        Phone phone = new Phone();
        Phone smartPhone = new SmartPhone();
        phone.call();
        smartPhone.call();
    }
}
```

A Call-Phone

Call-Phone

B Call-Phone

Call-SmartPhone

C Call-Phone

null

D null

Call-SmartPhone

Answer: A

Explanation: Invocation of a static method is tied to the type of the reference variable and doesn't depend on the type of the object that's assigned to the reference variable. The static method belongs to a class, not to its objects. Reexamine the following code:

Phone smartPhone = new SmartPhone();

smartPhone.call();

In the preceding code, the type of the reference variable smartPhone is Phone. Because call is a static method, smartPhone.call() calls the method call defined in the class Phone.

ME-Q31) Given the following code, which of the following statements are true? (Select 3 options.)

```
class MyExam {
    void question() {
        try {
            question();
        } catch (StackOverflowError e) {
            System.out.println("caught");
        }
    }

    public static void main(String args[]) {
        new MyExam().question();
    }
}
```

A **The code will print caught.**

B The code won't print caught.

C **The code would print caught if StackOverflowError were a runtime exception.**

D **The code would print caught if StackOverflowError were a checked exception.**

E The code would print caught if question() throws the exception NullPointer-Exception.

Answer: A, C, D

Explanation: Option (a) is correct. The control will be transferred to the exception handler for StackOverflowError when it's encountered. Hence it will print caught.

Options (c) and (d) are correct. Exception handlers execute when the corresponding checked or runtime exceptions are thrown.

Option (e) is incorrect. An exception handler for the class StackOverflow can't handle exceptions of the class NullPointerException because NullPointerException is not a superclass of StackOverflowError.

ME-Q32) A class Student is defined as follows:

```
public class Student {
    private String fName;
    private String lName;

    public Student(String first, String last) {
        fName = first;
        lName = last;
    }

    public String getName() {
        return fName + lName;
    }
}

The creator of the class later changes the method getName as
follows:

public String getName(){
        return fName+" "+lName;
}
```

What are the implications of this change? (Select 2 options.)

A The classes that were using the class Student will fail to compile.

B The classes that were using the class Student will work without any compilation issues.

C The class Student is an example of a well-encapsulated class.

D The class Student exposes its instance variable outside the class.

Answer: B, C

Explanation: This is an example of a well-encapsulated class. There's no change in the signature of the method getName after it's modified. Hence, none of the code that uses this class and method will face any compilation issues.Its instance variables (fName and lName) aren't exposed outside the class. They're available only via a public method: getName.

ME-Q33) What is the output of the following code? (Select 1 option.)

```
class ColorPack {
    int shadeCount = 12;

    static int getShadeCount() {
        return shadeCount;
    }
}

class Artist {
    public static void main(String args[]) {
        ColorPack pack1 = new ColorPack();
        System.out.println(pack1.getShadeCount());
    }
}
```

A 10

B 12

C No output

D Compilation error

Answer: D

Explanation: A static method can't access non-static instance variables of a class. Hence, the class ColorPack fails to compile.

ME-Q34) Paul defined his Laptop and Workshop classes to upgrade his laptop's memory. Do you think he succeeded? What is the output of this code? (Select 1 option.)

```
class Laptop {
    String memory = "1 GB";
}

class Workshop {
    public static void main(String args[]) {
        Laptop life = new Laptop();
        repair(life);

        System.out.println(life.memory);
    }

    public static void repair(Laptop laptop) {
        laptop.memory = "2 GB";
    }
}
```

A 1 GB

B 2 GB

C Compilation error

D Runtime exception

Answer: B

Explanation: The method repair defined in this example modifies the state of the method parameter laptop that's passed to it. It does so by modifying the value of the instance variable memory.

When a method modifies the state of an object reference variable that's passed to it, the changes made are visible in the calling method. The method repair makes changes to the state of the method parameter laptop; these changes are visible in the method main. Hence, the method main prints the value of life.memory as 2 GB.

ME-Q35) What is the output of the following code? (Select 1 option.)

```
public class Application {
    public static void main(String... args) {
        double price = 10;
        String model;
        if (price > 10)
            model = "Smartphone";
        else if (price <= 10)
            model = "landline";
        System.out.println(model);
    }
}
```

A landline

B Smartphone

C No output

D Compilation error

Answer: D

Explanation: The local variables aren't initialized with default values. Code that tries to print the value of an uninitialized local variable fails to compile.

In this code, the local variable model is only declared, not initialized. The initialization of the variable model is placed within the if and else-if constructs. If you initialize a variable within an if or else-if construct, the compiler can't be sure whether these conditions will evaluate to true, resulting in no initialization of the local variable. Because there's no else at the bottom and the compiler can't tell whether the if and else-if are mutually exclusive, the code won't compile.

If you remove the condition if (price <= 10) from the previous code, the code will compile successfully:

```
public class Application {
    public static void main(String... args) {
        double price = 10;
        String model;
        if (price > 10)
            model = "Smartphone";
        else
            model = "landline";
        System.out.println(model);
    }
}
```

In this code, the compiler can be sure about the initialization of the local variable model.

ME-Q36) What is the output of the following code? (Select 1 option.)

```
class EString {
    public static void main(String args[]) {
        String eVal = "123456789";

System.out.println(eVal.substring(eVal.indexOf("2"),eVal.indexOf("0"
)).concat("0"));
    }
}
```

A 234567890

B 34567890

C 234456789

D 3456789

E Compilation error

F Runtime exception

Answer: F

Explanation: When multiple methods are chained on a single code statement, the methods execute from left to right, not from right to left. eVal.indexOf("0") returns a negative value because, as you can see, the String eVal doesn't contain the digit 0. Hence, eVal.substring is passed a negative end value, which results in a runtime exception.

ME-Q37) Examine the following code and select the correct statements (choose 2 options).

