Repaso;

Friday, December 6, 2024 4:11 PM

Which three implementations are valid?

```
interface SampleCloseable {
        public void close() throws java.io.IOException;
}
```

class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }

class Test implements SampleCloseable { public void close() throws Exception { // do something } }

–

class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }

class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }

class Test implements SampleCloseable { public void close() { // do something }}

Respuesta correcta

class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }
class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }
class Test implements SampleCloseable { public void close() { // do something }}

-
-
- Un método sobrescrito puede:
  - Lanzar las mismas excepciones declaradas en la interfaz (o clase base).
  - Lanzar subclases de esas excepciones.
  - Lanzar ninguna excepción.
- No puede lanzar una excepción más genérica o una no relacionada con las excepciones declaradas originalmente.

- **public void close() throws java.io.IOException**
  Válida. Es idéntica a la declaración en la interfaz.
- **public void close() throws Exception**
  Inválida. Exception es más genérica que IOException, lo cual no está permitido.
- **public void close() throws FileNotFoundException**
  Válida. FileNotFoundException es una subclase de IOException.
- **public void close() {}**
  Válida. No lanzar ninguna excepción es siempre permitido.
- **public void close() throws Throwable**
  Inválida. Throwable no es una subclase de IOException.

//////////////////////////////

Which three lines will compile and output "Right on!"?

```
13.   public class Speak {
14.         public static void main(String[] args) {
15.                 Speak speakIT = new Tell();
16.                 Tell tellIt = new Tell();
17.                 speakIT.tellItLikeItIs();
18.                 (Truth) speakIT.tellItLikeItIs();
19.                 ((Truth) speakIT).tellItLikeItIs();
20.                 tellIt.tellItLikeItIs();
21.                 (Truth) tellIt.tellItLikeItIs();
22.                 ((Truth) tellIt).tellItLikeItIs();
23.         }
24.   }

class Tell extends Speak implements Truth {
        @Override
        public void tellItLikeItIs() {
                System.out.println("Right on!");
        }
}

interface Truth {
        public void tellItLikeItIs();
}
```

- Tell extiende Speak e **Implementa** la interfaz Truth, la cual define el método tellItLikeItIs().
- Tell sobrescribe este método, imprimiendo "Right on!".

- **Línea 19:** ((Truth) speakIT).tellItLikeItIs();
  - Aquí se hace un casting explícito a Truth. El objeto real es de tipo Tell, que implementa Truth. **Compila y produce "Right on!".**
- **Línea 20:** tellIt.tellItLikeItIs();
  - tellIt es declarado como Tell, que sobrescribe el método. **Compila y produce "Right on!".**

Línea 22: ((Truth) tellIt).tellItLikeItIs();
  - Similar a la anterior. **Compila y produce "Right on!".**

//

What changes will make this code compile? SELECCIONA 2

```
class X {
        X() { }
        private void one() { }
}
public class Y extends X {
        Y() { }
        private void two() {
        one();
        }
        public static void main(String[] args) {
        new Y().two();
        }
}
```

Adding the public modifier to the declaration of class X.
Adding the protected modifier to the X() constructor.
Changing the private modifier on the declarationof the one() method to protected.

Removing the Y() constructor.

Cambiar el acceso de private a protected permitirá que las clases derivadas, como Y, accedan a one().

Hacer el método two() accesible (de private a public).

//

¿Qué imprime?

```
public class Main {
  public static void main(string[] args) {
     int x = 2;
     for(;x<5;){
       x=x+1;
       System.out.println(x);
     }
  }
}
```

x se inicializa con el valor 2, cada que se cumple la condicion a x se le suma 1

El bucle termina hasta que x sea menor

Por lo que imprimira; 3 4 5

//////////////////////////////////////

```
public class Test {
  public static void main(String[] args) {
    int[][] array = { {0}, {0,1}, {0,2,4}, {0,3,6,9}, {0,4,8,12,16} };
    System.out.println(array{4}{1});
    System.out.println(array{1}{4});
  }
}
```

Array 4, indice 1 = 1 4
Array 1, indice 4 = ArrayIndexOutOfBoundsException (porque el arreflo solo tiene dos indices)

//

Which two possible outputs?

```
public class Main {
        public static void main(String[] args) throws Exception {
                doSomething();
        }
        private static void doSomething() throws Exception {
                System.out.println("Before if clause");
                if (Math.random() > 0.5) { throw new Exception();}
                System.out.println("After if clause");
        }
}
```

- **throw new Exception();** crea y lanza una instancia de java.lang.Exception.
- Exception es una clase de excepciones **verificadas (checked exceptions)**, lo que significa que debe ser declarada en la firma del método o manejada con un bloque try-catch.

Debido a que Math.random() es aleatorio, hay dos posibles flujos de ejecución:

Cuando Math.random() es menor o igual a 0.5:

- No se lanza la excepción.
- Se imprimen ambas frases

Before if clause
After if clause

Cuando Math.random() es mayor a 0.5:
- Se lanza una excepción después de la primera impresión.

