GUÍA DE PREGUNTAS DE APX 2

1. Which declaration initializes a boolean variable?

- ➢ a) boolean m = null
- ➢ b) Boolean j = (1<5)
- ➢ c) boolean k = 0
- ➢ d) boolean h = 1

Explicación:

La opción **b)** es correcta porque (1 < 5) es una expresión que se evalúa como true, y esa es la manera correcta de inicializar una variable booleana. Las otras opciones intentan asignar valores no válidos para un booleano, como null, 0, o 1.

2. What is the DTO pattern used for?

- ➢ a) To Exchange data between processes
- ➢ b) To implement the data Access layer
- ➢ c) To implement the presentation layer

Explicación:

El patrón DTO se usa para intercambiar datos entre procesos. Su propósito es agrupar datos en un solo objeto para facilitar la transferencia entre diferentes partes de una aplicación.

3. What is the result?

```
Int a = 10; int b = 37; int z = 0; int w = 0;
If (a==b) {z=3; } else if (a>b) {z=6;}
w = 10 * z;
```

- ➢ a) 30
- ➢ b) 0
- ➢ c) 60

Explicación:

El resultado es 0 porque las condiciones en el if y else if no se cumplen, dejando a z en 0. Luego, al calcular w = 10 * z, el valor de w también es 0.

4. Which three options correctly describe the relationship between the classes?

```
class Class1{String v1;}
class Class2{
      Class1 c1;
      String v2;
}
class Class3 {Class2 c1; String v3;}
```

- ➢ A. Class2 has-a v3.
- ➢ B. Class1 has-a v2.
- ➢ C. Class2 has-a v2.
- ➢ D. Class3 has a v1.
- ➢ E. Class2 has-a Class3.
- ➢ F. Class2 has-a Class1.

Explicación:

Las opciones correctas son C, D y F. Class2 tiene un atributo v2, Class3 indirectamente accede a v1 a través de sus relaciones con Class2 y Class1, y Class2 tiene un atributo de tipo Class1.

5. What is the result?

```
try {
    // assume "conn" is a valid Connection object
    // assume a valid Statement object is created
    // assume rollback invocations will be valid
    // use SQL to add 10 to a checking account
    Savepoint s1 = conn.setSavePoint();
    // use SQL to add 100 to the same checking account
    Savepoint s2 = conn.setSavePoint();
    // use SQL to add 1000 to the same checking account
    // insert valid rollback method invocation here
} catch (Exception e) {}
```

A) If conn.rollback(s1) is inserted, account will be incremented by 10.
B) If conn.rollback(s1) is inserted, account will be incremented by 1010.
C) If conn.rollback(s2) is inserted, account will be incremented by 100.
D) If conn.rollback(s2) is inserted, account will be incremented by 110.
E) If conn.rollback(s2) is inserted, account will be incremented by 1110.

Explicación:

Si se invoca rollback(s1), la cuenta se revierte al estado después de sumar 10, deshaciendo los 100 y 1000. Luego, si se agrega 1000, la cuenta se incrementará en 1010 en total.

6. Which two statements are true an Abstract ?

A) An abstract class can implement an interface.
B) An abstract class can be extended by an interface.
C) An interface CANNOT be extended by another interface.
D) An interface can be extended by an abstract class.
E) An abstract class can be extended by a concrete class.
F) An abstract class CANNOT be extended by an abstract class.

Explicación:

Una clase abstracta puede implementar una interfaz porque puede proporcionar implementaciones para los métodos de la interfaz y dejar otros métodos como abstractos. También puede ser extendida por una clase concreta, que debe completar los métodos abstractos de la clase abstracta para ser una clase funcional.

7. Which two are 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");
    }
}
```

A) Before if clause Exception in thread “main” java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
B) Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15) After if clause

C) Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15)
D) Before if clause After if clause

Expllicación:
Si el valor de Math.random() es mayor que 0.5, se lanza una excepción y solo se imprime "Before if clause" seguido del stack trace. Si el valor es menor o igual a 0.5, no se lanza excepción y se imprimen ambas líneas: "Before if clause" y "After if clause".

8. What value should replace kk in line 18 to cause jj = 5 to be output?

```
public class MyFive {
    public static void main(String[] args) {
        //short kk = ?;
        short ii;
        short jj = 0;
        for (ii = kk; ii > 6; ii-=1) {
            jj++;
        }
        System.out.println("jj = " + jj);
    }
}
```

A) -1
B) 1
C) 5
D) 8
E) 11

Explicación:

Para que jj sea 5, kk debe ser 11. Con kk en 11, el bucle se ejecuta 5 veces (desde 11 hasta 7) y en cada iteración jj se incrementa en 1. Así, después de 5 iteraciones, jj será 5.

9. What will make this code compile and run?

