8 Full mock exam

This chapter covers

- Complete mock exam with 90 questions
- Answers to all mock exam questions with extensive explanations and the subobjective on which each exam question is based

On the real exam, each question displays the count of correct options that you should select. The exam engine won't allow you to select more answer options than are specified by this number. If you try to do so, a warning will be displayed. The questions in this mock exam also specify the correct number of answer options to align it more closely to the real exam.

8.1 Mock exam

ME-Q1. Given the following definition of the classes `Animal`, `Lion`, and `Jumpable`, select the correct combinations of assignments of a variable (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();`

ME-Q2. Which of the following statements are true? (Select 3 options.)

- **a** A Java class can define multiple methods.
- **b** A Java class can define multiple variables.
- **c** A Java class can be defined in multiple packages.
- **d** A Java class can import multiple packages.
- **e** A Java class can't define more than 108 constructors.
- **f** End-of-line comments can't follow `import` or `package` statements.
- **g** Multiline comments can only be defined within a method definition.

ME-Q3. 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`

ME-Q4. What is the output of the following code? (Select 1 option.)

```
int a = 10; String name = null;
try {
    a = name.length();
    a++;
}
catch (RuntimeException e){
    ++a;
}
System.out.println(a);
```

- **a** `5`
- **b** `6`
- **c** `10`
- **d** `11`
- **e** `12`
- **f** Runtime exception

ME-Q5. 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

ME-Q6. Given the following code, which code can be used to create and initialize an object of 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");`

ME-Q7. 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.

ME-Q8. 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`.

ME-Q9. 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 imports 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 {}
```

ME-Q10. 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() {
```

ME-Q11. Which of the following statements are true? (Select 2 options.)

- **a** Given that you changed the access of a `public` method `B`, in class `A`, to a `private` method, class `C` that uses method `B` will fail to compile.
- **b** Given that you changed the access of a `private` method `B`, in class `A`, to a `public` method, none of the classes that use class `A` will fail to compile.
- **c** Given that you changed the access of a `protected` method `B`, in class `A`, to a method with default access, class `C` from the same package as class `A` won't be able to access method `B`.
- **d** A change in the accessibility of the methods in your class never affects any other class that uses your class.

ME-Q12. Which of the following statements are correct? (Select 3 options.)

- **a** You may not be able to handle all the checked exceptions in your code.
- **b** If required, you can handle all the runtime exceptions in your code.
- **c** You can handle an exception in your code even if you don't know its exact name.
- **d** A single exception handler can be used to handle all types of runtime and checked exceptions.
- **e** You must handle all errors that can be thrown by your code.
- **f** Runtime exceptions are also known as checked exceptions.

ME-Q13. Given the following code,

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

what's 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** `31`
- **j** `4`

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

```
class Person {
    int age;
    float height;
    boolean result;
    String name;
}
public class EJava {
    public static void main(String arguments[]) {
        Person person = new Person();
        System.out.println(person.name + person.height + person.result
                                    + person.age);
    }
}
```

- **a** `null0.0false0`
- **b** `null0false0`

- **c** `null0.0ffalse0`
- **d** `0.0false0`
- **e** `0false0`
- **f** `0.0ffalse0`
- **g** `null0.0true0`
- **h** `0true0`
- **i** `0.0ftrue0`

ME-Q15. 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`

ME-Q16. Which statements describe the use of exception handling in Java? (Select 2 options.)

- **a** Exception handling can prevent an application from crashing or producing incorrect outputs or incorrect input values.
- **b** Exception handlers can't define an alternative flow of action in the event of an exception.
- **c** Exception handlers enable a programmer to define separate code blocks for handling different types of exceptions.
- **d** Exception handlers help to define well-encapsulated classes.
- **e** Exception handlers help with efficient inheritance.

ME-Q17. Which statements are true for reference and primitive variables? (Select 3 options.)

- **a** The names of the reference variables are limited to a length of 256 characters.
- **b** There is no limit on the length of the names of primitive variables.
- **c** Multiple reference variables may refer to exactly the same object in memory.
- **d** Values stored by primitive and reference variables can be compared for equality by using the equals operator (`==`) or by using the method `equals`.
- **e** A primitive variable can't refer to an object and vice versa.

ME-Q18. What is the output of the following code? (Select 1 option.)

```
public class Handset {
    public static void main(String... args) {
        double price;
        String model;
        System.out.println(model + price);
    }
}
```

- **a** null0
- **b** null0.0
- **c** 0
- **d** 0.0
- **e** Compilation error

ME-Q19. 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

ME-Q20. Which of the following options defines the correct structure of a Java class? (Select 1 option.)

- **a**
```
package com.ejava.guru;
package com.ejava.oracle;
class MyClass { }
```
- **b**
```
import com.ejava.guru.*;
import com.ejava.oracle.*;
package com.ejava;
class MyClass { }
```
- **c**
```
class MyClass {
    import com.ejava.guru.*;
}
```
- **d**
```
class MyClass {
    int abc;
}
```

ME-Q21. 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.

ME-Q22. 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.

ME-Q23. 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 9.

ME-Q24. Which of the following is the correct declaration of a method that accepts two `String` arguments and an `int` argument and doesn't return any value? (Select 2 options.)

- **a** `void myMethod(String str1, int str2, String str3)`
- **b** `myMethod(String val1, int val2, String val3)`
- **c** `void myMethod(String str1, str2, int a)`
- **d** `void myMethod(String val1, val2, int val3)`
- **e** `void myMethod(int str2, String str3, String str1)`

ME-Q25. 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};`

ME-Q26. Assume that Oracle has asked you to create a method that returns the concatenated value of two `String` objects. Which of the following methods do you think can accomplish this job? (Select 2 options.)

- **a**
```
public String add(String 1, String 2) {
    return str1 + str2;
}
```
- **b**
```
private String add(String s1, String s2) {
    return s1.concat(s2);
}
```
- **c**
```
protected String add(String value1, String value2) {
    return value2.append(value2);
}
```
- **d**
```
String subtract(String first, String second) {
    return first.concat(second.substring(0));
}
```

ME-Q27. In Java, the class `String` is defined in the package `java.lang`, and this package is automatically imported in all the Java classes. Given this statement, which of the following options represents the correct definition of class `EJava`, which can define a variable of class `String`? (Choose 2 options.)

a
```
import java.lang;
class EJava {
    String guru;
}
```

b
```
import java.lang.String.*;
class EJava {
    String guru;
}
```

c
```
class EJava {
    String guru;
}
```

d
```
import java.lang.String;
import java.lang.String;
class EJava {
    String guru;
}
```

ME-Q28. Given the following definitions of the class `ChemistryBook`, select the statements that are correct individually (choose 2 options):

```
import java.util.ArrayList;
class ChemistryBook extends Book {
    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.

ME-Q29) Given the following definition of the class `Home`, select the correct statements (choose 4 options):

```
class Home {
    String name;
    int rooms;
    Home() {}
}
```

- **a** The class `Home` will be provided a default constructor.
- **b** The class `Home` won't be provided a default constructor.
- **c** A default constructor can't coexist with overloaded constructors.
- **d** A default constructor doesn't accept any method parameters.
- **e** After compilation, the class `Home` has only a no-argument constructor.

- f After compilation, the class `Home` has two constructors: a no-argument constructor and a default constructor.
- g When an object of class `Home` is created, its variables `name` and `rooms` are not assigned any default values.

ME-Q30. 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;`

ME-Q31. Ideally, which of the following should never be handled by an exception handler? (Select 2 options.)

- a `StackOverflowError`
- b `OutOfMemoryError`
- c `ArrayIndexOutOfBoundError`
- d `ClassLoadingException`
- e `CompilationError`
- f `OutOfStorageError`

ME-Q32. What is the output of the following code? (Select 1 option.)