```
class Artist {
    Artist assistant;
}

class Studio {
    public static void main(String... args) {
        Artist a1 = new Artist();
        Artist a2 = new Artist();
        a2.assistant = a1;
        a2 = null; // Line 1
    }
// Line 2
}
```

A At least two objects are garbage collected on line 1.

B At least one object is garbage collected on line 1.

C No objects are garbage collected on line 1.

D The number of objects that are garbage collected on line 1 is unknown.

E At least two objects are eligible for garbage collection on line 2.

Answer: D, E

Explanation: Options (a), (b), and (c) are incorrect.

When an object reference is marked as null, the object is marked for garbage collection. But you can't be sure exactly when a garbage collector will kick in to garbage collect the objects. A garbage collector is a low-priority thread, and its exact execution time will depend on the OS. The OS will start this thread if it needs to claim unused space. You can

be sure only about the number of objects that are eligible for garbage collection. You can never be sure about which objects have been garbage collected, so any statement that asserts that a particular number of objects *have* been garbage collected is incorrect.
Option (d) is correct. As mentioned previously, the exact number of objects garbage collected at any point in time can't be determined.
Option (e) is correct. If you marked this option incorrect, think again. The question wants you to select the correct statements, and this is a correct statement. You may argue that at least two objects were already made eligible for garbage collection at line 1, and you're correct. But because nothing changes on line 2, at least two objects are still eligible for garbage collection.

ME-Q38) What is the output of the following code? (Select 1 option.)

```
class Book {
    String ISBN;
    Book(String val) {
        ISBN = val;
    }
}

class TestEquals {
    public static void main(String... args) {
        Book b1 = new Book("1234-4657");
        Book b2 = new Book("1234-4657");
        System.out.print(b1.equals(b2) + ":");
        System.out.print(b1 == b2);
    }
}
```

A true:false

B true:true

C false:true

D false:false

E Compilation error—there is no equals method in the class Book.

F Runtime exception.

Answer: D

Explanation: The comparison operator determines whether the reference variables refer to the same object. Because the reference variables b1 and b2 refer to different objects, b1==b2 prints false.

The method equals is a public method defined in the class java.lang.Object.
Because the class Object is the superclass for all the classes in Java, the method equals is inherited by all classes. Hence, the code compiles successfully. The default implementation of the method equals in the base class compares the object references and returns true if both reference variables refer to the same object, and false otherwise.
Because the class Book doesn't override this method, the method equals in the base class Object is called for b1.equals(b2), which returns false. Hence, the code prints false:false

ME-Q39) Which of the following statements are correct? (Select 2 options.)

A **StringBuilder sb1 = new StringBuilder() will create a StringBuilder object with no characters but with an initial capacity to store 16 characters.**

B StringBuilder sb1 = new StringBuilder(5*10) will create a StringBuilder object with a value of 50.

C Unlike the class String, the concat method in StringBuilder modifies the value of a StringBuilder object.

D **The insert method can be used to insert a character, number, or String at the start or end or a specified position of a StringBuilder.**

Answer: A, D

Explanation: There is no concat method in the StringBuilder class. It defines a whole army of append methods (overloaded methods) to add data at the end of a StringBuilder object.
new StringBuilder(50) creates a StringBuilder object with no characters but with an initial capacity to store 50 characters.

ME-Q40) Given the following definition of the class **Animal** and the interface **Jump**, select the correct array declarations and initialization (choose 3 options).

```
interface Jump {
}

class Animal implements Jump {
}
```

A **Jump eJump1[] = {null, new Animal()};**

B Jump[] eJump2 = new Animal()[22];

C Jump[] eJump3 = new Jump[10];

D **Jump[] eJump4 = new Animal[87];**

E Jump[] eJump5 = new Jump()[12];

Answer: A, C, D

Explanation: Option (b) is incorrect because the right side of the expression is trying to create a single object of the class Animal by using parentheses (). At the same time, it's also using the square brackets [] to define an array. This combination is invalid. Option (e) is incorrect. Apart from using an invalid syntax to initialize an array (as mentioned previously), it also tries to create objects of the interface Jump. Objects of interfaces can't be created.

ME-Q41) What is the output of the following code? (Select 1 option.)

```
class EJGArrayL {
    public static void main(String args[]) {
        ArrayList<String> seasons = new ArrayList<>();
        seasons.add(1, "Spring");
        seasons.add(2, "Summer");
        seasons.add(3, "Autumn");
        seasons.add(4, "Winter");
        seasons.remove(2);
        for (String s : seasons)
            System.out.print(s + ", ");
    }
}
```

A Spring, Summer, Winter,

B Spring, Autumn, Winter,

C Autumn, Winter,

D Compilation error

E Runtime exception

Answer: E

Explanation: The code throws a runtime exception, IndexOutOfBoundsException, because the ArrayList is trying to insert its first element at position 0. Before the first call to the method add, the size of the ArrayList seasons is 0. Because season's first element is stored at position 0, a call to store its first element at position 1 will throw a RuntimeException. The elements of an ArrayList can't be added to a higher position if lower positions are available.

ME-Q42) What is the output of the following code? (Select 1 option.)

```
class EIf {
    public static void main(String args[]) {
        bool boolean =false;
        do {
            if ( boolean =true)
            System.out.println("true");
else
            System.out.println("false");
        }
        while (3.3 + 4.7 > 8);
    }
}
```

A The class will print true.

B The class will print false.

C The class will print true if the if condition is changed to boolean == true.

D The class will print false if the if condition is changed to boolean != true.

E The class won't compile.

F Runtime exception.

Answer: E

Explanation: This question tries to trick you on two points. First, there's no data type bool in Java. Second, the name of an identifier can't be the same as a reserved word.
The code tries to define an identifier of type bool with the name boolean.

ME-Q43) How many Fish did the Whale (defined as follows) manage to eat? Examine the following code and select the correct statements (choose 2 options).

```
class Whale {
    public static void main(String args[]) {
        boolean hungry = false;
        while (hungry = true) {
            ++Fish.count;
        }
        System.out.println(Fish.count);
    }
}
class Fish {
    static byte count;
}
```

A The code doesn't compile.

B The code doesn't print a value.

C The code prints 0.

D Changing ++Fish.count to Fish.count++ will give the same results.

Answer: B, D

Explanation: Option (a) is incorrect because the code compiles successfully.

Option (c) is incorrect. This question tries to trick you by comparing a boolean value when it's assigning a boolean value in the while construct. Because the while loop assigns the value true to the variable hungry, it will always return true, incrementing the value of the variable count and thus getting stuck in an infinite loop.

ME-Q44) Given the following code, which option, if used to replace /* REPLACE CODE HERE */, will make the code print the name of the phone with the position at which it's stored in the array phones? (Select 1 option.)