```
//1  public class Simple {
//2
//3   public float price;
//4   public static void main(String[] args) {
//5
//6   Simple price = new Simple();
```

```
//7       price = 4;
//8   }
//9}
```

A. Change line 3 to the following: public int price;
B. Change line 7 to the following: int price = new Simple();
C. Change line 7 to the following: float price = new Simple();
D. Change line 7 to the following: price = 4f;
E. Change line 7 to the following: price.price = 4;

Explicación:

Para que el código compile y funcione, se debe de usar price.price = 4 Esto es porque price es una instancia de Simple, y price.price accede al campo price de esa instancia, permitiendo asignarle el valor 4.

10. In the Java collections framework a Set Is:

A. A collection that cannot contain duplicate elements.
B. An ordered collection that can contain duplicate elements.
C. An object that maps value key sets and cannot contain values Duplicates.

Explicación:

Un Set en Java es una colección que no permite elementos duplicados. No garantiza un orden específico de los elementos y no maneja pares clave-valor como un Map.

11. What should statement1, statement2, and statement3, be respectively, in order to produce the result?

Shape: constructor
Shape: foo
Square: foo

```
public class SuperTest {
    public static void main(String[] args) {
        // statement1
        // statement1
        // statement1
    }
}
public class Shape {
    public Shape() {
```

```
        System.out.println("Shape: constructor");
    }

    public void foo() {
        System.out.println("Shape: foo");
    }
}

public class Square extends Shape {
    public Square() {
        super();
    }

    public Square(String label) {
        System.out.println("Square: constructor");
    }

    public void foo() {
        super.foo();
    }

    public void foo(String label) {
        System.out.println("Square: foo");
    }
}
```

A. Square square = new Square("bar"); square.foo("bar"); square.foo();
B. Square square = new Square("bar"); square.foo("bar"); square.foo("bar");
C. Square square = new Square(); square.foo(); square.foo(bar);
D. Square square = new Square(); square.foo(); square.foo("bar");
E. Square square = new Square(); square.foo(); square.foo();

Explicación:

Square square = new Square();: Esto crea una instancia de Square usando el constructor predeterminado de Square, que llama al constructor de Shape y luego el constructor de Square, lo que imprime Shape: constructor.

square.foo();: Llama al método foo() de la clase Square. En Square, foo() llama al método foo() de la clase Shape, que imprime "Shape: foo".

square.foo("bar"); Esta llamada invoca el método foo(String label) de la clase Square, el cual imprime "Square: foo".

12. What is the result?

```
public class SampleClass {
    public static void main(String[] args) {
        AnotherSampleClass asc = new AnotherSampleClass();
        SampleClass sc = new SampleClass();
        sc = asc;
        System.out.println("sc: " + sc.getClass());
        System.out.println("asc: " + asc.getClass());
    }
}
class AnotherSampleClass extends SampleClass {}
```

A. sc: class.Object asc: class.AnotherSampleClass
B. sc: class.SampleClass asc: class.AnotherSampleClass
C. sc: class.AnotherSampleClass asc: class.SampleClass
D. sc: class.AnotherSampleClass asc: class.AnotherSampleClass

Explicación:

Esto ocurre porque después de asignar asc a sc, ambas variables referencian el mismo objeto de AnotherSampleClass. Por eso, al imprimir el tipo de clase de sc y asc, ambos muestran AnotherSampleClass.

13. What is true about the class Wow?

```
public abstract class Wow {
    private int wow;
    public Wow(int wow) { this.wow = wow; }
    public void wow() { }
    private void wowza() { }
}
```

A. It compiles without error.
B. It does not compile because an abstract class cannot have private methods.
C. It does not compile because an abstract class cannot have instance variables.
D. It does not compile because an abstract class must have at least one abstract method.
E. It does not compile because an abstract class must have a constructor with no arguments.

Explicación:

Las clases abstractas pueden tener métodos privados, variables de instancia, y no necesitan incluir métodos abstractos o tener un constructor sin argumentos.

14. The Singleton pattern allows:

A. Have a single instance of a class and this instance cannot be used by other classes.
B. Having a single instance of a class, while allowing all classes have access to that instance.
C. Having a single instance of a class that can only be accessed by the first methods that calls it.

Explicación:

El patrón Singleton garantiza que solo se cree una instancia de una clase y que esta instancia sea accesible desde cualquier parte del código, permitiendo que todas las clases la usen.

15. How many times is 2 printed?

```
public static void main(String[] args) {
    String[] table = {"aa", "bb", "cc"};
    int ii = 0;
    for (String ss : table) {
        while (ii < table.length) {
            System.out.println(ii); ii++;
            break;
        }
    }
}
```

A. Zero.
B. Once.
C. Twice.
D. Thrice.
E. It is not printed because compilation fails.

Explicación:

1. Durante la primera iteración del bucle for, ii es 0 y se imprime 0. Luego, ii se incrementa a 1 y el bucle while termina debido al break.
2. En la segunda iteración del bucle for, ii es 1 y se imprime 1. Después, ii se incrementa a 2 y el bucle while termina.
3. En la tercera iteración del bucle for, ii es 2, así que se imprime 2. Luego, ii se incrementa a 3 y el bucle while termina.

Las impresiones son 0, 1 y 2, pero el valor 2 se imprime solo una vez durante la tercera iteración del bucle for.SS

16. What is the result?

