What would the following code display?

```
int[] a = {1,2,3,4,5,6};
int i = a.length - 1;
while(i>=0){
    System.out.print(a[i]);
    i--;
}
```

a. 123456

b. An exception could be thrown at runtime.

c. 654321

d. nothing

e. 65432

f. 12345

Pretty obvious once you have understood that it's looping in decreasing order and when you have the boundaries right.

A and E are classes, B and D are interfaces
Which would be correct?

a. `interface F implements B,D { }`
b. `interface F implements D { }`
c. `interface F extends D { }`
d. `interface F extends E { }`
e. `interface F extends B,D { }`

"To implement" means to provide the code for a method specified in an interface. An interface can only extend, and extend **one** other interface (no multiple inheritance in Java)

What is inheritance?

a. It is the process where one object acquires the properties of another.
b. inheritance is the ability of an object to take on many forms.
c. inheritance is a technique to define different methods of same type.
d. None of the above.

b refers to "polymorphism" and c (although it says nothing about method names) *might* refer to overloading.

What is polymorphism?

a. Polymorphism is a technique to define different objects of same type.

b. Polymorphism is the ability of an object to take on many forms.

c. Polymorphism is a technique to define different methods of same type.

d. None of the above

Poly = several, morph = shape (in Greek)

What are instance variables?

a. Instance variables are static variables within a class but outside any method.
b. Instance variables are variables defined inside methods, constructors or blocks.
c. Instance variables are variables within a class but outside any method.
d. None of the above.

*static* variables may be class variables (outside a method) or variables that you only see within a method but retain their values across calls (they aren't on the stack); b refers to *local* variables, allocated on the stack.

When people refer to "a member of a class", they mean:

a. An attribute
b. A method
c. Attribute or method
d. A sub-class

"member" is a generic term for whatever belongs to a larger group (group member, family member ...). Anything inside a class is a "member".

What would display the following statements?

```
int[] nums = {1,2,3,4,5,6};
System.out.println((nums[1] + nums[3]));
```

a. 6

b. 2+4

c. 1+3

d. 4

Hope you would have gotten this one right ...

If classes Student, Staff and Faculty extend class Person, which one makes sense:

a.
```
Faculty[] faculties={new Person(),
        new Staff(), new Student()};
```
b.
```
Staff[] staff={new Person(),
       new Faculty(), new Student()};
```
c.
```
Person[] persons={new Faculty(),
       new Staff(), new Student()};
```

A reference to the parent can be assigned a reference to a child, not the reverse. Every Faculty, Student and Staff is a Person.

# What is the output of the following program?

```
class Recursion {

  int func(int n) {
    int result;
     if (n == 0) {
        result = 3;
     } else {
        result = func(n - 1);
     }
     return result;
  }
}

public class Prog {
  public static void main(String args[]) {
     Recursion obj = new Recursion() ;
     System.out.print(obj.func(5));
  }
}
```

a. 5
b. 18
c. 3
d. Compilation Error
e. Runtime Error

The function returns the value for a smaller n until it hits zero and returns 3, so it can only return 3.

# What is the output of the following program?

```
class Recursion {
   int func(int n) {
      int result = 0;
      if (n < 10) {
        if (n == 8) {
          result = 1;
        }
      } else {
        result = func(n % 10) + func(n / 10);
      }
      return result;
   }
}
public class Prog {
   public static void main(String args[]) {
       Recursion obj = new Recursion() ;
       System.out.print(obj.func(123) + " ");
       System.out.print(obj.func(8328) + " ");
       System.out.println(obj.func(8325));
   }
}
```

a. 0 0 0
b. 0 2 1
c. 0 1 0
d. Compilation Error

The function actually counts how many '8' there are in the number.

Which of the following is a valid annotation definition ?

a. `public @annotation MyAnnotation{ }`
b. `private @interface MyAnnotation{ }`
c. `public @interface MyAnnotation{ }`
d. `public @MyAnnotation{ }`

An annotation is a special interface, and there is no reason to make it private.

Which of the following belongs to meta-annotations?

a. `@Override`
b. `@Retention`
c. `@Deprecated`
d. `@SuppressWarnings()`

A meta-annotation is an annotation about annotation.
The other annotations refer to the annotated code.

Which of the following are valid retention policy types in Java (multiple answers)?

a. `SOURCE`
b. `CLASS`
c. `RUNTIME`
d. `CODE`
e. `TOOLS`

CODE and TOOLS don't exist.

# What will be the output of the following program?

```
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotation {
   int value();
}

class Hello {
   @MyAnnotation(value = 10)
   public void goodEvening() {
      System.out.print("Good Evening");
   }
}
public class HelloAnnotation {
  public static void main(String args[]) throws Exception {
     Hello h = new Hello();
     Method m = h.getClass().getMethod("goodEvening");
     MyAnnotation myAnn = m.getAnnotation(MyAnnotation.class);
     System.out.println("Prints " + myAnn.value());
  }
}
```

a. Prints HelloAnnotation 10
b. Good Evening Prints 10
c. Prints 10
d. Some other output
e. Output cannot be determined

The method goodEvening() isn't called, we only retrieve the annotation.

# What will be the output of the following program?

```
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.CLASS)
@interface MyAnnotation {
    int value();
}
class Hello {
  @MyAnnotation(value = 10)
  public void goodEvening() {
    System.out.print("Good Evening");
  }
}
public class HelloAnnotation {
  public static void main(String args[]) throws Exception {
    Hello h = new Hello();
    Method m = h.getClass().getMethod("goodEvening");
    MyAnnotation myAnn = m.getAnnotation(MyAnnotation.class);
    System.out.println("Prints " + myAnn.value());
  }
}
```

a. Prints HelloAnnotation 10
b. Good Evening Prints 10
c. Prints 10
d. Some other output
e. Output cannot be determined
f. Compilation Error
g. Runtime Exception

As the retention isn't RUNTIME, trying to get the annotation returns a null reference when the program runs.