```
class Phones {
    public static void main(String args[]) {
        String phones[]= {"BlackBerry", "Android", "iPhone"};

        for (String phone : phones)
        /* REPLACE CODE HERE */
    }
}
```

A System.out.println(phones.count + ":" + phone);

B System.out.println(phones.counter + ":" + phone);

C System.out.println(phones.getPosition() + ":" + phone);

D System.out.println(phones.getCtr() + ":" + phone);

E System.out.println(phones.getCount() + ":" + phone);

F System.out.println(phones.pos + ":" + phone);

G None of the above

Answer: G

Explanation: The enhanced for loop doesn't provide you with a variable to access the position of the array that it's being used to iterate over. This facility comes with the regular for loop.

ME-Q45) Given the following code,

```
Byte b1 = (byte) 100; // 1
Integer i1 = (int) 200; // 2
Long l1 = (long) 300; // 3
Float f1 = (float) b1 + (int)l1; // 4
String s1 = 300; // 5
```

```
    if (s1 == (b1 + i1)) // 6
        s1 = (String) 500; // 7
    else // 8
        f1 = (int) 100; // 9
    System.out.println(s1 + ":" + f1); // 10
```
what is the output? Select 1 option.

A Code fails compilation at line numbers 1, 3, 4, 7.

B Code fails compilation at line numbers 6, 7.

C Code fails compilation at line numbers 7, 9.

D Code fails compilation at line numbers 4, 5, 6, 7, 9.

E No compilation error—outputs 500:300.

F No compilation error—outputs 300:100.

G Runtime exception.

Answer: D

Explanation: This question tests you on multiple concepts:

■ Autoboxing and unboxing wrapper classes

■ The difference between casting primitive values and wrapper classes, for example, from long to int and casting Long to int

■ Implicit and explicit casting of primitives and objects

■ Exception thrown due to invalid explicit casting (implicit casting doesn't throw any exception)

The code on lines 1, 2, and 3 doesn't throw a compilation error or runtime exceptions.
You can explicitly cast all numeric primitive data types to one another. For
example, a double primitive value can be implicitly cast to a byte. A char primitive
can be explicitly cast to any other numeric data type:

char c = 100;

Float f = (float)c;

The code at line 4 will throw a compilation error. The code (int)l1 isn't casting primitive long to int. It's trying to cast Long to primitive int. You can explicitly cast a wrapper object to only the type that it wraps.

The code at line 5 doesn't compile because you can't assign int values to String reference variables by using the assignment operator. You can use the String method valueOf(), passing it a numeric value (integer or decimal) to return a String instance.
The code at line 6 doesn't compile because you can't compare String instances with instances of wrapper classes.
The code at line 7 doesn't compile. You can't explicitly cast a numeric literal value to convert it to a String value.
The code at line 9 doesn't compile because int can't be converted to Float.

ME-Q46) What is the output of the following code? (Select 1 option.)

```
import java.time.LocalDate;

class Book {
    String ISBN;

    Book(String val) {
        ISBN = val;
    }

    public boolean equals(Object b) {
        if (b instanceof Book) {
            return ((Book) b).ISBN.equals(ISBN);
        } else
            return false;
    }
}

class TestEquals {
    public static void main(String args[]) {
       Book b1 = new Book("1234-4657");
       Book b2 = new Book("1234-4657");
       LocalDate release = null;
       release = b1.equals(b2)? b1 == b2 ? LocalDate.of(2050, 12,12)
: LocalDate.parse("2072-02-01"): LocalDate.parse("9999-09-09");
       System.out.print(release);
    }
}
```

A 2050-12-12

B 2072-02-01

C 9999-09-09

D Compilation error

E Runtime exception

Answer: B

Explanation: This question tests you on multiple concepts:

■ Usage of equals() and the comparison operator == to determine object equality

■ Usage of the ternary operator

■ Correct method and syntax to instantiate LocalDate

The comparison operator determines whether the reference variables refer to the same object. Because the reference variables b1 and b2 refer to different objects, b1==b2 will return false.

The method equals is a public method defined in the class java.lang.Object. Because the class Object is the superclass for all the classes in Java, equals is inherited by all classes. The default implementation of equals in the base class compares the object references and returns true if both reference variables refer to the same object and false otherwise.

If a class has overridden this method, it returns a boolean value depending on the logic defined in this class. The class Book overrides the equals method and returns true if the Book object defines the same ISBN value as the Book object being compared to. Because the ISBN object value of both variables b1 and b2 is the same, b1.equals(b2) returns true.

Here's the syntax of the ternary operator:

variable = booleanValue? returnValueIfTrue : returnValueIfFalse;

All the components of a ternary operator are compulsory. Unlike an if construct, you can't leave out the else part in a ternary operator. The code in this question uses a nested ternary operator. Let's indent the code, which will make it easy to evaluate the expression:

```
release = b1.equals(b2) ?
b1 == b2?
LocalDate.of(2050,12,12):
LocalDate.parse("2072-02-01"):
LocalDate.parse("9999-09-09");
```

Because b1.equals(b2) returns true, the control evaluates the nested ternary operator. Because b1 == b2 returns false, the code returns the LocalDate instance created in the else part of the ternary operator, 2072-02-01.

ME-Q47) What is the output of the following code? (Select 1 option.)