```
public static void main(String[] args) {
    int [][] array2D = { {0, 1, 2}, {3, 4, 5, 6} };
    System.out.print(array2D[0].length + "");
    System.out.print(array2D[1].getClass().isArray() + "" );
    System.out.println(array2D[0][1]);
}
```

A. 3false1
B. 2true3
C. 2false3
D. 3true1
E. 3false3
F. 2true1
G. 2false1

Explicación:

El código imprime tres valores consecutivos. Primero, la longitud del primer array que es 3. Luego, verifica si el segundo array es efectivamente un array, lo cual es cierto, por lo que imprime "true". Finalmente, accede al segundo elemento del primer array, que es 1. Así, el resultado completo impreso en la consola es "3true1".

17. In Java the difference between throws and throw is:

A. Throws throws an exception and throw indicates the type of exception that the method.
B. Throws is used in methods and throw in constructors.
C. Throws indicates the type of exception that the method does not handle and throw an exception.

Explicación:

"throws" se coloca en la firma del método y se usa para declarar posibles excepciones pero no son manejadas dentro del mismo, obligando a quien llame al método a manejarlas. Por otro lado, "throw" se usa para lanzar una excepción específica en un momento dado.

18. What is the result?

```
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.printiln(e.empID);
      }
}
```

A. 4321.
B. 0000.
C. An exception is thrown at runtime.
D. Compilation fails because of an error in line 18.

Explicación:

El código no compila porque la clase Employee no llama al constructor de Person, que requiere un argumento de tipo String. Las subclases deben llamar explícitamente al constructor de la superclase con super(), pasando el argumento necesario.

19. Which is true?

```
class Building {}
      public class Barn extends Building{
            public static void main(String[] args){
                  Building build1 = new Building();
                  Barn barn1 = new Barn();
                  Barn barn2 = (Barn) build1; //10
                  Object obj1 = (Object) build1; //11
                  String str1 = (String) build1; //12
                  Building build2 = (Building) barn1; //13
      }
}
```

A. If line 10 is removed, the compilation succeeds.

B. If line 11 is removed, the compilation succeeds.
C. If line 12 is removed, the compilation succeeds.
D. If line 13 is removed, the compilation succeeds.
E. More than one line must be removed for compilation to succeed.

Explicación:

La línea 10 (Barn barn2 = (Barn) build1;) intenta convertir de manera directa un objeto Building a Barn, lo cual no es válido. La línea 12 (String str1 = (String) build1;) intenta convertir un objeto Building a String, lo cual también es inválido.

20. What is the result?

```
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();}
}
```

A. Compilation fails.
B. Atom granite.
C. Granite granite.
D. Atom granite granite.
E. An exception is thrown at runtime.
F. Atom granite atom granite.

Explicación:

La salida es Atom granite atom granite porque el constructor de Mountain llama al constructor de Rock usando super(), lo cual a su vez llama al constructor de Atom (imprimiendo "atom "), y luego imprime "granite" desde el constructor de Rock. Este proceso se repite cuando se crea un nuevo objeto Rock dentro del constructor de Mountain, imprimiendo "atom granite" de nuevo.

21. Which statement is true?

```
class ClassA {
	public int numberOfInstances;
	protected ClassA(int numberOfInstances) {
		this.numberOfInstances = numberOfInstances;
	}
}
public class ExtendedA extends ClassA {
	private ExtendedA(int numberOfInstances) {
		super(numberOfInstances);
	}
	public static void main(String[] args) {
		ExtendedA ext = new ExtendedA(420);
		System.out.print(ext.numberOfInstances);
	}
}
```

A. 420 is the output.
B. An exception is thrown at runtime.
C. All constructors must be declared public.
D. Constructors CANNOT use the private modifier.
E. Constructors CANNOT use the protected modifier.

Explicación:

Este código imprime "420" porque en la superclase ClassA hay una variable llamada numberOfInstances que se establece cuando se crea una nueva instancia de la clase. La clase ExtendedA hereda de esta superclase y su constructor también recibe un número, que pasa al constructor de la clase superclase para inicializar numberOfInstances. En el método principal (main), se crea una nueva instancia de ExtendedA con el valor 420. Este valor se asigna a numberOfInstances y luego se imprime.

22. What is the result?

```
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]);
```

```
    }
}
```

A. 4 Null.

B. Null 4.

C. An IllegalArgumentException is thrown at run time.

D. 4 An ArrayIndexOutOfBoundsException is thrown at run time.

Explicación:

En este código se define un arreglo bidimensional llamado con varias submatrices. Luego, se intenta imprimir dos elementos específicos de este arreglo:

- Elemento en la posición [4][1]: La submatriz en la posición 4 es {0, 4, 8, 12, 16}, y el elemento en la posición 1 de esta submatriz es 4.
- Elemento en la posición [1][4]. La submatriz en la posición 1 es {0, 1}, que solo tiene dos elementos (índices 0 y 1), pero al intentar acceder al índice 4 de esta submatriz causa una excepción de índice fuera de rango.

23. What is the result?