```
public class MyCalendar {
    public static void main(String arguments[]) {
        Season season1 = new Season();
        season1.name = "Spring";

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

        season1 = season2;
        System.out.println(season1.name);
        System.out.println(season2.name);
    }
}
class Season {
    String name;
}
```

a
```
String
Autumn
```

b
```
Spring
String
```

c
```
Autumn
Autumn
```

d
```
Autumn
String
```

ME-Q33. What is true about the following code? (Select 1 option.)

```
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) {}
```

ME-Q34. 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 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;`

ME-Q35. 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.

ME-Q36. Given the following code, which option, if used to replace `/* INSERT CODE HERE */`, will make the code print the value of the variable `screenSize`? (Select 1 option.)

```
class Tablet {
    float screenSize = 7.0f;
    float getScreenSize() {
        return screenSize;
    }
    void setScreenSize(float val) {
        screenSize = val;
    }
}

class DemoTabs {
    public static void main(String args[]) {
        Tablet tab = new Tablet();
        System.out.println(/* INSERT CODE HERE */);

    }
}
```

- **a** `tab.screenSize`
- **b** `tab->getScreensize()`
- **c** `tab::getScreen()`
- **d** `tab:screenSize`

ME-Q37. Given the following definitions of the class `Person` and the interface `Movable`, the task is to declare a class `Emp` that inherits from the class `Person` and implements the interface `Movable`. Select the correct option to accomplish this task (choose 1 option):

```
class Person {}
interface Movable {}
```

- **a** `class Emp implements Person extends Movable{}`
- **b** `class Emp implements Person, Movable{}`
- **c** `class Emp extends Person implements Movable{}`
- **d** `class Emp extends Person, Movable{}`

ME-Q38. 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
  ```

ME-Q39. 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 will print `caught` if `question()` throws exception `NullPointerException`.

ME-Q40. 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.

ME-Q41. 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

ME-Q42. 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 = "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.memory = "2GB";
    }
}
```

- **a** 1 GB
- **b** 2 GB
- **c** Compilation error
- **d** Runtime exception

ME-Q43. 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

ME-Q44. 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

ME-Q45. 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.

ME-Q46. 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`.

ME-Q47. 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 chars.
- **b** `StringBuilder sb1 = new StringBuilder(5*10)` will create a `StringBuilder` object with a value `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 at a specified position of a `StringBuilder` object.

ME-Q48. 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];`

ME-Q49. What is the output of the following code? (Select 1 option.)

```
import java.util.*;
class EJGArrayL {
    public static void main(String args[]) {
        ArrayList<String> seasons = new ArrayList<String>();
        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

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

```
class EIf {
    public static void main(String args[]) {
        bool boolean = false;
        if (boolean = true)
            System.out.println("true");
```

```
        else
            System.out.println("false");
    }
}
```

- **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.

ME-Q51. 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.

ME-Q52. Given the following code, which option, if used to replace `/* INSERT CODE HERE */`, will make the code print the name of the phone with the position at which it's stored in the array `phone`? (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

ME-Q53. Which of the following classes represent runtime exceptions in Java (select 4 options):

- [ ] **a** `RuntimeException`
- [ ] **b** `CheckedException`
- [ ] **c** `NullPointerException`
- [ ] **d** `ArrayIndexOutOfBoundsException`
- [ ] **e** `CompilationException`
- [ ] **f** `Throwable`
- [ ] **g** `StackOverflowException`
- [ ] **h** `MemoryOutOfBoundsException`
- [ ] **i** `IllegalArgumentException`
- [ ] **j** `NumberException`

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

```
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");
        System.out.print(b1.equals(b2) +":");
        System.out.print(b1 == b2);
    }
}
```

- ( ) **a** `true:false`
- ( ) **b** `true:true`
- ( ) **c** `false:true`
- ( ) **d** `false:false`

ME-Q55. 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

ME-Q56. 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; }`

ME-Q57. 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 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;
```

ME-Q58. 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;
        if (!b++ > 100 && a++ == 10) {
            System.out.println(a+b);
        }
    }
}
```

- **a** 100
- **b** 101
- **c** 102
- **d** Code fails to compile.
- **e** No output is produced.

ME-Q59. Given the following definitions of the interfaces `Movable` and `Jumpable`, the task is to declare a class `Person` that inherits both of these interfaces. Which of the following code snippets will accomplish this task? (Select 2 options.)

```
interface Movable {}
interface Jumpable {}
```

- **a**
```
interface Movable {}
interface Jumpable {}
class Person implements Movable, Jumpable {}
```
- **b**
```
interface Movable {}
interface Jumpable {}
class Person extends Movable, Jumpable {}
```
- **c**
```
interface Movable {}
interface Jumpable {}
class Person implements Movable extends Jumpable {}
```
- **d**
```
interface Movable {}
interface Jumpable implements Movable {}
class Person implements Jumpable {}
```
- **e**
```
interface Movable {}
interface Jumpable extends Movable {}
class Person implements Jumpable {}
```

ME-Q60. 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; }
}
```

ME-Q61. 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
```

ME-Q62. Given the following requirements, choose the best looping construct to implement them (choose 1 option):

Step 1: Meet the director of Oracle.
Step 2: Schedule another meeting with the director.
Step 3: Repeat steps 1 and 2, as long as more meetings are required.

a `for` loop
b enhanced `for` loop
c `do-while` loop
d `while` loop

ME-Q63. 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

ME-Q64. 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

ME-Q65. 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.

ME-Q66. 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.

ME-Q67. 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"}};`

ME-Q68. 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

ME-Q69. 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 class `Phone`.

ME-Q70. Which of the following statements are incorrect about *the* `main` method used to start a Java application? (Select 2 options.)

- **a** A class can't define multiple `main` methods.
- **b** More than one class in an application can define the `main` method.
- **c** The `main` method may accept a `String`, a `String` array, or varargs (`String... arg`) as a method argument.
- **d** The `main` method shouldn't define an object of the class in which the `main` method itself is defined.