Before if clause

//

¿Cuál es la salida?

```
public class Main {
  public static void main(String[] args) {
     int x = 2;
     if(x==2) System.out.println("A");
     else System.out.println("B");
     else System.out.println("C");
  }
}
```

Error de sintaxis, una sentencia if solo puede contener un else

//////////////////////

10. How many times is 2 printed?

```
class Menu {
       public static void main(String[] args) {
              String[] breakfast = {"beans", "egg", "ham", "juice"};
              for (String rs : breakfast) {
                     int dish = 1;
                     while (dish < breakfast.length) {
                     System.out.println(rs + "," + dish);
                     ++dish;
                     }
              }
```

Debido al operador **pre-incremento** (++dish), el valor de dish se incrementa antes de ejecutarse la instrucción System.out.println(rs + "," + dish);. Por esta razón, la impresión empieza con 2 y no con 1.

- **nicialización:**
  - dish se inicializa con 1.
- **Primera iteración del while:**
  - La condición dish < breakfast.length es 1 < 4 (verdadero).
  - **Pre-incremento:** ++dish incrementa dish de 1 a 2 **antes** de imprimir.
  - Se imprime 2.
- **Segunda iteración del while:**
  - dish ahora es 2.
  - La condición dish < breakfast.length es 2 < 4 (verdadero).
  - **Pre-incremento:** ++dish incrementa dish de 2 a 3.
  - Se imprime 3.
- **Tercera iteración del while:**

- dish ahora es 3.
- La condición dish < breakfast.length es 3 < 4 (verdadero).
- **Pre-Incremento:** ++dish incrementa dish de 3 a 4.
- Se imprime 4.

- **Cuarta Iteración del while:**
  - dish ahora es 4.
  - La condición dish < breakfast.length es 4 < 4 (falso), por lo que el bucle termina.

beans,2, beans,3, egg,2, egg,3, ham,2, ham,3, juice,2, juice,3 4 veces se imprime 2

//////////////////////////

```
class Person {
    String name = "No name";
    public Person(String nm) { name = nm; }
}

class Employee extends Person {
    String empID = "0000";
    public Employee(String id) { empID " //18
    }
}

public class EmployeeTest {
    public static void main(String[] args) {
        Employee e = new Employee("4321");
        System.out.println(e.empID);
    }
}
```

- Falta el operador de asignación = para asignar id a empID.
- Falta el punto y coma ; al final de la instrucción.
- **Solución:**
  Corregir la línea para que se vea así:

  Public Employee (String id) { empID = id; }

//

```
class Atom {
       Atom() {System.out.print("atom ");}
}
class Rock extends Atom {
       Rock(String type) {System.out.print(type);}
}
public class Mountain extends Rock {
       Mountain(){
              super("granite ");
              new Rock("granite ");
       }
       public static void main(String[] a) {new Mountain();}
} //atom granite atom granite
```

- **super("granite ")** en el constructor de Mountain:
  - Atom constructor → "atom "
  - Rock constructor → "granite "
- **new Rock("granite ") en el constructor de Mountain:
  - Atom constructor → "atom "
  - Rock constructor → "granite "

//

13. What is the result?

```
import java.text.*;
public class Align {
	public static void main(String[] args) throws ParseException {
		String[] sa = {"111.234", "222.5678"};
		NumberFormat nf = NumberFormat.getInstance();
		nf.setMaximumFractionDigits(3); //NO HACE NADA
		for (String s : sa) {
			System.out.println(nf.parse(s));
		}
	}
}
```

A. 111.234 222.567
B. 111.234 222.568
C. 111.234 222.5678
D. An exception is thrown at runtime

parse convierte una cadena que representa un número en un objeto Number.

- **Arreglo sa:**
  Contiene dos cadenas:

  {"111.234", "222.5678"}
- **NumberFormat nf = NumberFormat.getInstance();**
  Crea una instancia de NumberFormat para formatear números de acuerdo con las configuraciones regionales predeterminadas del sistema.
- **nf.setMaximumFractionDigits(3);**
  Esta línea **no afecta a parse**. El método parse simplemente convierte una cadena en un número sin redondear ni modificar los dígitos fraccionarios. La configuración de setMaximumFractionDigits(3) afecta solo cuando se usa format, no cuando se usa parse.
- **Bucle for:**
  Itera sobre cada elemento del arreglo sa y llama a nf.parse(s), que convierte la cadena a un número (Number).

//////////////////////////////

. What is the result?

```
interface Rideable {
	String getTicket(); //No recuerdo muy bien el nombre del método, pero
era diferente
}
public class Camel implements Rideable {
	int weight = 2;
	String getGait() {
		return mph + ", lope";
	}
	void go(int speed) {
		++speed;
		weight++;
		int walkrate = speed * weight;
		System.out.print(walkrate + getGait());
	}
	public static void main(String[] args) {
		new Camel().go(8);
	}
}
```

//