```
public class X {
     public static void main(String[] args) {
          String theString = "Hello World";
          System.out.println(theString.charAt(11));
     }
}
```

A. There is no output.
B. d is output.
C. A StringIndexOutOfBoundsException is thrown at runtime.
D. An ArrayIndexOutOfBoundsException is thrown at runtime.
E. A NullPointException is thrown at runtime.
F. A StringArrayIndexOutOfBoundsException is thrown at runtime.

Explicación:

En resumen, el código provoca una `StringIndexOutOfBoundsException` porque el índice 11 está fuera del rango de la longitud de la cadena.

24. What is the result?

```
public class Bees {
	public static void main(String[] args) {
		try {
			new Bees().go();
		}catch (Exception e) {
		System.out.println("thrown to main");
		}
	}

	synchronized void go() throws InterruptedException {
		Thread t1 = new Thread();
		t1.start();
		System.out.print("1 ");
		t1.wait(5000);
		System.out.print("2 ");
	}
}
```

A. The program prints 1 then 2 after 5 seconds.
B. The program prints: 1 thrown to main.
C. The program prints: 1 2 thrown to main.
D. The program prints: 1 then t1 waits for its notification.

Explicación:

El programa imprime "1 thrown to main" porque el método wait se llama sin el contexto sincronizado adecuado. En el método go, se intenta hacer que el hilo t1 espere 5000 milisegundos, pero wait debe llamarse desde un bloque sincronizado con el objeto que se espera. Como esto no ocurre, se lanza una IllegalMonitorStateException, que es capturada en el bloque catch del método main, imprimiendo "thrown to main".

25. ¿Cuál será el resultado?

```
public class SampleClass {
	public static void main(String[] args) {
		SampleClass sc, scA, scB;
		sc = new SampleClass();
		scA = new SampleClassA();
		scB = new SampleClassB();
		System.out.println(“Hash is: “ + sc.getHash() +
		“, “ + scA.getHash() + “, “ + scB.getHash());
	}
```

```
	public int getHash() {
		return 111111;
	}
}
class SampleClassA extends SampleClass {
	public int getHash() {
		return 44444444;
	}
}
class SampleClassB extends SampleClass {
	public int getHash() {
		return 999999999;
	}
}
```

a) Compilation fails
b) An exception is thrown at runtime
c) There is no result because this is not correct way to determine the hash code
d) Hash is: 111111, 44444444, 999999999.

Explicación:

El programa imprimirá Hash is: 111111, 44444444, 999999999. Esto se debe a que cada instancia (sc, scA, scB) llama a su propia versión del método getHash(), mostrando los valores específicos definidos en cada clase. sc usa el método de SampleClass, scA usa el de SampleClassA, y scB usa el de SampleClassB.

26. ¿Cuál sería el resultado?

```
public class Test {
		public static void main(String[] args) {
				int b = 4;
				b-;
				System.out.println(--b);
				System.out.println(b);
		}
}
```

a) 2 2
b) 1 2
c) 3 2
d) 3 3

Explicación:

La variable b comienza con el valor 4. Primero, se aplica el operador b--, que reduce b a 3 . Luego, se utiliza --b, que decrementa b nuevamente a 2 y este valor se imprime. Finalmente, se imprime el valor actual de b, que es 2. Por lo tanto, el resultado del programa será 2 y 2.

27. ¿Cuál sería el resultado?

```
import java.util.*;
public class App {
	public static void main(String[] args) {
		List p = new ArrayList();
		p.add(7);
		p.add(1);
		p.add(5);
		p.add(1);
		p.remove(1);
		System.out.println(p);
	}
}
```

a) [7, 1, 5, 1]
b) [7, 5, 1]
c) [7, 5]
d) [7, 1]

Explicación:

El código crea una lista de enteros y añade los valores 7, 1, 5 y 1, en ese orden. Luego, elimina el primer elemento que coincide con el valor 1. La lista inicial contiene los elementos [7, 1, 5, 1]. Al eliminar el primer 1, la lista se convierte en [7, 5, 1]. Así que el resultado final que se imprimirá es [7, 5, 1].

28. ¿Cuál sería el resultado?

```
public classDoCompare4 {
	public static void main(String[] args) {
		String[] table = {“aa”, “bb”, “cc”};
		int ii = 0;
		do {
			while (ii < table.length) {
				System.out.println(ii++);
			}
		} while (ii < table.length);
```

```
        }
}
```

a) 0
b) 0 1 2
c) 0 1 2 0 1 2 0 1 2
d) Compilation fails

Explicación:

El bucle do-while ejecuta el bloque al menos una vez. Dentro de este bloque, el bucle while imprime los valores de ii desde 0 hasta 2, y luego el do-while termina porque la condición ya no se cumple.

29. ¿Cuál sería el resultado?