ME-Q71. What is the output of the following code? (Select 1 option.)

```
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`

ME-Q72. 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 correctly if line 4 is changed to the following:

  ```
  StringBuilder ejg = null;
  ```

ME-Q73. When considered individually, which of the options is correct for the following code? (Select 1 option.)

```
interface Jumpable { void jump(); }
class Chair implements Jumpable {
    public void jump() {
        System.out.println("Chair cannot jump");
    }
}
```

- **a** The class `Chair` can't implement the interface `Jumpable` because a `Chair` can't define a method `jump`.
- **b** If the name of the interface is changed to `Movable` and the definition of class `Chair` is updated to `class Chair implements Movable`, class `Chair` will compile successfully.

- **c** If the definition of the method `jump` is removed from the definition of the class `Chair`, it will compile successfully.
- **d** If the name of the method `jump` is changed to `run` in the interface `Jumpable`, the class `Chair` will compile successfully.

ME-Q74. 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)`

ME-Q75. 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** Code that creates an object of class `Employee` in class `Test` didn't pass a `String` value to the constructor of class `Employee`.

ME-Q76. Select the correct statements. (Choose 4 options.)

- **a** Checked exceptions are subclasses of `java.lang.Throwable`.
- **b** Runtime exceptions are subclasses of `java.lang.Exception`.
- **c** Errors are subclasses of `java.lang.Throwable`.
- **d** `java.lang.Throwable` is a subclass of `java.lang.Exception`.
- **e** `java.lang.Exception` is a subclass of `java.lang.Error`.

- f Errors aren't subclasses of `java.lang.Exception`.
- g `java.lang.Throwable` is a subclass of `java.lang.Error`.
- h Checked exceptions are subclasses of `java.lang.CheckedException`.

ME-Q77. 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.

ME-Q78. 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

ME-Q79. Given that the method `write` has been defined as follows,

```
class NoInkException extends Exception {}
class Pen{
    void write(String val) throws NoInkException {
        //.. some code
    }
    void article() {
        //INSERT CODE HERE
    }
}
```

which of the following options, when inserted at `//INSERT CODE HERE`, will define 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) {}
```

ME-Q80. 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** Code fails to compile.

ME-Q81. 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`

ME-Q82. 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.

ME-Q83. Which of the following options include the ideal conditions for choosing to use a `do-while` loop over a `while` loop? (Select 2 options.)

- **a** Repeatedly display a menu to a user and accept input until the user chooses to exit the application.
- **b** Repeatedly allow a student to sit in the exam only if she carries her identity card.
- **c** Repeatedly serve food to a person until he wants no more.
- **d** Repeatedly allow each passenger to board an airplane if the passengers have their boarding passes.

ME-Q84. Given the following definition of the classes `Person`, `Father`, and `Home`, which options, 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 {}
```

ME-Q85. What is the output of the following code? (Select 1 option.)

```
class Camera {
    public static void main(String args[]) {
        String settings;
        while (false) {
            settings = "Adjust settings manually";
        }
        System.out.println("Camera:" + settings);
    }
}
```

- **a** The code prints `Camera:null`.
- **b** The code prints `Camera:Adjust settings manually`.
- **c** The code will print `Camera:`.
- **d** The code will fail to compile.

ME-Q86. 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.

ME-Q87. What is the output of the following code? (Select 1 option.)

```
String ejgStr[] = new String[][]{{null},new String[]{"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

ME-Q88. Examine the following code and select the correct statement (choose 1 option):

```
import java.util.*;
class Person {}
class Emp extends Person {}

class TestArrayList {
    public static void main(String[] args) {
        ArrayList<Object> list = new ArrayList<Object>();
```

```
        list.add(new String("1234"));       //LINE1
        list.add(new Person());             //LINE2
        list.add(new Emp());                //LINE3
        list.add(new String[]{"abcd", "xyz"});//LINE4
    }
}
```

- 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 None of the above.

ME-Q89. 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

ME-Q90. Select the incorrect statement (choose 1 option):

- a An enhanced `for` loop can be used to iterate through the elements of an array and `ArrayList`.
- b The loop counter of an enhanced `for` loop can be used to modify the current element of the array being iterated over.
- c `do-while` and `while` loops can be used to iterate through the elements of an array and `ArrayList`.
- d The loop counter of a regular `for` loop can be used to modify the current element of an `ArrayList` being iterated over.

8.2 Answers to mock exam questions

This section contains answers to all the mock exam questions in section 8.1. Also, each question is preceded by the exam objective that the question is based on.

[7.3] Differentiate between the type of a reference and the type of an object

ME-Q1. Given the following definition of the classes `Animal`, `Lion`, and `Jumpable`, select the correct combinations of assignments of a variable (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();`**

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`.

[1.2] Define the structure of a Java class

ME-Q2. Which of the following statements are true? (Select 3 options.)

- ☑ **a** **A Java class can define multiple methods.**
- ☑ **b** **A Java class can define multiple variables.**
- ☐ **c** A Java class can be defined in multiple packages.
- ☑ **d** **A Java class can import multiple packages.**
- ☐ **e** A Java class can't define more than 108 constructors.
- ☐ **f** End-of-line comments can't follow `import` or `package` statements.
- ☐ **g** Multiline comments can only be defined within a method definition.

Answer: a, b, d

Explanation: Option (c) is incorrect. The same class can't be defined in multiple packages. If you try to define the same class—say, the class `Person`—in the packages `com.ejava` and `com.eoracle`, you're defining two classes with the same name but in separate packages. In this case, `com.ejava.Person` and `com.eoracle.Person` will refer to two different classes.

Option (e) is incorrect because there is no theoretical limit on the number of constructors that can be defined by a class.

Option (f) is incorrect because end-of-line comments can follow any line of code.

Option (g) is incorrect because multiline comments can also be placed outside a method definition.

[8.2] Create a try-catch block and determine how exceptions alter normal program flow

ME-Q3. 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 `ArrayIndexOutOfBoundsException`.

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

[8.2] Create a try-catch block and determine how exceptions alter normal program flow

ME-Q4. What is the output of the following code? (Select 1 option.)

```
int a = 10; String name = null;
try {
    a = name.length();
    a++;
}
catch (RuntimeException e){
    ++a;
}
System.out.println(a);
```

- **a** 5
- **b** 6
- **c** 10
- **d** **11**
- **e** 12
- **f** Runtime exception

Answer: d

Explanation: Because the variable `name` isn't assigned a value, the following line of code will throw `NullPointerException`:

```
name.length();
```

Hence, the original value of the variable `a` isn't modified and the control is transferred to the exception handler, which increments the value of the variable `a` to `11`.

[3.5] Use a switch statement

ME-Q5. 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.

[7.5] Use super and this to access objects and constructors

ME-Q6. Given the following code, which code can be used to create and initialize an object of 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);`
- [x] **c** **`ColorPencil var3 = new ColorPencil("RED");`**
- [x] **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 class `ColorPencil`, which isn't defined in class `ColorPencil`.