What is the output of the following program?

```
interface Int{}
class Simple implements Int{}
class SimpleTest {
   public static void main(String[] args) {
     try {
       Class c=Class.forName("Simple");
       System.out.print(c.isInterface() + " ");
       c=Class.forName("Int");
       System.out.println(c.isInterface());
     } catch (Exception e) {
       System.out.println(e);
     }
   }
}
```

An interface kind of extends class (lightweight abstract class)

a. false true
b. true true
c. true false
d. false false

Which of the following can obtain the location of file Reflection.class at runtime?

```
public class Reflection {}
```

a. `Reflection.class.getClassLoader().getResource("Reflection.class")`
b. `Reflection.getClass().getResource("Reflection.class")`
c. `Reflection.getClass().getClassLoader().getResource("Reflection.class")`
d. `Reflection.class.getClassLoader("Reflection.class")`

getClass() applies to an object, .class to a class. File "Reflection.class" is a resource for the JVM.

Consider the following program:

```
class MyClass {
   private float val;
   @Deprecated
   MyClass(int n) {
     val = n;
   }
   MyClass(float f) {
     val = f;
   }
}

public class TestDeprecated {
   public static void main(String[] args) {
     int val = 3;
     MyClass obj = new MyClass(3);
   }
}
```

What of the following is true?

a. It generates one warning (call of the deprecated constructor) when compiling, nothing when running

b. It generates two warnings (definition of the deprecated constructor and call of the deprecated constructor) when compiling, nothing when running

c. No warning when compiling but it generates one warning when running

d. No warning when compiling but it generates two warnings when running

e. No warning when compiling, exception when running

f. One warning when compiling, exception when running

Only developers USING the deprecated function are warned.

What is the main function of JDBC?

a. Create a connection to the database.
b. Send SQL statements to the database.
c. Processing data and query results
d. All of the above

Not too hard.

(multiple choice) which of the following steps are performed when accessing the database through JDBC?

a. Loading the JDBC driver.
b. Setting up the connection.
c. Executing queries or applying changes to the database
d. Close the connection

Not too hard either. Note that although 99.9% of JDBC applications load the driver, it's also possible to import it (but then the .jar must be in your classpath when you compile)

Which package allows Java to access a database?

a. java.sql
b. java.db
c. java.dbms
d. java.jdbc
e. java.lang
f. java.util

b, c, d don't exist. e is the default package. f contains a lot of useful stuff, but not JDBC.

In general, you prefer adding a TableView to a:

a. BorderPane
b. StackPane
c. ScrollPane
d. SplitPane
e. TabPane
f. TitledPane

A TableView is used to display data retrieved from a data source (often a database, but it could as well be the result of streaming an in-memory collection, or data retrieved from the network). You rarely know how many rows you'll display, and the wisest choice is a ScrollPane.

FXCollections: In Java, plural is often used to name classes that contain static methods (Arrays, Collections, ...)

a. Is a package that contains classes for collections that you should use as backend to some JavaFX widgets.

b. Is a class that contains wrapper methods for regular collections so that you can use them as backend to some JavaFX widgets.

c. Is a class that only contains the ObservableList inner class

d. Is an enum that is used for defining the collections that can be used with JavaFx widgets (for instance a TreeSet cannot be used, and doesn't appear in the enum)

"static" in java indicates:

a. Something that you cannot change
b. Something that is attached to the class, not to a particular object instance
c. Something that cannot throw exceptions
d. Something that cannot be overriden (methods) or extended

Something that you cannot change (variable) or cannot be overriden (method) or extended (class) is preceded by ***final***.

The protocol that takes a message from a network to another network is:

a. FTP
b. TCP
c. IP
d. HTTP

IP = Internet Protocol.

*Internet* means to net(work)s what *international* means to nations.

FTP = File Transfer Protocol (more like HTTP, but dedicated to exchanging files)
TCP = Transmission Control Protocol, computer-to-computer (uses IP underneath)
HTTP = Hyper Text Transfer Protocol, needs no introduction.

Which tool isn't a build tool:

a. ant
b. junit
c. make
d. maven

Testing tool.

The time taken by a message to travel between computers is called:

a. bandwidth
b. routing
c. latency
d. delay

Bandwidth refers to how much data can be sent at the same time, not to speed.
Routing is finding a circuit (route) to go to computer to computer; it's about traffic redirection.
Delay is just a general term.

You need to have in your classpath the .jar of the JDBC driver that you are using:

a. Only when you compile the program
b. Only when you run the program
c. Both when you compile and run the program

b is the good answer for 99.9% of the cases, when you load the driver dynamically. Now, you can also import it, but in that case the good answer would be c. And if you are using JavaDB (Derby), as it's part of the standard JDK/JRE you don't need to specify anything special.
A quiz isn't the place for fine-point distinctions. Always pick the most likely.

If you serialize an object that contains references, when you reload it the referenced objects will go at exactly the same place in memory:

a. True
b. False

Memory location is the business of the operating system, not yours. It can put things wherever it likes, as long as it returns references to you.

To make an object serializable:

a. You don't need to do anything - all objects are serializable by default.

b. You only need to say that it implements the Serializable interface.

c. You need to say that it implements the Serializable interface and implement the two methods writeObject() and readObject() defined in the interface.

It's not done by default because it adds overhead (more code) that not everybody needs.