```
public class DoCompare1 {
    public static void main(String[] args) {
        String[] table = {"aa", "bb", "cc"};
        for (String ss : table) {
            int ii = 0;
            while (ii < table.length) {
                System.out.println(ss + ", " + ii);
                ii++;
            }
        }
    }
}
```

A) Zero.
B) Once.
C) Twince
D) Thrice
E) Compilation fails

Explicación:

El programa imprime 9 líneas en total. Cada elemento del array table (es decir, "aa", "bb", "cc") se combina con los números 0, 1, y 2. Esto ocurre porque para cada elemento del array, el bucle while recorre los números del 0 al 2, imprimiendo el elemento actual del array table y el número en cada iteración. Por lo tanto, se imprimen 3 líneas por cada uno de los 3 elementos del array, resultando en un total de 9 líneas con la siguiente salida:

aa, 0,

aa, 1,
aa, 2,
bb, 0,
bb, 1,
bb, 2,
cc, 0,
cc, 1,
cc, 2

30. What is the result?

```
public class Boxer1 {
    Integer i = 0; // Inicializar i con 0
    int x;

    public Boxer1(int y) {
        x = i + y;
        System.out.println(x);
    }
    public static void main(String[] args) {
        new Boxer1(new Integer(4));
    }
}
```

A. The value “4” is printed at the command line.
B. Compilation fails because of an error in line 5.
C. Compilation fails because of an error in line 9.
D. A NullPointerException occurs at runtime.
E. A NumberFormatException occurs at runtime.
F. An IllegalStateException occurs at runtime.

Explicación:

El programa imprime el valor 4 porque en el constructor de Boxer1, x se calcula como la suma de i (un Integer con valor 0, convertido a int mediante *autounboxing*) y y (un int con valor 4). La compilación es exitosa porque el autounboxing convierte el Integer a int sin problemas, y no se producen excepciones en tiempo de ejecución.

31. Which code fragments is illegal ?

```
A. Class Base1 { abstract class Abs1 { } }

B. Abstract class Abs2 { void doit() { } }
```

```
C. class Base2 { abstract class Abs3 extends Base2 { } }
```

```
D. class Base3 { abstract int var1 = 89; }
```

Explicación:

La opción ilegal es **D**. No se pueden tener variables abstractas en una clase. Las variables deben estar completamente inicializadas, y `abstract` solo se usa para métodos y clases, no para variables. Las otras opciones son legales y válidas en Java.

32. ¿Cuál sería el resultado?

```
public class ScopeTest {
    int z;
    public static void main(String[] args) {
        ScopeTest myScope = new ScopeTest();
        int z = 6;
        System.out.print(z);
        myScope.doStuff();
        System.out.print(z);
        System.out.print(myScope.z);
    }

    void doStuff() {
        int z = 5;
        doStuff2();
        System.out.print(z);
    }

    void doStuff2() {
        z = 4;
    }
}
```

A. 6564
B. 6554
C. 6566
D. 6565

Explicación:

En el método main, se declara una variable z con el valor 6 y se imprime. Luego se llama a doStuff, donde se declara otra variable z con el valor 5. En el método doStuff2, se cambia el valor del campo z de la clase a 4. Después, doStuff imprime su variable local z con el valor 5. Finalmente, en main, se imprime la variable z local (que sigue siendo 6) y el campo z de la clase (que es 4).

33.

```
class Foo {
    public int a = 3;
    public void addFive() {a += 5; System.out.print("f"); }
}

class Bar extends Foo {
    public int a = 8;
    public void addFive() { this.a += 5; System.out.print("b");}
}

public class Main {
    public static void main(String[] args) {
        Foo f = new Bar();
        f.addFive();
        System.out.println(f.a);
    }
}
```

A. b 3
B. b 8
C. b 13
D. f 3
E. f 8
F. f 13
G. Compilation fails
H. An exception is thrown at runtime

Explicación:

El programa imprime "b" debido a la ejecución del método addFive() de la clase Bar, que modifica la variable a en Bar, y luego imprime 3, que es el valor del campo a en la clase Foo (ocultado en Bar).

34. Which one will compile, and can be run successfully using the following command? java Fred1 hello walls

```
A. class Fred1 { public static void main(String args) {
   System.out.println(args[1]); } }
B. abstract class Fred1 { public static void main(String[] args) {
   System.out.println(args[2]); } }
C. class Fred1 { public static void main(String[] args) {
   System.out.println(args); } }
D. class Fred1 { public static void main(String[] args) {
   System.out.println(args[1]); } }
```

Explicación:

Imprime el segundo argumento pasado en la línea de comandos. Si se ejecuta java Fred1 hello walls, imprimirá walls ya que args[1] corresponde al segundo argumento.

35. What is printed out when the program is executed?