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

[2.3] Read or write to object fields

ME-Q7. 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).

[3.1] Use Java operators

ME-Q8. 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.

[1.4] Import other Java packages to make them accessible in your code

ME-Q9. 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 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.

[6.6] Apply access modifiers

ME-Q10. 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.

[6.6] Apply access modifiers

ME-Q11. Which of the following statements are true? (Select 2 options.)

- ☑ **a Given that you changed the access of a `public` method `B`, in class `A`, to a `private` method, class `C` that uses method `B` will fail to compile.**

- [x] **b Given that you changed the access of a `private` method `B`, in class `A`, to a `public` method, none of the classes that use class `A` will fail to compile.**
- [ ] c Given that you changed the access of a `protected` method `B`, in class `A`, to a method with default access, class `C` from the same package as class `A` won't be able to access method `B`.
- [ ] d A change in the accessibility of the methods in your class never affects any other class that uses your class.

Answer: a, b

Explanation: Option (a) is correct. A `public` method is accessible to other classes. If you change the accessibility of a `public` method to a `private` method, it will no longer be accessible outside its class. Any class that uses such a method will fail to compile after this modification.

Option (b) is correct. A `private` method isn't accessible outside the class in which it's defined. In other words, a `private` method isn't known to the other classes, so it can't be accessed by other classes. If classes can't even access the private methods of other classes, it won't make a difference to them if their accessibility is changed.

Option (c) is incorrect. A method with default access can be accessed by classes defined in the same package.

Option (d) is incorrect. A change in the accessibility of the methods in your class affects other classes that use your class. If you assign a weaker accessibility to any of your methods, it may no longer be accessible to the other classes. If this happens, the other class will fail to compile.

[8.1] Differentiate among checked exceptions, RuntimeExceptions, and Errors

ME-Q12. Which of the following statements are correct? (Select 3 options)

- [ ] a You may not be able to handle all the checked exceptions in your code.
- [x] b **If required, you can handle all the runtime exceptions in your code.**
- [x] c **You can handle an exception in your code even if you don't know its exact name.**
- [x] d **A single exception handler can be used to handle all types of runtime and checked exceptions.**
- [ ] e You must handle all errors that can be thrown by your code.
- [ ] f Runtime exceptions are also known as checked exceptions.

Answer: b, c, d

Explanation: Option (a) is incorrect. If, for example, `callingMethod()` calls `calledMethod()`, which throws checked exceptions, `callingMethod()` can't ignore checked exceptions thrown by `calledMethod()`. `callingMethod()` should handle these exceptions itself or declare them to be thrown. `callingMethod()` can't ignore *any* checked exceptions thrown by `calledMethod()`. If it tries to do so, the code won't compile.

Option (b) is correct. It is indeed possible to handle all the runtime exceptions in your code.

Options (c) and (d) are correct. The superclass of all types of exceptions (checked and runtime) is class `java.lang.Exception`, so if you define a handler for `java.lang.Exception` in your code, you are able to handle all runtime and checked exceptions, and this will include any exceptions whose names you don't know.

Option (e) is incorrect. Even though errors can be caught by an exception handler, you shouldn't handle them because they're serious exceptions thrown by the JVM as a result of an error in the environment state that processes your code.

Option (f) is incorrect because runtime exceptions are also known as unchecked exceptions.

[1.3] Create executable Java applications with a main method

ME-Q13. Given the following code,

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

what's 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** `31`
- **j** `4`

Answer: h

Explanation: This question tests you on multiple points:

- *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
```

- *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`.
- `+` *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`.

[2.2] Differentiate between object reference variables and primitive variables

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

```
class Person {
    int age;
    float height;
    boolean result;
    String name;
}
public class EJava {
    public static void main(String arguments[]) {
        Person person = new Person();
        System.out.println(person.name + person.height + person.result
                                       + person.age);
    }
}
```

- **a** **`null0.0false0`**
- **b** `null0false0`
- **c** `null0.0ffalse0`
- **d** `0.0false0`
- **e** `0false0`
- **f** `0.0ffalse0`
- **g** `null0.0true0`
- **h** `0true0`
- **i** `0.0ftrue0`

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
```

[7.4] Determine when casting is necessary

ME-Q15. 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` 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.

[8.3] Describe what exceptions are used for in Java

ME-Q16. Which statements describe the use of exception handling in Java? (Select 2 options.)

- ☑ a **Exception handling can prevent an application from crashing or producing incorrect outputs or incorrect input values.**
- ☐ b Exception handlers can't define an alternative flow of action in the event of an exception.

- ☑ **c Exception handlers enable a programmer to define separate code blocks for handling different types of exceptions.**
- ☐ d Exception handlers help to define well-encapsulated classes.
- ☐ e Exception handlers help with efficient inheritance.

Answer: a, c

Explanation: Option (b) is incorrect. One of the main purposes of an exception handler is to define an alternative flow of action.

Options (d) and (e) are incorrect. Definitions of exception handlers, well-encapsulated classes, and efficient inheritance aren't related in behavior as stated by these options.

[2.2] Differentiate between object reference variables and primitive variables

ME-Q17. Which statements are true for reference and primitive variables? (Select 3 options.)

- ☐ a The names of the reference variables are limited to a length of 256 characters.
- ☑ b **There is no limit on the length of the names of primitive variables.**
- ☑ c **Multiple reference variables may refer to exactly the same object in memory.**
- ☐ d Values stored by primitive and reference variables can be compared for equality by using the equals operator (`==`) or by using the method `equals`.
- ☑ e **A primitive variable can't refer to an object and vice versa.**

Answer: b, c, e

Explanation: Option (a) is incorrect. Theoretically, there is no limit on the number of characters that can be used to define the name of a primitive variable or object reference.

Option (d) is incorrect. Unlike object references, primitive variables can be compared for equality by using the equals operator (`==`) only.

Option (e) is correct. A primitive variable can never refer to an object and vice versa.

[2.1] Declare and initialize variables

ME-Q18. What is the output of the following code? (Select 1 option.)

```
public class Handset {
    public static void main(String... args) {
        double price;
        String model;
        System.out.println(model + price);
    }
}
```

- ◯ a `null0`
- ◯ b `null0.0`
- ◯ c `0`

- **d** `0.0`
- **e** **Compilation error**

Answer: e

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.

[3.1] Use Java operators

ME-Q19. 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 is 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 is 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 is incremented by `1`, to `4`. This means that the expression evaluates to

```
1 + 3 + 4
```

The previous expression evaluates to 8.

[1.2] Define the structure of a Java class

ME-Q20. Which of the following options defines the correct structure of a Java class? (Select 1 option.)

a
```
package com.ejava.guru;
package com.ejava.oracle;
class MyClass { }
```

b
```
import com.ejava.guru.*;
import com.ejava.oracle.*;
package com.ejava;
class MyClass { }
```

c
```
class MyClass {
    import com.ejava.guru.*;
}
```

d
```
class MyClass {
    int abc;
}
```

Answer: d

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

Option (b) is incorrect. Though 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.