```
int a = 10;
for (; a <= 20; ++a) {
    if (a % 3 == 0)
        a++;
    else if (a % 2 == 0)
        a = a * 2;
    System.out.println(a);
}
```

A 11

13

15

17

19

B 20

C 11

14

17

20

D 40

E Compilation error

Answer: B

Explanation: This question requires multiple skills: understanding the declaration of a for loop, the use of operators, and the use of the if-else construct.

The for loop is correctly defined in the code. The for loop in this code doesn't use its variable initialization block; it starts with ; to mark the absence of its variable initialization block. The code for the if construct is deliberately incorrect, because you may encounter similar code in the exam.

For the first iteration of the for loop, the value of the variable a is 10. Because a <= 20 evaluates to true, control moves on to the execution of the if construct. This if construct can be indented properly as follows:

if (a%3 == 0)

a++;

else if (a%2 == 0)

a=a*2;

(a%3 == 0) evaluates to false and(a%2 == 0) evaluates to true, so a value of 20 (a*2) is assigned to a. The subsequent line prints the value of a as 20.

The increment part of the loop statement, (++a), increments the value of variable a to 21. For the next loop iteration, its condition evaluates to false (a <= 20), and the loop terminates.

ME-Q48) Given the following code, which option, if used to replace // INSERT CODE HERE, will define an overloaded **rideWave** method? (Select 1 option.)

```
class Raft {
    public String rideWave() {
        return null;
    }
//INSERT CODE HERE
}
```

A public String[] rideWave() { return null; }

B protected void riceWave(int a) {}

C private void rideWave(int value, String value2) {}

D default StringBuilder rideWave (StringBuffer a) { return null; }

Answer: C

Explanation: Option (a) is incorrect. Making a change only in the return value of a method doesn't define a valid overloaded method.

Option (b) is incorrect. The name of the method in this option is riceWave and not rideWave. Overloaded methods should have the same method name.

Option (d) is incorrect. default isn't a valid access modifier. The default modifier is marked by the absence of an access modifier.

ME-Q49) Given the following code, which option, if used to replace // INSERT CODE HERE, will correctly calculate the sum of all the even numbers in the array num and store it in the variable sum? (Select 1 option.)

```
int num[] = {10, 15, 2, 17};
int sum = 0;
for (int number : num) {
//INSERT CODE HERE
```

sum += number;

}

A if (number % 2 == 0)

continue;

B if (number % 2 == 0)

break;

C if (number % 2 != 0)

continue;

D if (number % 2 != 0)

break;

Answer: C

Explanation: To find the sum of the even numbers, you first need to determine whether a number is an even number. Then you need to add the even numbers to the variable sum. Option (c) determines whether the array element is completely divisible by 2. If it isn't, it skips the remaining statements in the for loop by using the continue statement, which starts execution of the for loop with the next array element. If the array element is completely divisible by 2, continue doesn't execute, and the array number is added to the variable sum.

ME-Q50) What is the output of the following code? (Select 1 option.)

```
class Op {
    public static void main(String... args) {
        int a = 0;
        int b = 100;
        Predicatea<Integer> compare = (var) -> var++ == 10;
        if (!b++ > 100 && compare.test(a)) {
            System.out.println(a + b);
        }
    }
}
```

A 100

B 101

C 102

D Code fails to compile.

E No output is produced.

Answer: D

Explanation: The code defines the Lambda expression correctly.

Although it may seem that the unary negation operator ! is being applied to the expression b++ > 100, it's actually being applied to the variable b of type int. Because a unary negation operator ! can't be applied to a variable of type int, the code fails to compile. The correct if condition would be as follows:

if (!(b++ > 100) && compare.test(a)) {

ME-Q51) Choose the option that meets the following specification: Create a well encapsulated class Pencil with one instance variable model. The value of model should be accessible and modifiable outside Pencil. (Select 1 option.)

A

```
  class Pencil {
      public String model;
   }
```

B

```
class Pencil {
     public String model;

     public String getModel() {
        return model;
     }

     public void setModel(String val) {
        model = val;
     }
  }
```

C

```
class Pencil {
    private String model;

    public String getModel() {
        return model;
    }

    public void setModel(String val) {
        model = val;
    }
}
```

D
```
class Pencil {
    public String model;

    private String getModel() {
        return model;
    }

    private void setModel(String val) {
        model = val;
    }
}
```

Answer: C

Explanation: A well-encapsulated class's instance variables shouldn't be directly accessible outside the class. They should be accessible via non-private getter and setter methods.

ME-Q52) What is the output of the following code? (Select 1 option.)

```
class Phone {
    void call() {
        System.out.println("Call-Phone");
    }
}

class SmartPhone extends Phone {
    void call() {
        System.out.println("Call-SmartPhone");
    }
}

class TestPhones {
    public static void main(String[] args) {
        Phone phone = new Phone();
        Phone smartPhone = new SmartPhone();
        phone.call();
        smartPhone.call();
    }
}
```

A Call-Phone
Call-Phone

B Call-Phone
Call-SmartPhone

C Call-Phone
null

D null

Call-SmartPhone

Answer: B

Explanation: The method call is defined in the base class Phone. This method call is inherited and overridden by the derived class SmartPhone. The type of both reference variables, phone and smartphone, is Phone. But the reference variable phone refers to an object of the class Phone, and the variable smartPhone refers to an object of the class SmartPhone. When the method call is called on the reference variable smart-Phone, it calls the method call defined in the class SmartPhone, **because a call to an overridden method is resolved at runtime and is based on the type of the object on which a method is called.**

ME-Q53) What is the output of the following code? (Select 1 option.)

```
class Phone {
    String keyboard = "in-built";
}

class Tablet extends Phone {
    boolean playMovie = false;
}

class College2 {
    public static void main(String args[]) {
        Phone phone = new Tablet();
        System.out.println(phone.keyboard + ":" + phone.playMovie);
    }
}
```

A in-built:false

B in-built:true

C null:false

D null:true

E Compilation error

Answer: E

Explanation: This code won't compile. The object reference variable phone, of type Phone, can be used to refer to an object of its derived type, Tablet. But variables of a base class can't access variables and methods of its derived classes without an explicit cast to the object of the derived class. So phone can access **keyboard** but not **playMovie**.

ME-Q54) What is the output of the following code? (Select 1 option.)

