1. **Which declaration initializes a boolean variable?**

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

Esta declaración utiliza la clase Boolean, que es un objeto, y el resultado de la expresión (1 < 5) es true, por lo que se asigna correctamente.

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

El patrón DTO (Data Transfer Object) se utiliza para transferir datos entre diferentes capas o procesos de una aplicación, generalmente para optimizar el rendimiento al minimizar las llamadas remotas.

3. **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;

What is the result?

a) 30
b) 0
c) 60

If (a==b): Evalúa si a es igual a b. Aquí, a (10) no es igual a b (37), por lo que esta condición es falsa.

else if (a>b): Evalúa si a es mayor que b. En este caso, a (10) no es mayor que b (37), por lo que también es falsa.

Como ninguna de las condiciones es verdadera, el valor de z permanece en 0.

w = 10 * z;: Como z sigue siendo 0, el valor de w será 10 * 0, que es 0.

4.
```
class Class1{String v1;}
class Class2{
          Class1 c1;
          String v2;
}
class Class3 {Class2 c1; String v3;}
```

Which three options correctly describe the relationship between the classes?

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.

Class1 tiene un atributo de tipo String llamado v1.

Class2 tiene dos atributos: uno es una instancia de Class1 llamado c1 y otro es un String llamado v2.

Class3 tiene dos atributos: uno es una instancia de Class2 llamado c1 y otro es un String llamado v3.

5.
```
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) {}
```

What is the result?

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.

Si se hace rollback a s1, se deshace todo lo que ocurrió después de este punto, lo que significa que solo el primer incremento de 10 se mantendría.

Si se hace rollback a s2, el incremento de 1000 se deshace, pero se mantienen los incrementos anteriores de 10 y 100, sumando 110 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.

Una clase abstracta puede implementar una o más interfaces. Aunque la clase abstracta no esté obligada a proporcionar implementaciones para todos los métodos de la interfaz, las clases que extienden la clase abstracta deben hacerlo.
Una clase concreta puede extender una clase abstracta y debe proporcionar implementaciones para todos los métodos abstractos que no hayan sido implementados por la clase abstracta.

7.
```
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");
  }
}
```

Which two are possible outputs?

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

La opción **a** se produce cuando se evalúa en el if si el valor generado por Math.random() es mayor que 0.5. En ese caso, se lanza una excepción y no se ejecuta la línea

System.out.println("After if clause");. Por lo tanto, solo se imprime "Before if clause" seguido del stack trace de la excepción.

La opción **d** se produce cuando el valor generado por Math.random() es menor o igual a 0.5. Debido a que no entra en el if, no se lanza una excepción y ambas líneas System.out.println("Before if clause"); y System.out.println("After if clause"); se ejecutan.

```
8. 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);
    }
}
```

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

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

El bucle for continuará decrementando ii y aumentando jj mientras ii > 6. El valor de jj se incrementa cada vez que el bucle se ejecuta.

El bucle debe ejecutarse 5 veces para que jj alcance el valor de 5

Si kk = 11:

Comienza el bucle: ii = 11

Iteraciones: ii = 11, 10, 9, 8, 7 → 5 iteraciones, luego ii = 6 y el bucle se detiene.

Al finalizar, jj habrá sido incrementado 5 veces, resultando en jj = 5.

9.
```
public class Simple {
  public float price;
  public static void main(String[] args) {
      Simple simpleObject = new Simple();
      simpleObject.price = 4;
  }
}
```

What will make this code compile and run?

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;**

Si quisiéramos cambiar la referencia de el price de la clase, tendríamos que ingresar al objeto para hacerlo, de otra forma estamos intentando cambiar la referencia de Simple price, y 4 (int) no es de la clase Simple.

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.**

Un Set pertenece a collection y no permite el tener elementos duplicados en su interior.

Los otros dos puntos hacen referencia a List y Map, respectivamente.