[3.2] Use parentheses to override operator precedence

ME-Q21. 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.

[1.1] Define the scope of variables

ME-Q22. 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 class `EJava`, and methods `calc` and `main`.

[2.4] Explain an object's life cycle

ME-Q23. 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 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 see, the `Emp` object creation calls itself recursively, resulting in `java.lang.StackOverflowError`.

[6.1] Create methods with arguments and return values

ME-Q24. Which of the following is the correct declaration of a method that accepts two `String` arguments and an `int` argument and doesn't return any value? (Select 2 options.)

- ☑ **a** **`void myMethod(String str1, int str2, String str3)`**
- ☐ **b** `myMethod(String val1, int val2, String val3)`
- ☐ **c** `void myMethod(String str1, str2, int a)`
- ☐ **d** `void myMethod(String val1, val2, int val3)`
- ☑ **e** **`void myMethod(int str2, String str3, String str1)`**

Answer: a, e

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 should be `void` to specify that the method doesn't return any value.

Option (b) is incorrect. It won't compile because the method signature doesn't include a return type.

Options (c) and (d) are incorrect. The method signatures of these methods don't define data types for all their method parameters.

[4.1] Declare, instantiate, initialize, and use a one-dimensional array

ME-Q25. 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.

[6.1] Create methods with arguments and return values

ME-Q26. Assume that Oracle has asked you to create a method that returns the concatenated value of two `String` objects. Which of the following methods do you think can accomplish this job? (Select 2 options.)

- ☐ **a**
```
public String add(String 1, String 2) {
    return str1 + str2;
}
```
- ☑ **b**
```
private String add(String s1, String s2) {
    return s1.concat(s2);
}
```

☐ **c**
```
protected String add(String value1, String value2) {
    return value2.append(value2);
}
```

☑ **d**
```
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.

[1.4] Import other Java packages to make them accessible in your code

ME-Q27. In Java, the class `String` is defined in the package `java.lang`, and this package is automatically imported in all the Java classes. Given this statement, which of the following options represents the correct definition of class `EJava`, which can define a variable of class `String`? (Choose 2 options.)

☐ **a**
```
import java.lang;
class EJava {
    String guru;
}
```

☐ **b**
```
import java.lang.String.*;
class EJava {
    String guru;
}
```

☑ **c**
```
class EJava {
    String guru;
}
```

☑ **d**
```
import java.lang.String;
import java.lang.String;
class EJava {
    String guru;
}
```

Answer: c, d

Explanation: Options (a) and (b) are incorrect. The code in both these options won't compile because they use incorrect `import` statement. The following line of code will import all the classes from package `java.lang` (including class `String`):

```
import java.lang.*;
```

You can use the following `import` statement to import just the class `String`:

```
import java.lang.String;
```

Option (c) is correct. The class `EJava` can create variables of the class `String` because the class `java.lang.String` is automatically imported in all the Java classes. Hence, it's available to `EJava`, even if this class doesn't import it explicitly.

Option (d) is correct. It doesn't make a difference if you import the same class more than once in your code.

[6.3] Create an overloaded method

ME-Q28. 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` only differ 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.

[6.4] Differentiate between default and user-defined constructors

ME-Q29. Given the following definition of the class `Home`, select the correct statements (choose 4 options):

```
class Home {
    String name;
```

```
    int rooms;
    Home() {}
}
```

- [ ] **a** The class `Home` will be provided a default constructor.
- [x] **b** **The class `Home` won't be provided a default constructor.**
- [x] **c** **A default constructor can't coexist with overloaded constructors.**
- [x] **d** **A default constructor doesn't accept any method parameters.**
- [x] **e** **After compilation, the class `Home` has only a no-argument constructor.**
- [ ] **f** After compilation, the class `Home` has two constructors: a no-argument constructor and a default constructor.
- [ ] **g** When an object of class `Home` is created, its variables `name` and `rooms` are not assigned any default values.

Answer: b, c, d, e

Explanation: Option (b) is correct. The class `Home` doesn't contain a default constructor. A default constructor is generated by Java when the user doesn't define any constructor. In this case, the class `Home` does define a constructor.

Option (c) is correct. A default constructor is generated only in the absence of a constructor. Hence, it can't coexist with other constructors.

Option (d) is correct. The default constructor doesn't accept any method parameters. It initializes the instance variables of a class to their default values.

Option (e) is correct and (f) is incorrect. No default constructor will be generated for class `Home` because `Home` already defines a no-argument constructor. A no-argument constructor is a constructor that defines no method parameters. After compilation, class `Home` has only one constructor that doesn't accept any method parameters.

Option (g) is incorrect. If you don't assign explicit values to instance variables of a class, they are initialized to their default values.

[5.5] Use break and continue

ME-Q30. 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 during the first iteration of the `for` loop so that no output is printed.

[8.5] Recognize common exception classes and categories

ME-Q31. Ideally, which of the following should never be handled by an exception handler? (Select 2 options.)

- [x] **a** **`StackOverflowError`**
- [x] **b** **`OutOfMemoryError`**
- [ ] **c** `ArrayIndexOutOfBoundError`
- [ ] **d** `ClassLoadingException`
- [ ] **e** `CompilationError`
- [ ] **f** `OutOfStorageError`

Answer: a, b

Explanation: Options (c), (d), (e), and (f) are incorrect because the Java API doesn't define these exception or error classes.

Options (a) and (b) are correct. You should never try to handle these errors in your code because `StackOverflowError` and `OutOfMemoryError` are serious errors.

[2.1] Declare and initialize variables

ME-Q32. What is the output of the following code? (Select 1 option.)

```
public class MyCalendar {
    public static void main(String arguments[]) {
        Season season1 = new Season();
        season1.name = "Spring";

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

        season1 = season2;
        System.out.println(season1.name);
        System.out.println(season2.name);
    }
}
class Season {
    String name;
}
```

- ○ **a** `String`
  `Autumn`

- **b** Spring
  String
- **c** **Autumn**
  Autumn
- **d** Autumn
  String

Answer: c

Explanation: 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 is 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 previous code is as follows:

```
Autumn
Autumn
```

[6.5] Create and overload constructors

ME-Q33. What is true about the following code? (Select 1 option.)

```
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 the 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.

[7.5] Use super and this to access objects and constructors

ME-Q34. 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 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 variable `RED`, which isn't defined in the code.

[2.3] Read or write to object fields

ME-Q35. 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.