```
public class Wall {
    public static void main(String args[]) {
        double area = 10.98;
        String color;
        if (area < 5)
            color = "red";
        else
            color = "blue";
        System.out.println(color);
    }
}
```

A red

B blue

C No output

D Compilation error

Answer: B

Explanation: When you answer a question that includes accessing the value of a local variable, check whether it has been initialized or not. Code that tries to access an uninitialized local variable won't compile.

In this question, the local variable color is initialized using an if-else construct.

The variable color will be initialized as the value "red" if the value of another local variable (area) is less than 5 but will be initialized as "blue" otherwise. So, irrespective of the value of area, color will be initialized for sure. In an if-else construct, at least one of the two blocks of code, corresponding to if and else, is sure to execute.

The code will execute successfully, printing blue.

Watch out for code that initializes a local variable using a conditional statement. If the compiler can't seem to guarantee initialization of a local variable, *code that tries to access it* won't compile. Here's a modified version of the code (modifications in bold) included in this question, which doesn't compile:

```
public class Wall {
    public static void main(String args[]) {
        double area = 10.98;
        String color;
        if (area < 5)
            color = "red";
        if (area >= 5)
            color = "blue";
        System.out.println(color);     }
}
```

ME-Q55) What is the output of the following code? (Select 1 option.)

```
class Diary {
    int pageCount = 100;

    int getPageCount() {
        return pageCount;
    }

    void setPageCount(int val) {
        pageCount = val;
    }
}

class ClassRoom {
    public static void main(String args[]) {
        System.out.println(new Diary().getPageCount());
        new Diary().setPageCount(200);
        System.out.println(new Diary().getPageCount());
    }
}
```

A 100

200

B 100

100

C 200

200

D Code fails to compile.

Answer: B

Explanation: The constructor of a class creates and returns an object of the class in which it's defined. This returned object can be assigned to a reference variable. In case the returned object isn't assigned to any reference variable, none of the variables or methods of this object can be accessed again. This is what happens in the class ClassRoom. All calls to the methods getPageCount and setPageCount in the example operate on unrelated objects.

ME-Q56) How many times do you think you can shop with the following code (that is, what's the output of the following code)? (Select 1 option.)

```
class Shopping {
    public static void main(String args[]) {
        boolean bankrupt = true;
        do System.out.println("enjoying shopping");
```

```
         bankrupt = false;
          while (!bankrupt) ;     }
}
```

A The code prints enjoying shopping once.

B The code prints enjoying shopping twice.

C The code prints enjoying shopping in an infinite loop.

D The code fails to compile.

Answer: D

Explanation: The code fails to compile because it's trying to stuff two lines of code between the do and while statements without using curly braces.

ME-Q57) Which of the following options are valid for defining multidimensional arrays? (Choose 4 options.)

A String ejg1[][] = new String[1][2];

B String ejg2[][] = new String[][] { {}, {} };

C String ejg3[][] = new String[2][2];

D String ejg4[][] = new String[][]{{null},new String[]{"a","b","c"},{new String()}};

E String ejg5[][] = new String[][2];

F String ejg6[][] = new String[][]{"A", "B"};

G String ejg7[][] = new String[]{{"A"}, {"B"}};

Answer: A, B, C, D

Explanation: Options (a), (b), (c), and (d) define a multidimensional array correctly.

Option (e) is incorrect because the size in the first square bracket is missing.

Option (f) is incorrect. The correct code must use an additional pair of {} on the right-hand side, as follows:

String ejg6[][] = new String[][]{**{**"A"**}**, **{**"B"**}**};

Option (g) is incorrect. The correct code must use an additional [] on the right-hand side as follows:

String ejg7[][] = new String[]**[]**{{"A"}, {"B"}};

ME-Q58) What is the output of the following code? (Select 1 option.)

```
class Laptop {
    String memory = "1GB";
}
```

```
class Workshop {
    public static void main(String args[]) {
        Laptop life = new Laptop();
        repair(life);
        System.out.println(life.memory);
    }

    public static void repair(Laptop laptop) {
        laptop = new Laptop();
        laptop.memory = "2GB";
    }
}
```

A **1 GB**

B 2 GB

C Compilation error

D Runtime exception

Answer: A

Explanation: The method repair defined in this example assigns a new object to the method parameter laptop that's passed to it. Then it modifies the state of this new assigned object by assigning 1 GB to its instance variable, memory.

When a method reassigns an object reference variable that's passed to it, the changes made to its state aren't visible in the calling method. This is because the changes are made to a new object and not to the one that was initially passed to this method.

The method repair assigns a new object to the reference variable laptop that's passed to it and then modifies its state. Hence, the changes made to the state of the method parameter laptop aren't visible in method main, and it prints the value of **life.memory** as 1 GB.

ME-Q59) Given the following code, which option, if used to replace //INSERT CODE HERE, will enable a reference variable of type Roamable to refer to an object of the Phone class? (Select 1 option.)

```
interface Roamable {
}

class Phone {
}

class Tablet extends Phone implements Roamable {
//INSERT CODE HERE
}
```

A Roamable var = new Phone();

B Roamable var = (Roamable)Phone();

C Roamable var = (Roamable)new Phone();

D Because the interface Roamable and the class Phone are unrelated, a reference variable of type Roamable can't refer to an object of the class Phone.

Answer: C

Explanation: Option (a) is incorrect. Without explicit casting, a reference variable of type Roamable can't refer to an object of the class Phone.

Option (b) is incorrect because this is an invalid line of code that will fail to compile.

Option (d) is incorrect because a reference variable of the type Roamable can refer to an object of the class Phone with an explicit cast.

Note that although option (c) will compile, it will throw a ClassCastException if it's executed.

ME-Q60) What is the output of the following code? (Select 1 option.)

```java
class Paper {
    Paper() {
        this(10);
        System.out.println("Paper:0");
    }

    Paper(int a) {
        System.out.println("Paper:1");
    }
}
class PostIt extends Paper {
}

class TestPostIt {
    public static void main(String[] args) {
        Paper paper = new PostIt();
    }
}
```

A Paper:1

B Paper:0

C Paper:0

Paper:1

D Paper:1

Paper:0

Answer: B

Explanation: new PostIt() creates an object of the class PostIt by calling its compiler-provided no-argument constructor. The no-argument constructor of the class PostIt calls its base class no-argument constructor, which calls the other constructor that accepts one int method argument. The constructor that accepts an int argument prints Paper:1 and then returns control to the no-argument constructor. The no-argument constructor then prints Paper:0.

ME-Q61) Examine the following code and select the correct statement (choose 1 option).