```
public class MainMethod {
        void main() {
                System.out.println("one");
        }
        static void main(String args) {
                System.out.println("two");
        }
        public static final void main(String[] args) {
                System.out.println("three");
        }
        void mina(Object[] args) {
                System.out.println("four");
        }
}
```

a. one
b. two
c. three
d. four
e. There is no output.

Explicación:

Al ejecutar este código, se imprimirá "three". Esto se debe a que el método main que es ejecutado por la JVM tiene la firma public static void main(String[] args). Aunque hay otros métodos main definidos con diferentes firmas, solo este método específico es reconocido como el punto de entrada del programa.

37. Which five methods, inserted independently at line 5, will compile? (Choose five)

```
1.  public class Blip {
2.      protected int blipvert(int x) { return 0; }
3.  }
4.  class Vert extends Blip {
5.      // insert code here
6.  }
```

A. Private int blipvert(long x) { return 0; }
B. Protected int blipvert(long x) { return 0; }
C. Protected long blipvert(int x, int y) { return 0; }
D. Public int blipvert(int x) { return 0; }
E. Private int blipvert(int x) { return 0; }
F. Protected long blipvert(int x) { return 0; }
G. Protected long blipvert(long x) { return 0; }

Explicación:

Para que los métodos en la línea 5 compilen, deben cumplir con reglas de sobrecarga y anulación. Los cinco métodos que siguen estas reglas son:

1. Public int blipvert(int x): Anula el método de la clase base con mayor visibilidad.
2. Protected int blipvert(long x): Sobrecarga con un parámetro de tipo diferente.

3. Private int blipvert(long x): Sobrecarga y cambia la visibilidad a "private".

4. Protected long blipvert(int x, int y): Sobrecarga añadiendo un parámetro adicional.

5. Protected long blipvert(long x): Sobrecarga con un tipo de parámetro y tipo de retorno diferente.

38. What value of x, y, z will produce the following result? 1234,1234,1234 ------, 1234, ------

```
public static void main(String[] args) {
        // insert code here
        int j = 0, k = 0;
        for (int i = 0; i < x; i ++) {
                do {
                        k = 0;
                        while (k < z) {
                                k++;
                                System.out.print(k + " ");
                        }
                        System.out.println(" ");
                        j++;
                } while (j < y);
                System.out.println("---");
        }
}
```

A. int x = 4, y = 3, z = 2;
B. int x = 3, y = 2, z = 3;
C. int x = 2, y = 3, z = 3;
D. int x = 2, y = 3, z = 4;
E. int x = 4, y = 2, z= 3;

Explicación:

Con los valores x = 2, y = 3, z = 4 pasa lo siguiente:

- z=4 hace que el bucle while imprima los números del 1 al 4 en cada iteración del bucle do-while, dando el resultado de 1234.
- y=3 hace que el bucle do-while interno se ejecute tres veces por cada iteración del bucle externo, y por eso se imprime 1234 1234 1234
- x=2 hace que el bucle externo for se ejecute dos veces, por eso se imprime ------

39. What is the result if the integer value is 33?

```
public static void main(String[] args) {
    if (value >= 0) {
        if (value != 0) {
            System.out.print("the ");
        } else {
            System.out.print("quick ");
        }
        if (value < 10) {
            System.out.print("brown");
        }
        if (value > 30) {
                System.out.print("fox ");
```

```
        } else if (value < 50) {
                System.out.print("jumps ");
        } else if (value < 10) {
                System.out.print("over ");
        } else {
                System.out.print("the ");
            }
        if (value > 10) {
            System.out.print("lazy");
        } else {
            System.out.print("dog");
        }
        System.out.print("... ");
    }
}
```

A. The fox jump lazy ?
B. The fox lazy ?
C. Quick fox over lazy ?
D. Quick fox the ?

Explicación

- **"the "** - porque 33 es diferente `0`.
- **"fox "** - porque 33 es mayor que `30`.
- **"lazy"** - porque 33 es mayor que `10`.
- **"..."** - siempre se imprime al final.

40. Which two declarations will compile?

```
public static void main(String[] args) {  //14
    int a, b, c = 0;                      //15
    int a, b, c;                          //16
    int g, int h, int i = 0;              //17
    int d, e, f;                          //18
    int k, l, m = 0;                      //19
}                                         //20
```

a. Line 15
b. Line 16
c. Line 17
d. Line 18
e. Line 19
f. Line 20

Explicación:

- Line 15: int a, b, c = 0; (se declara e inicializa c a 0)
- Line 18: int d, e, f; (se declaran d, e, y f sin inicializarlos)
- Line 19: int k, l, m = 0; (se declara e inicializa m a 0)

41. What code should be inserted?

```
public class Bark {
   // Insert code here - Line 5
   public abstract void bark();
   // Insert code here - Line 9
   public void bark() {
      System.out.println("woof");
   }
}
```

A. 5. class Dog {9. public class Poodle extends Dog {
B. 5. abstract Dog {9. public class poodle extends Dog {
C. 5. abstract class Dog {9. public class Poodle extends Dog {
D. 5. abstract Dog {9. public class Poodle implements Dog {
E. 5. abstract Dog {9. public class Poodle implements Dog {
F. 5. abstract class Dog {9. public class Poodle implements Dog {

Explicación:

- Línea 5: abstract class Dog {
- Línea 9: public class Poodle extends Dog {

Esto asegura que Dog sea una clase abstracta y que Poodle extienda dicha clase adecuadamente.

42. ¿Cuál de las siguientes opciones inicializa un StringBuilder con una capacidad de 128 caracteres?

A: StringBuilder sb = new String("128");
B: StringBuilder sb = StringBuilder.setCapacity(128);
C: StringBuilder sb = StringBuilder.getInstance(128);

D: StringBuilder sb = new StringBuilder(128);

Explicación:

La opción D es la única que crea un nuevo objeto StringBuilder y establece su capacidad inicial en 128 caracteres.

43. . What is the result?

