Q1

```
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

Q2

```
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

Q3

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"}};`

Q4

```
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

Q5

```
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
        list.add(LocalDate.now().plus(1));             //LINE5
    }
}
```

- a The code on line 1 won't compile.
- b The code on line 2 won't compile.
- c The code on line 3 won't compile.
- d The code on line 4 won't compile.
- e The code on line 5 won't compile.
- f None of the above.
- g All the options from (a) through (e).

Q6

```
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.

Q7

IF we replace insert code which will compile correctly

```
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 {}
```

Q 8

```
class EMyMethods {
    static String name = "m1";
    void riverRafting() {
        String name = "m2";
        if (8 > 2) {
            String name = "m3";
            System.out.println(name);
        }
    }
    public static void main(String[] args) {
        EMyMethods m1 = new EMyMethods();
        m1.riverRafting();
    }
}
```

- a m1
- b m2
- c m3
- d The code fails to compile.

Q9

```
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.

Q10

```
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;`

Q11

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

- a The code will print `LionX`.
- b The code will print `Lion`.
- c The code will print `Lion` if line 5 is changed to the following:

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

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

```
StringBuilder ejg = null;
```

Q12

```
Byte b1 = (byte)100;                        // 1
Integer i1 = (int)200;                      // 2
Long l1 = (long)300;                        // 3
Float f1 = (float)b1 + (
     0int)l1;              // 4
String s1 = 300;                            // 5
if (s1 == (b1 + i1))                        // 6
    s1 = (String)500;                       // 7
else                                        // 8
    f1 = (int)100;                          // 9
System.out.println(s1 + ":" + f1);          // 10
```

what is the output? Select 1 option.

- a Code fails compilation at line numbers 1, 3, 4, 7.
- b Code fails compilation at line numbers 6, 7.
- c Code fails compilation at line numbers 7, 9.
- d Code fails compilation at line numbers 4, 5, 6, 7, 9.
- e No compilation error—outputs `500:300`.
- f No compilation error—outputs `300:100`.
- g Runtime exception.

Q13

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

- a The class will print `true`.
- b The class will print `false`.
- c The class will print `true` if the `if` condition is changed to `boolean == true`.
- d The class will print `false` if the `if` condition is changed to `boolean != true`.
- e The class won't compile.
- f Runtime exception.

Q 14

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];`

Q15

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

Q16

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

- a `true:false`
- b `true:true`
- c `false:true`
- d `false:false`
- e Compilation error—there is no `equals` method in the class `Book`.
- f Runtime exception.

Q17

```
class MyExam {
    void question() {
        try {
            question();
        } catch (StackOverflowError e) {
            System.out.println("caught");
        }
    }
    public static void main(String args[]) {
        new MyExam().question();
    }
}
```

- a The code will print `caught`.
- b The code won't print `caught`.
- c The code would print `caught` if `StackOverflowError` were a runtime exception.
- d The code would print `caught` if `StackOverflowError` were a checked exception.
- e The code would print `caught` if `question()` throws the exception `NullPointer-Exception`.

Q18

```
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

Q19

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

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

- a `Home() {}`
- b `Float Home() {}`
- c `protected Home(int rooms) {}`
- d `final Home() {}`
- e `private Home(long name) {}`
- f `float Home(int rooms, String name) {}`
- g `static Home() {}`

Q20

```
long result;
```

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

- a `Short myMethod1(String str1, int str2, String str3)`
- b `Int myMethod2(String val1, int val2, String val3)`
- c `Byte myMethod3(String str1, str2, int a)`
- d `Float myMethod4(String val1, val2, int val3)`
- e `Long myMethod5(int str2, String str3, String str1)`
- f `Long myMethod6(String... val1, int val2)`
- g `Short myMethod7(int val1, String... val2)`

Q21

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};`