11.
```
public class SuperTest {
    public static void main(String[] args) {
    //statement1
    //statement2
    //statement3
    }
}
class Shape {
    public Shape() {
        System.out.println("Shape: constructor");
    }
    public void foo() {
        System.out.println("Shape: foo");
    }
}
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");
    }
}
```

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

Shape: constructor

Shape: foo

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();**

Solo en la opción D obtenemos el resultado que buscamos, ya que:

Square square = new Square();: Imprime "Shape: constructor".

square.foo();: Imprime "Shape: foo".

square.foo("bar");: Imprime "Square: foo".

12.
```
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{}
```

What is the result?

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

getClass evalúa el la clase de el objeto, y no de la variable de referencia, por lo que al pasarle a sc el objeto AnotherSampleClass() , nos da este resultado.

13.
```
public abstract class Wow{
    private int wow;
    public Wow(int wow){this.wow = wow;}
    public void wow(){}
    private void wowza(){}
}
```

What is true about the class Wow?

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.

La clase Wow es una clase abstracta que cumple con las reglas de Java. Aunque tiene métodos privados y un constructor con un argumento, estos elementos son válidos en una clase abstracta. La clase abstracta no necesita tener métodos abstractos ni constructores sin argumentos. Además, los métodos privados no afectan la capacidad de la clase para compilar, ya que son simplemente métodos internos que no se pueden acceder desde fuera de la clase.

It does not compile because an abstract class cannot have private methods.

- Incorrecto: Una clase abstracta puede tener métodos privados. Los métodos privados son accesibles sólo dentro de la clase en la que están definidos y no afectan la compilación de la clase abstracta.

It does not compile because an abstract class cannot have instance variables.

- Incorrecto: Las clases abstractas pueden tener variables de instancia. En este caso, la variable de instancia wow está correctamente definida y no causa problemas de compilación.

It does not compile because an abstract class must have at least one abstract method.

- Incorrecto: Aunque es una buena práctica que las clases abstractas contengan al menos un método abstracto, no es un requisito para la compilación. Una clase abstracta puede no tener métodos abstractos y aún así compilar correctamente.

It does not compile because an abstract class must have a constructor with no arguments.

- Incorrecto: No es necesario que una clase abstracta tenga un constructor sin argumentos. La clase abstracta Wow tiene un constructor con un argumento, lo cual es válido en Java.

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.

Respuesta: B. Having a single instance of a class, while allowing all classes have access to that instance

15.
```
public static void main(String[] args) {
    String[] table = {"aa", "bb", "cc"};
    int i = 0;
```

```
      for (String ss : table) {
            while (ii < table.length) {
            System.out.println(ii); ii++;
            break;
            }
      }
}
```

How many times is 2 printed?

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

Dentro del foreach tenemos un ciclo que implementa un break sin condición por lo que el ciclo while no continuará después de imprimir e incrementar el valor de “ii”, el valor de “ii” se incrementará hasta 2, debido a la expresión booleana dada en el ciclo. Así que sólo una vez tendrá el valor de 2 y solo se podrá imprimir una vez.

```
   16.      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]);
}
```

What is the result?

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

En la primera impresión se obtiene el tamaño del primer arreglo en array2D que es de 3, la segunda impresión obtiene la clase a la que pertenece el segundo arreglo almacenado en array2D, y verifica si es de tipo Array, por lo que se obtiene true, y la última impresión imprime el entero almacenado en el primer arreglo que se encuentra en el índice 1, y corresponde a 1.

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.**

Esto mezcla los conceptos de throws y throw. throw lanza una excepción, mientras que throws se usa para declarar excepciones que un método puede lanzar.

Tanto throws como throw pueden usarse en métodos y constructores. No hay una distinción en su uso basada en métodos versus constructores.

throws se utiliza para indicar las excepciones que un método no maneja y, por lo tanto, puede lanzar. throw se utiliza para lanzar una excepción específica.

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.**

El constructor de Employee tiene un carácter no válido justo antes de id.