```
class MyKeys {
        Integer key;
        MyKeys(Integer k) { key = k; }
        public boolean equals(Object o) {
                return ((MyKeys) o).key == this.key;
        }
}
```

And this code snippet:

```
Map m = new HashMap();
MyKeys m1 = new MyKeys(1);
MyKeys m2 = new MyKeys(2);
MyKeys m3 = new MyKeys(1);
MyKeys m4 = new MyKeys(new Integer(2));
m.put(m1, "car");
m.put(m2, "boat");
m.put(m3, "plane");
m.put(m4, "bus");
System.out.print(m.size());
```

A. 2
B. 3
C. 4
D. Compilation fails.

Explicación:

El resultado del código es 4 debido a que:

1. Se crean cuatro objetos MyKeys con valores de claves distintos o iguales.
2. Aunque m1 y m3 tienen el mismo valor de clave 1, y m2 y m4 tienen el mismo valor de clave 2, el método hashCode no se ha sobrescrito en la clase MyKeys.
3. La falta de sobrescritura del método hashCode hace que cada instancia se trate como única en el HashMap, ya que se compara la igualdad utilizando el método equals y no por el valor de la clave.
4. Por tanto, el HashMap mantiene las cuatro entradas separadas, resultando en un tamaño de 4.

44. What changes will make this code compile?

```
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();
      }
}
```

A. Adding the public modifier to the declaration of class X.
B. Adding the protected modifier to the class X() constructor.
C. Changing the private modifier on the declaration of the one() method to protected.
D. Removing the Y() constructor.
E. Removing the private modifier from the two() method

Explicación:

Para que el código compile, se debe cambiar el modificador del método `one()` en la clase `X` de `private` a `protected`. Esto permitirá que la clase `Y` acceda al método `one()` de la clase `X`.

45. What is the best way to test that the values of h1 and h2 are the same?

```
public static void main(String[] args) {
      String h1 = "Bob";
      String h2 = new String("Bob");
}
```

a. if(h1==h2)
b. if(h1.equals(h2))
c. if(h1.toString()== h2.toString())
d. if(h1.same(h2))

Explicación:

Para probar si los valores de h1 y h2 son iguales en contenido, la mejor manera es usar h1.equals(h2). Este método compara el contenido de las cadenas y devolverá true si ambos String tienen el mismo contenido.

46. Which three are valid? (Choose three)

```
class ClassA {}
class ClassB extends ClassA {}
class ClassC extends ClassA {}
And:
ClassA p0 = new ClassA();
ClassB p1 = new ClassB();
ClassC p2 = new ClassC();
ClassA p3 = new ClassB();
ClassA p4 = new ClassC();
```

A. p0 = p1;
B. p1 = p2;
C. p2 = p4;
D. p2 = (ClassC)p1;
E. p1 = (ClassB)p3;
F. p2 = (ClassC)p4;

Explicación:

- p0 = p1; Esto es válido porque ClassB es una subclase de ClassA, por lo que un objeto de tipo ClassB puede ser asignado a una variable de tipo ClassA.
- p1 = (ClassB)p3; Esta asignación es válida porque p3 es de tipo ClassA, pero en realidad es un objeto ClassB (polimorfismo). Al hacer un cast a ClassB se puede asignar a p1.
- p2 = (ClassC)p4; Similar al caso anterior, p4 es de tipo ClassA, pero en realidad es un objeto ClassC. Al hacer un cast a ClassC se puede asignar a p2.

47. What is the result?

```
public static void main(String[] args) {
    String color = "Red";
    switch (color) {
        case "Red":
            System.out.println("Found Red");
        case "Blue":
            System.out.println("Found Blue");
        case "White":
            System.out.println("Found White");
            break;
        default:
            System.out.println("Found Default");
```

```
    }
}
```

A. Found Red.
B. Found Red Found Blue.
C. Found Red Found Blue Found White.
D. Found Red Found Blue Found White Found Default

Explicación:

El código evalúa el valor de una variable llamada "color", que contiene la palabra "Red". Utiliza una estructura switch para tomar decisiones basadas en este valor. Sin embargo, debido a la falta de la palabra reservada break (que se encarga de interrumpir la ejecución de un bloque de código) en algunos casos, el programa continúa ejecutando el código de los casos siguientes. Por esta razón, el programa imprime "Found Red", "Found Blue" y "Found White" en la consola, aunque solo el valor "Red" coincida con un caso.

48. What is the result?