[2.5] Call methods on objects

ME-Q36. Given the following code, which option, if used to replace `/* INSERT CODE HERE */`, will make the code print the value of the variable `screenSize`? (Select 1 option.)

```
class Tablet {
    float screenSize = 7.0f;
    float getScreenSize() {
        return screenSize;
    }
    void setScreenSize(float val) {
        screenSize = val;
    }
}

class DemoTabs {
    public static void main(String args[]) {
        Tablet tab = new Tablet();
        System.out.println(/* INSERT CODE HERE */);

    }
}
```

- **a** **`tab.screenSize`**
- **b** `tab->getScreensize()`
- **c** `tab::getScreen()`
- **d** `tab:screenSize`

Answer: a

Explanation: Only the dot operator (`.`) can be used to access an instance variable or a method of an object in Java. The rest of the operators (`->`, `::`, and `:`) used in options (b), (c), and (d), respectively, aren't valid operators in Java.

[7.1] Implement inheritance

ME-Q37. Given the following definitions of the class `Person` and the interface `Movable`, the task is to declare a class `Emp` that inherits from the class `Person` and implements the interface `Movable`. Select the correct option to accomplish this task (choose 1 option):

```
class Person {}
interface Movable {}
```

- **a** `class Emp implements Person extends Movable{}`
- **b** `class Emp implements Person, Movable{}`
- **c** **`class Emp extends Person implements Movable{}`**
- **d** `class Emp extends Person, Movable{}`

Answer: c

Explanation: Options (a) and (b) are incorrect because a `class` can't use the keyword `implements` to inherit a `class`.

Option (d) is incorrect because a class can't use the keyword `extends` to inherit an `interface`.

[7.2] Develop code that demonstrates the use of polymorphism

ME-Q38. 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. Re-examine 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`.

[8.1] Differentiate among checked exceptions, RuntimeExceptions, and Errors

ME-Q39. 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 will print `caught` if `question()` throws exception `NullPointerException`.

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 class `StackOverflow` can't handle exceptions of class `NullPointerException` because `NullPointerException` is not a superclass of `StackOverflowError`.

[6.7] Apply encapsulation principles to a class

ME-Q40. 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 is no change in the method signature of 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 are available only via a `public` method: `getName`.

[6.2] Apply the static keyword to methods and fields

ME-Q41. 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.

[6.8] Determine the effect upon object references and primitive values when they are passed into methods that change the values

ME-Q42. 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 = "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.memory = "2GB";
    }
}
```

- 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 is 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 is 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`.

[2.1] Declare and initialize variables

ME-Q43. 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 is 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 preceding 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`.

[2.7] Create and manipulate strings

ME-Q44. 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")).c
     oncat("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 `RuntimeException`.

[2.4] Explain an object's life cycle

ME-Q45. 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 to claim unused space 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 are correct. But because nothing changes on line 2, at least two objects are still eligible for garbage collection.

[3.3] Test equality between strings and other objects using == and equals()

ME-Q46. 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`.

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 method `equals` in the base class compares the object references and returns `true` if both the 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
```

[2.6] Manipulate data using the StringBuilder class and its methods

ME-Q47. 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 chars.**

- ☐ **b** `StringBuilder sb1 = new StringBuilder(5*10)` will create a `StringBuilder` object with a value `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 to a `StringBuilder` object.

`new StringBuilder(50)` creates a `StringBuilder` object with no characters, but with an initial capacity to store 50 chars.

[4.1] Declare, instantiate, initialize, and use a one-dimensional array

ME-Q48. 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 round brackets, `()`. 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.

[4.3] Declare and use an ArrayList

ME-Q49. What is the output of the following code? (Select 1 option.)

```
import java.util.*;
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.

[3.4] Create if and if-else constructs

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

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

- 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.**

Answer: e

Explanation: This question tries to trick you on two points. First, there is 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`.

[5.1] Create and use while loops

ME-Q51. 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.
- [x] **b** **The code doesn't print a value.**
- [ ] **c** The code prints `0`.
- [x] **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 a 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.

Option (d) is correct because when the unary increment operator (`++`) is not part of an expression, postfix and prefix notation behave in exactly the same manner.

[5.2] Create and use for loops including the enhanced for loop

ME-Q52. Given the following code, which option, if used to replace `/* INSERT CODE HERE */`, will make the code print the name of the phone with the position at which it's stored in the array `phone`? (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.

[8.5] Recognize common exception classes and categories

ME-Q53. Which of the following classes represent runtime exceptions in Java (select 4 options):

- [x] **a `RuntimeException`**
- [ ] b `CheckedException`
- [x] **c `NullPointerException`**
- [x] **d `ArrayIndexOutOfBoundsException`**
- [ ] e `CompilationException`
- [ ] f `Throwable`
- [ ] g `StackOverflowException`
- [ ] h `MemoryOutOfBoundsException`
- [x] **i `IllegalArgumentException`**
- [ ] j `NumberException`

Answer: a, c, d, i

Explanation: Options (b), (e), (g), (h), and (j) are incorrect because there are no exception classes by these names in the Java API.

Option (f) is incorrect because `Throwable` is the base class of all the exceptions—checked and runtime—and errors in Java. `RuntimeExceptions` is a subset of this class.

[3.3] Test equality between strings and other objects using == and equals()

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

```
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");
        System.out.print(b1.equals(b2) +":");
        System.out.print(b1 == b2);
    }
}
```

- **a** **true:false**
- **b** true:true
- **c** false:true
- **d** false:false

Answer: a

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, `equals` is inherited by all classes. The default implementation of `equals` in the base class compares the object references and returns `true` if both the 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 the variables `b1` and `b2` is the same, `b1.equals(b2)` returns `true`.

[5.2] Create and use for loops including the enhanced for loop

ME-Q55. 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, use of operators, and 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.

[6.3] Create an overloaded method

ME-Q56. 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 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.

[5.5] Use break and continue

ME-Q57. 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 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`.

[3.1] Use Java operators

ME-Q58. 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;
        if (!b++ > 100 && a++ == 10) {
            System.out.println(a+b);
        }
    }
}
```

- **a** 100
- **b** 101
- **c** 102
- **d** **Code fails to compile.**
- **e** No output is produced.

Answer: d

Explanation: Although it may seem that the negation unary 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) && a++ == 10) {
```

[7.1] Implement inheritance

ME-Q59. Given the following definitions of the interfaces `Movable` and `Jumpable`, the task is to declare a class `Person` that inherits both of these interfaces. Which of the following code snippets will accomplish this task? (Select 2 options.)