19. Which is true?

```
5. class Building {}
6. public class Barn extends Building {
7.    public static void main(String[] args) {
8.    Building build1 = new Building();
9.    Barn barn1 = new Barn();
10.   Barn barn2 = (Barn) build1;
11.   Object obj1 = (Object) build1;
12.   String str1 = (String) build1;
13.   Building build2 = (Building) barn1;
14.   }
15. }
```

- **If line 10 is removed, the compilation succeeds.**
- **If line 11 is removed, the compilation succeeds.**
- **If line 12 is removed, the compilation succeeds.**
- **If line 13 is removed, the compilation succeeds.**
- **More than one line must be removed for compilation to succeed.**

Esto causará un error de compilación porque no se puede convertir un Building a String.

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();}
}
```

- **Compilation fails.**
- **Atom granite.**
- **Granite granite.**
- **Atom granite granite.**
- **An exception is thrown at runtime.**
- **Atom granite atom granite.**

La salida será la combinación de las impresiones de ambos constructores:

1) "atom " por la primera llamada al constructor de Atom desde Rock.
2) "granite " por el constructor de Rock invocado por super("granite ").

3) "atom " por la segunda llamada al constructor de Atom cuando se crea una nueva instancia de Rock.
4) "granite " por el constructor de Rock invocado con new Rock("granite ")

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);
    }
}
```

- **420 is the output.**
- **An exception is thrown at runtime.**
- **All constructors must be declared public. Constructors CANNOT use the private modifier.**
- **Constructors CANNOT use the protected modifier.**

La salida del programa es 420 porque cuando se crea una instancia de ExtendedA con el valor 420, el constructor de ExtendedA llama al constructor de ClassA con super(numberOfInstances). Esto asigna el valor 420 al campo numberOfInstances en ClassA.

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]);
      }
}
```

- **4 Null.**
- **Null 4.**
- **An IllegalArgumentException is thrown at run time.**
- **4 An ArrayIndexOutOfBoundsException is thrown at run time.**

En la Segunda impression en consola, se sale del tamaño de ese array, solo tiene 2 elementos y lo busca en el index 4

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.

El método charAt(int index) devuelve el carácter en la posición especificada en la cadena.

Sin embargo, el índice proporcionado aquí es 11. Dado que los índices válidos para esta cadena van de 0 a 10, intentar acceder al índice 11 resulta en un error.

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 ");
      }
}
```

- **The program prints 1 then 2 after 5 seconds.**
- **The program prints: 1 thrown to main.**
- **The program prints: 1 2 thrown to main.**
- **The program prints:1 then t1 waits for its notification.**

Esto se debe a que t1.wait(5000); lanza una excepción IllegalMonitorStateException al no estar en un bloque sincronizado, y dicha excepción es capturada e imprime el mensaje "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.**

sc es una instancia de SampleClass, scA es una instancia de SampleClassA y scB es una instancia de SampleClassB. Tanto scA y scB están haciendo un Override al método getHash();. El método main imprimirá los valores de las instancias, dando como resultado sc 111111, scA 44444444 y scB 999999999

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

b se inicializa en 4, posterior se convierte el 3 por el decremento, posteriormente se le aplica un predecremento lo que le da un valor de 2 e imprime el valor de b con el predecremento. En la última línea se pide imprimir el valor de b, el cual ahora es un 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]

Dentro del código se añaden 7, 1, 5 y 1 con el p.add. Posterior a esto, con p.remove se quita el elemento del índice 1, por lo que el primer 1 se elimina, dejando así 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)
d) 0 1 2 0 1 2 0 1 2

Inicia en 0, se incrementa en 1 por lo que ya es 1 que sigue siendo menos que la longitud, se incrementa en 1 la i y ahora 2, por lo que sigue siendo menor a la longitud, por lo que imprime 012Compilation fails