```
line1> class StringBuilders {
line2> public static void main(String... args) {
line3> StringBuilder sb1 = new StringBuilder("eLion");
line4> String ejg = null;
line5> ejg = sb1.append("X").substring(sb1.indexOf("L"),sb1.indexOf("X"));
line6> System.out.println(ejg);
line7> }
line8> }
```

A The code will print LionX.

B The code will print Lion.

C The code will print Lion if line 5 is changed to the following:

ejg = sb1.append("X").substring(sb1.indexOf('L'), sb1.indexOf('X'));

D The code will compile only when line 4 is changed to the following:

StringBuilder ejg = null;

Answer: B

Explanation: Option (a) is incorrect and option (b) is correct. The substring method doesn't include the character at the end position in the result that it returns.
Hence, the code prints Lion.
Option (c) is incorrect. If line 5 is changed as suggested in this option, the code won't compile. You can't pass a char to StringBuilder's method indexOf; it accepts String.
Option (d) is incorrect because there are no compilation issues with the code.

ME-Q62) Given the following code,

```
interface Jumpable {
    int height = 1;

    default void worldRecord() {
        System.out.print(height);
    }
}

interface Moveable {
    int height = 2;

    static void worldRecord() {
        System.out.print(height);
    }
}

class Chair implements Jumpable, Moveable {
    int height = 3;

    Chair() {
        worldRecord();
    }

    public static void main(String args[]) {
        Jumpable j = new Chair();
        Moveable m = new Chair();
        Chair c = new Chair();
    }
}
```

what is the output? Select 1 option.

A 111

B 123

C 333

D 222

E Compilation error

F Runtime exception

Answer: A

Explanation: The constructor of the class Chair invokes the default non-static method defined in the interface Jumpable. Moreover, if only the static world-Record() method in the interface Moveable were defined, its invocation would have to be qualified (that is, Moveable.worldRecord();) for the class Chair to compile.

ME-Q63) Given the following code, which option, if used to replace /* INSERT CODE HERE */, will enable the class Jungle to determine whether the reference variable animal refers to an object of the class Lion and print 1? (Select 1 option.)

```
class Animal {
    float age;
}

class Lion extends Animal {
    int claws;
}
class Jungle {
    public static void main(String args[]) {
        Animal animal = new Lion();
        /* INSERT CODE HERE */
        System.out.println(1);
    }
}
```

A if (animal instanceof Lion)

B if (animal instanceOf Lion)

C if (animal == Lion)

D if (animal = Lion)

Answer: A

Explanation: Option (b) is incorrect because the correct operator name is **instanceof** and not instanceOf (note the capitalized O).

Options (c) and (d) are incorrect. Neither of these lines of code will compile because they are trying to compare and assign a class name to a variable, which isn't allowed.

ME-Q64) Given that the file Test.java, which defines the following code, fails to compile, select the reasons for the compilation failure (choose 2 options).

```
class Person {
    Person(String value) {
    }
}
class Employee extends Person {
}
class Test {
    public static void main(String args[]) {
        Employee e = new Employee();
    }
}
```

A The class Person fails to compile.

B The class Employee fails to compile.

C The default constructor can call only a no-argument constructor of a base class.

D The code that creates the object of the class Employee in the class Test did not pass a String value to the constructor of the class Employee.

Answer: B, C

Explanation: The class Employee doesn't compile, so the class Test can't use a variable of type Employee, and it fails to compile.

While trying to compile the class Employee, the Java compiler generates a default constructor for it, which looks like the following:

```
Employee(){
        super();
}
```

Note that a derived class constructor must always call a base class constructor. When Java generates the previous default constructor for the class Employee, it fails to compile because the base class doesn't have a no-argument constructor. The default constructor that's generated by Java can only define a call to a no-argument constructor in the base class. It can't call any other base class constructor.

ME-Q65) Examine the following code and select the correct statements (choose 2 options).

```
class Bottle {
    void Bottle() {
    }

    void Bottle(WaterBottle w) {
    }
}

class WaterBottle extends Bottle {
}
```

A A base class can't pass reference variables of its defined class as method parameters in constructors.

B The class compiles successfully—a base class can use reference variables of its derived class as method parameters.

C The class Bottle defines two overloaded constructors.

D The class Bottle can access only one constructor.

Answer: B, D

Explanation: A base class can use reference variables and objects of its derived classes. Note that the methods defined in the class Bottle aren't constructors but regular methods with the name Bottle. The return type of a constructor isn't void.

ME-Q66) Given the following code, which option, if used to replace /* INSERT CODE HERE */, will cause the code to print 110? (Select 1 option.)

```
class Book {
    private int pages = 100;
}

class Magazine extends Book {
    private int interviews = 2;

    private int totalPages() { /* INSERT CODE HERE */ }

    public static void main(String[] args) {
        System.out.println(new Magazine().totalPages());
    }
}
```

A return super.pages + this.interviews*5;

B return this.pages + this.interviews*5;

C return super.pages + interviews*5;

D return pages + this.interviews*5;

E None of the above

Answer: E

Explanation: The variable pages has private access in the class Book, and it can't be accessed from outside this class.

ME-Q67) Given the following code,

```
    class NoInkException extends Exception {
    }

    class Pen {
        void write(String val) throws NoInkException {
            int c = (10 - 7) / (8 - 2 - 6);
        }
        void article() {
    //INSERT CODE HERE
        }
    }
```

which of the options, when inserted at //INSERT CODE HERE, will define a valid use of the method write in the method article? (Select 2 options.)

A
```
try{
new Pen().write("story");
}catch(NoInkException e){}
```

B
```
try{
new Pen().write("story");
}finally{}
```

C
```
try{
write("story");
}catch(Exception e){}
```

D
```
try{
new Pen().write("story");
}catch(RuntimeException e){}
```

Answer: A, C

Explanation: On execution, the method write will always throw an Arithmetic-Exception (a RuntimeException) due to division by 0. But this method declares to throw a NoInkException, which is a checked exception.

Because NoInkException extends the class Exception and not RuntimeException, NoInkException is a checked exception. When you call a method that throws a checked exception, you can either handle it using a try-catch block or declare it to be thrown in your method signature.

Option (a) is correct because a call to the method write is enclosed within a try block. The try block is followed by a catch block, which defines a handler for the exception NoInkException.

Option (b) is incorrect. The call to the method write is enclosed within a try block, followed by a finally block. The finally block isn't used to handle an exception.

Option (c) is correct. Because NoInkException is a subclass of Exception, an exception handler for the class Exception can handle the exception NoInkException as well.

Option (d) is incorrect. This option defines an exception handler for the class RuntimeException. Because NoInkException is not a subclass of RuntimeException, this code won't handle NoInkException.

ME-Q68) What is the output of the following code? (Select 1 option.)