```
class MySort implements Comparator<Integer> {
        public int compare(Integer x, Integer y) {
                return y.compareTo(x);
        }
}

And the code fragment:

Integer[] primes = {2, 7, 5, 3};
MySort ms = new MySort();
Arrays.sort(primes, ms);
for (Integer p2 : primes) { System.out.print(p2 + " "); }
```

A. 2 3 5 7
B. 2 7 5 3
C. 7 5 3 2
D. Compilation fails.

Explicación:

Se define una clase llamada MySort que implementa la interfaz Comparator. El método compare de esta clase compara dos números enteros, x y y, en orden inverso al habitual (de mayor a menor) utilizando y.compareTo(x). A continuación, se crea un arreglo con los valores {2, 7, 5, 3} y se ordena utilizando una instancia de MySort como comparador. Debido a esta lógica de comparación, el arreglo se imprimirá en la siguiente secuencia de números: 7 5 3 2.

49-What is the result?

```
public class Calculator {
    int num = 100;
    public void calc(int num) {
        this.num = num * 10;
    }

    public void printNum() {
        System.out.println(num);
    }

    public static void main(String[] args) {
        Calculator obj = new Calculator();
        obj.calc(2);
        obj.printNum();
    }
}
```

a-20
b-100
c-1000
d-2

Explicación:

50-¿Qué tres modificaciones, hechas de manera independiente, permitirán que la clase Greet compile y se ejecute?

```
package handy.dandy;
public class KeyStroke {
    public void typeExclamation() {
        System.out.println("!");
    }
}

package handy;
public class Greet {
    public static void main(String[] args) {
        String greeting = "Hello";
        System.out.print(greeting);
        KeyStroke stroke = new KeyStroke();
        stroke.typeExclamation();
```

```
    }
}
```

A. Line 8 replaced with handy.dandy.KeyStroke stroke = new KeyStroke();
B. Line 8 replaced with handy.*.KeyStroke stroke = new KeyStroke();
C. Line 8 replaced with handy.dandy.KeyStroke stroke = new handy.dandy.KeyStroke();
D. import handy.*; added before line 1.
E. import handy.dandy.*; added after line 1.
F. import handy.dandy.KeyStroke; added after line 1.
G. import handy.dandy.KeyStroke.typeExclamation(); added after line 1.

Explicación:

Opción A: Al usar handy.dandy.KeyStroke en lugar de simplemente KeyStroke, el compilador sabe exactamente dónde encontrar la clase KeyStroke.

Opción E: Importar handy.dandy.* permite que la clase KeyStroke sea accesible en el código sin necesidad de usar el nombre completo del paquete.

Opción F: Importar handy.dandy.KeyStroke directamente hace que la clase esté disponible para su uso sin necesidad de calificarla con el nombre completo del paquete.

51-What is the result?

```
class Feline {
    public String type = "f ";
    public Feline() {
        System.out.print("feline ");
    }
}

public class Cougar extends Feline {
    public Cougar() {
        System.out.print("cougar ");
    }

    void go() {
        type = "c ";
        System.out.print(this.type + super.type);
    }
```

```
        public static void main(String[] args) {
            new Cougar().go();
        }
    }
```

A. Cougar c f.
B. Feline cougar c c.
C. Feline cougar c f.
D. Compilation fails.

Explicación:

El programa imprime "feline cougar c c". Primero, el constructor de la superclase Feline imprime "feline", luego el constructor de Cougar imprime "cougar". Finalmente, el método go() cambia el valor de type a "c " y lo imprime junto con el valor actualizado de super.type, resultando en "c c".

52. 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);
        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

Explicación:

El resultado es 111.234 222.5678 porque el método setMaximumFractionDigits(3) limita los dígitos decimales a tres al convertir números a cadenas. Sin embargo, NumberFormat.parse() convierte las cadenas a números sin aplicar el límite de dígitos decimales, por lo que el formato original de las cadenas se conserva.

53-What is the result?

```
interface Rideable {
    String getGait();
}

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);
    }
}
```

A. 16 mph, lope.
B. 24 mph, lope.
C. 27 mph, lope.
D. Compilation fails.

Explicación:

El código no compila debido a un error en el método getGait(). El código intenta usar la variable mph, que no está definida en ninguna parte de la clase Camel.

54-¿Cuáles de las siguientes opciones son instanciaciones e inicializaciones válidas de un arreglo multidimensional? Elge dos

a-int[][] array2D = {{0, 1, 2, 4}{5, 6}};
  int[][] array2D = new int[2][2];

b- array2D[0][0] = 1;
   array2D[0][1] = 2;
   array2D[1][0] = 3;

```
    array2D[1][1] = 4;
    int[] array3D = new int[2][2][2];

c-int[][] array3D = {{{0, 1}, {2, 3}, {4, 5}}};
  int[] array = {0, 1};

d-array3D[0][0] = array;
  array3D[0][1] = array;
  array3D[1][0] = array;
  array3D[1][1] = array;
```

Explicación:

La opción **a** es válida porque puedes definir un arreglo bidimensional con valores específicos o inicializar un arreglo vacío con un tamaño definido. La opción **b** es válida para asignar valores a un arreglo bidimensional ya creado.