```
interface Movable {}
interface Jumpable {}
```

- ☑ **a**
```
interface Movable {}
interface Jumpable {}
class Person implements Movable, Jumpable {}
```
- ☐ **b**
```
interface Movable {}
interface Jumpable {}
class Person extends Movable, Jumpable {}
```
- ☐ **c**
```
interface Movable {}
interface Jumpable {}
class Person implements Movable extends Jumpable {}
```

☐ **d**
```
interface Movable {}
interface Jumpable implements Movable {}
class Person implements Jumpable {}
```

☑ **e**
```
interface Movable {}
interface Jumpable extends Movable {}
class Person implements Jumpable {}
```

Answer: a, e

Explanation: Option (b) is incorrect because the right keyword for a `class` to inherit interfaces is `implements` and not `extends`.

Option (c) is incorrect because a class can't use the keyword `extends` to inherit interfaces.

Option (d) is incorrect because an interface should use the keyword `extends` to inherit another interface.

[6.7] Apply encapsulation principles to a class

ME-Q60. Choose the options 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. It should be accessible via non-`private` getter and setter methods.

[7.2] Develop code that demonstrates the use of polymorphism

ME-Q61. 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 `smartPhone`, 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.

[5.4] Compare loop constructs

ME-Q62. Given the following requirements, choose the best looping construct to implement them (choose 1 option):

Step 1: Meet the director of Oracle.
Step 2: Schedule another meeting with the director.
Step 3: Repeat 1 and 2, as long as more meetings are required.

- a `for` loop
- b enhanced `for` loop
- c **`do-while` loop**
- d `while` loop

Answer: c

Explanation: The question asks you to choose the best looping construct. The condition requires meeting the director at least once, without checking any other condition. This requirement is best implemented using a `do-while` loop.

A regular `for` loop and `while` loop will check the condition of whether a meeting is required before the first meeting. This condition doesn't match with the requirements. An enhanced `for` loop is normally used to iterate through elements of a collection. We don't have any collection to iterate through here.

[7.3] Differentiate between the type of a reference and the type of an object

ME-Q63. 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`.

[2.1] Declare and initialize variables

ME-Q64. 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: In this piece of code, the local variable will always be initialized. It's initialized using the `if-else` construct. One of these constructs (`if` or `else`) is sure to execute, initializing the local variable `color` with a value. Hence, there are no issues with the initialization of the variable `color`. The code will execute successfully, printing `blue`.

[2.5] Call methods on objects

ME-Q65. 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 an object of the class in which it's defined and returns the created object. 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 class `ClassRoom`. All calls to the methods `getPageCount` and `setPageCount` in the example operate on unrelated objects.

[5.3] Create and use do-while loops

ME-Q66. 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.

[4.2] Declare, instantiate, initialize, and use a multidimensional array

ME-Q67. 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];`

- [x] **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 multidimensional arrays correctly.

Option (e) is incorrect because the size in the first pair of square brackets is missing.

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

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

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

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

[6.8] Determine the effect upon object references and primitive values when they are passed into methods that change the values

ME-Q68. 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 is 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 is 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 is 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`.

[7.4] Determine when casting is necessary

ME-Q69. 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 interface `Roamable` and class `Phone` are unrelated, a reference variable of type `Roamable` can't refer to an object of 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 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 is executed.

[1.3] Create executable Java applications with a main method

ME-Q70. Which of the following statements is incorrect about *the* `main` method used to start a Java application? (Select 2 options.)

- a A class can't define multiple `main` methods.
- b More than one class in an application can define the `main` method.
- c **The main method may accept a `String`, a `String` array, or varargs (`String... arg`) as a method argument.**
- d **The main method shouldn't define an object of the class in which the main method itself is defined.**

Answer: c, d

Explanation: Option (a) is a true statement. Even though you can define multiple methods with the name `main` in a class, you can define only one `main` method to start a Java application. This question specifically asks you about the `main` method used to start a Java application.

Option (b) is a true statement. Multiple classes in an application may define the `main` method.

Option (c) is a false statement. The `main` method must accept either a `String` array or varargs (`String... arg`) as a method argument. It can't accept a `String` object.

Option (d) is a false statement. There are no limitations on the types of the objects that this method can create. This includes the object of the class in which the `main` method is defined.

[7.5] Use super and this to access objects and constructors

ME Q71. What is the output of the following code? (Select 1 option.)

```
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: d

Explanation: `new PostIt()` creates an object of the class `PostIt` by calling its compiler-provided no-argument constructor. The no-argument constructor of 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 the control to the no-argument constructor. The no-argument constructor then prints `Paper:0`.

[2.6] Manipulate data using the StringBuilder class and its methods

ME-Q72. 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 correctly if 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.

[7.6] Use abstract classes and interfaces

ME-Q73. When considered individually, which of the options is correct for the following code? (Select 1 option.)

```
interface Jumpable { void jump(); }
class Chair implements Jumpable {
    public void jump() {
        System.out.println("Chair cannot jump");
    }
}
```

- a The class `Chair` can't implement the interface `Jumpable` because a `Chair` can't define a method `jump`.
- b **If the name of the interface is changed to `Movable` and definition of class `Chair` is updated to `class Chair implements Movable`, class `Chair` will compile successfully.**

- **c** If the definition of the method `jump` is removed from the definition of the class `Chair`, it will compile successfully.
- **d** If the name of the method `jump` is changed to `run` in the interface `Jumpable`, the class `Chair` will compile successfully.

Answer: b

Explanation: Option (a) is incorrect. The name of the interface or class doesn't matter when it comes to whether a class can implement an interface. If a class implements an interface, it should implement all of the methods defined by the interface.

Option (c) is incorrect. If the definition of the method `jump` is removed from the class `Chair`, it will no longer compile. Because `Chair` implements the interface `Jumpable`, it should implement all of the methods defined in the interface `Jumpable`.

Option (d) is incorrect. If the name of the method `jump` is changed to `run` in the interface `Jumpable`, the class `Chair` should implement this method to compile successfully.

[7.4] Determine when casting is necessary

ME-Q74. 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.

[6.4] Differentiate between default and user-defined constructors

ME-Q75. 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.
- [x] **b** **The class `Employee` fails to compile.**
- [x] **c** **The default constructor can call only no-argument constructor of a base class.**
- [ ] **d** Code that creates an object of class `Employee` in class `Test` didn't pass a `String` value to the constructor of class `Employee`.

Answer: b, c

Explanation: The class `Employee` doesn't compile, so 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.

[8.1] Differentiate among checked exceptions, RuntimeExceptions, and Errors

ME-Q76. Select the correct statements. (Choose 4 options.)

- [x] **a** **Checked exceptions are subclasses of `java.lang.Throwable`.**
- [x] **b** **Runtime exceptions are subclasses of `java.lang.Exception`.**
- [x] **c** **Errors are subclasses of `java.lang.Throwable`.**
- [ ] **d** `java.lang.Throwable` is a subclass of `java.lang.Exception`.
- [ ] **e** `java.lang.Exception` is a subclass of `java.lang.Error`.

- [x] **f Errors aren't subclasses of `java.lang.Exception`.**
- [ ] **g** `java.lang.Throwable` is a subclass of `java.lang.Error`.
- [ ] **h** Checked exceptions are subclasses of `java.lang.CheckedException`.

Answer: a, b, c, f

Explanation: Option (d) is incorrect because `java.lang.Exception` is a subclass of `java.lang.Throwable`.

Option (e) is incorrect because the class `java.lang.Exception` isn't a subclass of `java.lang.Error`. Both of these classes subclass `java.lang.Throwable`.

Option (g) is incorrect because `java.lang.Error` is a subclass of `java.lang.Throwable`.