```
class EMyMethods {
    static String name = "m1";

    void riverRafting() {
        String name = "m2";
        if (8 > 2) {
            String name = "m3";
            System.out.println(name);
        }
    }

    public static void main(String[] args) {
        EMyMethods m1 = new EMyMethods();
        m1.riverRafting();
    }
}
```

A m1

B m2

C m3

D The code fails to compile.

Answer: D

Explanation: The class EMyMethods defines three variables with the name name:

■ The static variable name with the value "m1"

■ The local variable name in the method riverRafting with the value "m2"

■ The variable name, local to the if block in the method riverRafting, with the value "m3".

The code fails to compile due to the definition of two local variables with the same name (name) in the method riverRafting. If this code were allowed to compile, the scope of these local variables would overlap—the variable name defined outside the if block would be accessible to the complete method riverRafting. The scope of the local variable name, defined within the if block, would be limited to the if block. Within the if block, how do you think the code would differentiate between these local variables? Because there's no way to do so, the code fails to compile.

ME-Q69) What is the output of the following code? (Select 1 option.)

```
class EBowl {
    public static void main(String args[]) {
        String eFood = "Corn";
        System.out.println(eFood);
```

```
            mix(eFood);
            System.out.println(eFood);
        }

        static void mix(String foodIn) {
            foodIn.concat("A");
            foodIn.replace('C', 'B');
        }
    }
```

A Corn

BornA

B Corn

CornA

C Corn

Born

D Corn

Corn

Answer: D

Explanation: String objects are immutable. This implies that using any method can't change the value of a String variable. In this case, the String object is passed to a method, which seems to, but doesn't, change the contents of String.

ME-Q70) Which statement is true for the following code? (Select 1 option.)

```
class SwJava {
    public static void main(String args[]) {
        String[] shapes = {"Circle", "Square", "Triangle"};
        switch (shapes) {
            case "Square":
                System.out.println("Circle");
                break;
            case "Triangle":
                System.out.println("Square");
                break;
            case "Circle":
                System.out.println("Triangle");
                break;
        }
    }
}
```

A The code prints Circle.

B The code prints Square.

C The code prints Triangle.

D The code prints

Circle

Square

Triangle

E The code prints

Triangle

Circle

Square

F The code fails to compile.

Answer: F

Explanation: The question tries to trick you; it passes a String[] value to a switch construct by passing it an array of String objects. The code fails to compile because an array isn't a valid argument to a switch construct. The code would have compiled if it passed an element from the array shapes (shapes[0], shapes[1], or shapes[2]).

ME-Q71) Given the following definition of the classes Person, Father, and Home, which option, if used to replace //INSERT CODE HERE, will cause the code to compile successfully? (Select 3 options.)

```
class Person {
}

class Father extends Person {
    public void dance() throws ClassCastException {
    }
}

class Home {
    public static void main(String args[]) {
        Person p = new Person();
        try {
            ((Father) p).dance();
        }
//INSERT CODE HERE
    }
}
```

A

```
catch(NullPointerException e){}
catch(ClassCastException e){}
catch(Exception e){}
catch(Throwable t){}
```

B

```
catch(ClassCastException e){}
catch(NullPointerException e){}
catch(Exception e){}
catch(Throwable t){}
```

C

```
catch(ClassCastException e){}
catch(Exception e){}
catch(NullPointerException e){}
catch(Throwable t){}
```

D

```
catch(Throwable t){}
catch(Exception e){}
catch(ClassCastException e){}
catch(NullPointerException e){}
```

E

```
finally{}
```

Answer: **A, B, E**

Explanation: Because NullPointerException and ClassCastException don't share a base class–derived class relationship, these can be placed before or after each other.

The class Throwable is the base class of Exception. Hence, the exception handler for the class Throwable can't be placed before the exception handler of the class Exception.

Similarly, Exception is a base class for NullPointerException and Class-CastException. Hence, the exception handler for the class Exception can't be placed before the exception handlers of the class ClassCastException or NullPointer-Exception.

Option (e) is OK because no checked exceptions are defined to be thrown.

ME-Q72) What is the output of the following code? (Select 1 option.)

```
import java.time.*;

class Camera {
    public static void main(String args[]) {
        int hours;
        LocalDateTime now = LocalDateTime.of(2020, 10, 01, 0, 0);
        LocalDate before = now.toLocalDate().minusDays(1);
        LocalTime after = now.toLocalTime().plusHours(1);
        while (before.isBefore(after) && hours < 4) {
            ++hours;
        }
        System.out.println("Hours:" + hours);
    }
}
```

A	The code prints Camera:null.

B	The code prints Camera:Adjust settings manually.

C	The code prints Camera:.

**D	The code will fail to compile.**

Answer: B

Explanation: Note the type of the variables now, before, and after—they aren't the same. The code fails compilation because the code before.isBefore(after) calls the isBefore method on an instance of LocalDate, passing it a LocalTime instance, which isn't allowed.
The local variable hours isn't initialized prior to being referred to in the while condition (hours < 4), which is another reason why the class doesn't compile.

ME-Q73) The output of the class TestEJavaCourse, defined as follows, is 300:

```
class Course {
    int enrollments;
}

class TestEJavaCourse {
    public static void main(String args[]) {
        Course c1 = new Course();
        Course c2 = new Course();
        c1.enrollments = 100;
        c2.enrollments = 200;
        System.out.println(c1.enrollments + c2.enrollments);
    }
}
```

What will happen if the variable enrollments is defined as a static variable? (Select 1 option.)

A	No change in output. TestEJavaCourse prints 300.

B	Change in output. TestEJavaCourse prints 200.

**C	Change in output. TestEJavaCourse prints 400.**

D	The class TestEJavaCourse fails to compile.

Answer: C

Explanation: The code doesn't fail compilation after the definition of the variable enrollments is changed to a static variable. A static variable can be accessed using a variable reference of the class in which it's defined. All the objects of a class share the same copy of the static variable. When the variable enrollments is modified using the reference variable c2, c1.enrollments is also equal to 200. Hence, the code prints the result of 200 + 200, that is, 400.

ME-Q74) What is the output of the following code? (Select 1 option.)

