Q>
1. Which ONE of the following statements is TRUE?
A. You cannot extend a concrete class and declare that derived class abstract
B. You cannot extend an abstract class from another abstract class
C. An abstract class must declare at least one abstract method in it
D. You can create an instance of a concrete subclass of an abstract class but cannot create an instance of an abstract class itself.

Answer: D

Q>Choose the correct answer based on the following class definition:

```
      public abstract final class Shape { }
```

A. Compiler error: a class must not be empty
B. Compiler error: illegal combination of modifiers abstract and final
C. Compiler error: an abstract class must declare at least one abstract method
D. No compiler error: this class definition is fine and will compile successfully

Answer: B

Q>Choose the best option based on this program:

```
class Shape {
            public Shape() {
                  System.out.println("Shape constructor");
            }
            public class Color {
                  public Color() {
                        System.out.println("Color constructor");
                  }
            }
}
class TestColor {
      public static void main(String []args) {
            Shape.Color black = new Shape().Color(); // #1
      }
}
```

A. Compiler error: the method Color() is undefined for the type Shape
B. Compiler error: invalid inner class
C. Works fine: Shape constructor, Color constructor
D. Works fine: Color constructor, Shape constructor

Answer: A

Q>

```
class Shape {
 private boolean isDisplayed;
 protected int canvasID;
      public Shape() {
            isDisplayed = false;
            canvasID = 0;
      }
      public class Color {
            public void display() {
                  System.out.println("isDisplayed: "+isDisplayed);
                  System.out.println("canvasID: "+canvasID);
            }
      }
}
class TestColor {
      public static void main(String []args) {
                  Shape.Color black = new Shape().new Color();
                  black.display();
      }
}
```
A. Compiler error: an inner class can only access public members of the outer class
B. Compiler error: an inner class cannot access private members of the outer class
C. Runs and prints this output:
```
       isDisplayed: false
      canvasID: 0
```
D. Compiles fine but crashes with a runtime exception

Answer: C

Q>
Determine the behavior of this program:
```
interface DoNothing {
      default void doNothing() { System.out.println("doNothing"); }//from jdk1.8
inside an interface we can have concrete methods also.
}

@FunctionalInterface
interface DontDoAnything extends DoNothing {
 @Override
 abstract void doNothing();
}

class LambdaTest {
      public static void main(String []args) {
                  DontDoAnything beIdle = () -> System.out.println("be idle");
                  beIdle.doNothing();
      }
}
```
A. This program results in a compiler error for DontDoAnything interface: cannot override default method to be an abstract method
B. This program results in a compiler error: DontDoAnything is not a functional interface
C. This program prints: doNothing
D. This program prints: be idle

Answer: D

Q>
Determine the behavior of this program:
```
interface BaseInterface {
      default void foo() { System.out.println("BaseInterface's foo"); }
}
interface DerivedInterface extends BaseInterface {
      default void foo() { System.out.println("DerivedInterface's foo"); }
}
interface AnotherInterface {
      public static void foo() { System.out.println("AnotherInterface's foo"); }
}
public class MultipleInheritance implements DerivedInterface, AnotherInterface {
      public static void main(String []args) {
                  new MultipleInheritance().foo();
      }
}
```
A. This program will result in a compiler error: Redundant method definition for

function foo
B. This program will result in a compiler error in MultipleInheritance class: Ambiguous call to function foo
C. The program prints: DerivedInterface’s foo
D. The program prints: AnotherInterface's foo

Answer: C

Q>
Determine the behavior of this program:

```
class LambdaFunctionTest {
 @FunctionalInterface
 interface LambdaFunction {
      int apply(int j);
      boolean equals(java.lang.Object arg0);
 }
 public static void main(String []args) {
      LambdaFunction lambdaFunction = i -> i * i; // #1
      System.out.println(lambdaFunction.apply(10));
 }
}
```

A. This program results in a compiler error: interfaces cannot be defined inside classes
B. This program results in a compiler error: @FunctionalInterface used for LambdaFunction that defines two abstract methods
C. This program results in a compiler error in code marked with #1: syntax error
D. This program compiles without errors, and when run, it prints 100 in console

Answer: D