Option (h) is incorrect because the Java API doesn't define any class `CheckedException` in package `java.lang`.

[6.5] Create and overload constructors

ME-Q77. 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.
- [x] **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.
- [x] **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`.

[7.5] Use super and this to access objects and constructors

ME-Q78. 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.

[8.4] Invoke a method that throws an exception

ME-Q79. Given that the method `write` has been defined as follows,

```
class NoInkException extends Exception {}
class Pen{
    void write(String val) throws NoInkException {
        //.. some code
    }
    void article() {
        //INSERT CODE HERE
    }
}
```

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

- [x] a
```
try {
    new Pen().write("story");
}
catch (NoInkException e) {}
```
- [ ] b
```
try {
    new Pen().write("story");
}
finally {}
```
- [x] c
```
try {
    write("story");
}
catch (Exception e) {}
```

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

Answer: a, c

Explanation: 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 is not 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`.

[1.1] Define the scope of variables

ME-Q80. 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 **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 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 both 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 is no way to do so, the code fails to compile.

[2.7] Create and manipulate strings

ME-Q81. 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`.

[3.5] Use a switch statement

ME-Q82. 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]`, and `shapes[2]`).

[5.3] Create and use do-while loops

ME-Q83. Which of the following options include the ideal conditions for choosing to use a `do-while` loop over a `while` loop? (Select 2 options.)

- [x] a **Repeatedly display a menu to a user and accept input until the user chooses to exit the application.**
- [ ] b Repeatedly allow a student to sit in the exam only if she carries her identity card.
- [x] c **Repeatedly serve food to a person until he wants no more.**
- [ ] d Repeatedly allow each passenger to board an airplane if the passengers have their boarding passes.

Answer: a, c

Explanation: Option (a) is correct. A menu should be displayed to a user at least once so that she can select whether she wants to exit an application.

Option (b) is incorrect. Because the condition of possession of an identity card should be checked *before* a student is allowed to sit an exam, this condition is best implemented using a `while` loop.

Option (c) is correct. Because a person should be served the food at least once *before* asking whether he needs more, this condition is best implemented using a `do-while` loop.

Option (d) is incorrect. Because none of the passengers would be allowed to board without a boarding pass, this condition is best implemented using a `while` loop.

[8.4] Invoke a method that throws an exception

ME-Q84. 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
    }
}
```

- [x] **a**
```
catch (NullPointerException e) {}
catch (ClassCastException e) {}
catch (Exception e) {}
catch (Throwable t) {}
```

- [x] **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) {}
```

- [x] **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 `ClassCastException`. Hence, the exception handler for the class `Exception` can't be placed before the exception handlers of either the class `ClassCastException` or `NullPointerException`.

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

[5.1] Create and use while loops

ME-Q85. What is the output of the following code? (Select 1 option.)

```
class Camera {
    public static void main(String args[]) {
        String settings;
        while (false) {
            settings = "Adjust settings manually";
        }
        System.out.println("Camera:" + settings);
    }
}
```

- **a** The code prints `Camera:null`.
- **b** The code prints `Camera:Adjust settings manually`.
- **c** The code will print `Camera:`.
- **d** **The code will fail to compile.**

Answer: d

Explanation: The key to answering this question is to remember that if the compiler finds unreachable code at compilation time, the code won't compile. Use of the literal value `false` in the `while` loop will ensure that the code in the loop's body won't execute, and this can be determined by the compiler at compilation time.

[6.2] Apply the static keyword to methods and fields

ME-Q86. 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 `200+200`, that is, `400`.

[4.2] Declare, instantiate, initialize, and use a multidimensional array

ME-Q87. What is the output of the following code? (Select 1 option.)

```
String ejgStr[] = new String[][]{{null},new String[]{"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]);
```

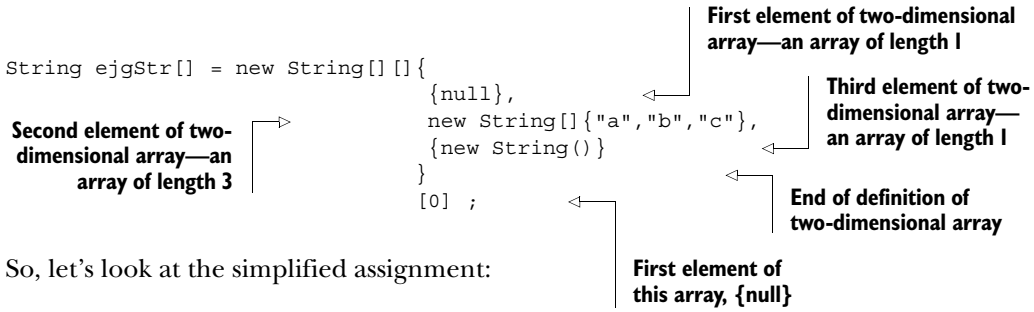
- **a** null
  NullPointerException
- **b** **null**
  null
  NullPointerException
- **c** NullPointerException
- **d** null
  null
  null

Answer: b

Explanation: The trickiest assignment in this code is the assignment of 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] ;
```

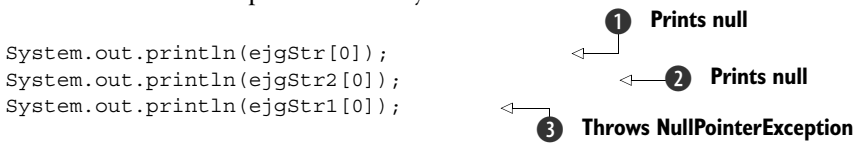
The preceding code assigns the first element of a two-dimensional `String` array to the variable `ejgStr`. The following indented code will make the previous statement easier to understand:



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:



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 `NullPointerException`.

[4.3] Declare and use an ArrayList

ME-Q88. Examine the following code and select the correct statement (choose 1 option):

```
import java.util.*;
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
    }
}
```

- 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 **None of the above.**

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. 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.

[3.4] Create if and if-else constructs

ME-Q89. 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`, and the code will print `3`.

[5.4] Compare loop constructs

ME-Q90. Select the incorrect statement (choose 1 option):

- a An enhanced `for` loop can be used to iterate through the elements of an array and `ArrayList`.
- b **The loop counter of an enhanced `for` loop can be used to modify the current element of the array being iterated over.**
- c `do-while` and `while` loops can be used to iterate through the elements of an array and `ArrayList`.
- d The loop counter of a regular `for` loop can be used to modify the current element of an `ArrayList` being iterated over.

Answer: b

Explanation: Note that this question asks you to select the incorrect statements.

All the looping constructs—a regular `for` loop, an enhanced `for` loop, and `do-while` and `while` loops—can be used to iterate through elements of an array and `ArrayList`. But the notion of a loop counter is available only for the `for` loop. A loop counter is a variable that's defined and accessible with the definition of the `for` loop. In the following quick example, the variable `ctr` is referred to as a loop counter:

```
for (int ctr=0; ctr< 10; ++ctr)
```