```
String ejgStr[]=new String[][]{{null},
new tring[]{"a","b","c"},
{new String()}}[0];
        String ejgStr1[]=null;
        String ejgStr2[]={null};
        System.out.println(ejgStr[0]);
        System.out.println(ejgStr2[0]);
        System.out.println(ejgStr1[0]);
```

A null

NullPointerException

B null

null

NullPointerException

C NullPointerException

D null

null

null

Answer: B

Explanation: The trickiest assignment in this code is the assignment of the variable ejgStr. The following line of code may *seem* to (but doesn't) assign a two-dimensional String array to the variable ejgStr:

String ejgStr[] = new String[][]{{null},new String[]{"a","b","c"},{new String()}}[0] ;

The preceding code assigns the first element of a two-dimensional String array to the variable ejgStr. The following indented code will make the previous statement easier to understand:

String ejgStr[] = new String[][]{

{null},

new String[]{"a","b","c"},

{new String()}

First element of two-dimensional

array—an array of length 1

Second element of two-dimensional

array—an array of length 3

Third element of twodimensional

array—

an array of length 1

}

[0] ;

So let's look at the simplified assignment:

String ejgStr[] = {null};

String ejgStr1[] = null;

String ejgStr2[] = {null};

Revisit the code that prints the array elements:

System.out.println(ejgStr[0]);

System.out.println(ejgStr2[0]);

System.out.println(ejgStr1[0]);

Because ejgStr refers to an array of length 1 ({null}), ejgStr[0] prints null.

ejgStr2 also refers to an array of length 1 ({null}), so ejgStr2[0] also prints null.

ejgStr1 refers to null, not to an array. An attempt to access the first element of ejgStr1 throws a NullPointerException.

ME-Q75) Examine the following code and select the correct statement (choose 1 option).

```
class Person {
}

class Emp extends Person {
}

class TestArrayList {
    public static void main(String[] args) {
        ArrayList<Object> list = new ArrayList<>();
        list.add(new String("1234")); //LINE1
        list.add(new Person()); //LINE2
        list.add(new Emp()); //LINE3
```

```
                list.add(new String[]{"abcd", "xyz"}); //LINE4
                list.add(LocalDate.now().plus(1)); //LINE5
            }
        }
```

A The code on line 1 won't compile.

B The code on line 2 won't compile.

C The code on line 3 won't compile.

D The code on line 4 won't compile.

E The code on line 5 won't compile.

F None of the above.

G All the options from (a) through (e).

Answer: E

Explanation: The type of an ArrayList determines the type of the objects that can be added to it. An ArrayList can add to it all the objects of its derived class. Options (a) to (d) will compile because the class Object is the superclass of all Java classes; the ArrayList list as defined in this question will accept all types of objects, including arrays, because they are also objects.

Although a LocalDate instance can be added to an ArrayList, the code in option (e) won't compile. LocalDate.now() returns a LocalDate instance. But the class LocalDate doesn't define a plus() method, which accepts an integer value to be added to it—there's actually one plus method that accepts a TemporalAmount instance. You can use any of the following methods to add days, months, weeks, or years to LocalDate, passing it long values:

■ plusDays(long daysToAdd)

■ plusMonths(long monthsToAdd)

■ plusWeeks(long weeksToAdd)

■ plusYears(long yearsToAdd)

You can also use the following method to add a duration of days to LocalDate, passing it an instance of Period (Period implements TemporalAmount):

plus(TemporalAmount amountToAdd)

ME-Q76) What is the output of the following code? (Select 1 option.)

```
public class If2 {
    public static void main(String args[]) {
        int a = 10;
        int b = 20;
        boolean c = false;
        if (b > a)
            if (++a == 10)
                if (c != true)
                    System.out.println(1);
                 else
                    System.out.println(2);
        else
           System.out.println(3);
    }
}
```

A 1

B 2

C 3

D No output

Answer: C

Explanation: The key to answering questions about unindented code is to indent it.

Here's how:

if (b > a)

if (++a == 10)

if (c!=true)

System.out.println(1);

else

System.out.println(2);

else System.out.println(3);

Now the code becomes much simpler to look at and understand. Remember that the last else statement belongs to the inner if (++a == 10). As you can see, if (++a == 10) evaluates to false, the code will print 3.

ME-Q77) Given the following code,

```
interface Movable {
    default int distance() {
        return 10;
    }
}

interface Jumpable {
    default int distance() {
        return 10;
    }
}
```

which options correctly define the class Person that implements interfaces Movable and Jumpable? (Select 1 option.)

A
```
class Person implements Movable, Jumpable {
}
```
B
```
class Person implements Movable, Jumpable {
    default int distance() {
        return 10;
    }
}
```
C
```
class Person implements Movable, Jumpable {
    public int distance() {
        return 10;
    }
}
```
D
```
class Person implements Movable, Jumpable {
    public long distance() {
        return 10;
    }
```
E
```
class Person implements Movable, Jumpable {
    int distance() {
        return 10;
    }
}
```
Answer: C

Explanation: Option (a) is incorrect because the class Person can't implement both interfaces, Jumpable and Movable, which define the method distance with the same signatures and a default implementation. The class Person must override the default implementation of distance() to implement both these interfaces.

Option (b) is incorrect. When a class overrides the default implementation of a method that it inherits from an interface, it can't use the keyword default. Such code won't compile.

Option (d) is incorrect. The method distance() with the return type long can't override distance() with the return type int.

Option (e) is incorrect and this code won't compile. The class Person is trying to decrease the accessibility of distance(), from public to the *default* access level.

Good Luck