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;9
            while (ii < table.length) {
                System.out.println(ss + ", " + ii);
                ii++;
            }
        }
    }
}
```

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

El resultado final de las impresiones será:aa, 2 -bb, 2 - cc, 2 = 3 veces sale el 2.

aa, 0

aa, 1

aa, 2

bb, 0

bb, 1

bb, 2

cc, 0

cc, 1

cc, 2

Es ver cada tarjeta una por una y escribir tres líneas para cada tarjeta, cada línea con la palabra en la tarjeta y un número del 0 al 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.

Para evitar el NullPointerException, debemos asegurarnos de que i esté inicializada antes de usarla en la operación. Aquí hay una versión corregida del código:

Ahora, cuando ejecutemos el código, y estará inicializada a 0, por lo que la operación x = i + y será x = 0 + 4, y se imprimirá 4.

31. What is the result?

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

- Esto es legal. Es posible tener una clase abstracta dentro de otra clase.

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

- Esto es legal. Una clase abstracta puede tener métodos concretos.

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

- Esto es legal. Es posible tener una clase abstracta que extiende otra clase dentro de una clase.

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

- Esto es ilegal. Las variables no pueden ser abstractas. La palabra clave abstract solo se aplica a métodos y clases.

Por lo tanto, el fragmento de código ilegal es el **D**.

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

- **System.out.print(z);** en main imprime 6.
- **myScope.doStuff();** llama a doStuff:
  - **doStuff** declara int z = 5;.
  - **doStuff2** establece myScope.z = 4;.
  - **System.out.print(z);** en doStuff imprime 5.
- **System.out.print(z);** en main después de doStuff imprime 6.
- **System.out.print(myScope.z);** imprime 4.

Entonces, el orden de las impresiones y sus valores será 6564.

33. What is the result?

```
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 ");
    }
}

// Invocado con:
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.

Creación del objeto f:

Se crea un objeto de tipo Bar, pero se lo referencia como un objeto de tipo Foo:

Foo f = new Bar();

Llamada a addFive():

Aunque f es de tipo Foo, dado que el objeto real es de tipo Bar, se invoca el método addFive() de la clase Bar. Esto se debe al polimorfismo en Java.

En Bar, addFive() incrementa el valor de this.a en 5 y luego imprime "b ".

En la clase Bar, a es inicialmente 8, por lo que después de this.a += 5, a será 13.

Impresión del valor de f.a:

Después de ejecutar addFive(), se imprime f.a.

Aquí es importante notar que f es de tipo Foo, por lo que f.a referirá a la variable a de la clase Foo, que sigue siendo 3 (no ha sido modificada, ya que la variable a en Bar es independiente de la variable a en Foo).

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

A. class Fred 1{ 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]); } }

Esta opción es válida:

El nombre de la clase es correcto.

El método main tiene la firma correcta.

Accede al segundo argumento del array args (args[1]), que será "walls".

Resultado: Compila y se ejecuta correctamente, imprimiendo "walls".

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 void main(String[] args){
		System.out.println("three");
	void mina(Object[] args){
		System.out.println("four");
	}
}
```

- **one**
- **two**
- **three**
- **four**
- **There is no output.**

Método main() sin parámetros (void main())

- Este método no es el método main que Java busca para iniciar la ejecución del programa. Java no lo reconocerá como el punto de entrada del programa.

Método main(String args)

- Este es un método estático que acepta un solo argumento de tipo String. Aunque es un método válido, no es el método main que Java busca para iniciar la ejecución del programa. Además, este método no es llamado en el código proporcionado.

Método main(String[] args)

- Este es el método main estándar que Java busca para iniciar la ejecución del programa. Es el único método main que será ejecutado cuando el programa se inicie. Este método imprimirá "three".

Método mina(Object[] argos)

- Este método tiene un nombre diferente y no es llamado desde el main, por lo que no se ejecutará.

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 }
```

- **Private int blipvert(long x) { return 0; }**
- **Protected int blipvert(long x) { return 0; }**
- **Protected long blipvert(int x, int y) { return 0; }**
- **Public int blipvert(int x) { return 0; }**
- **Private int blipvert(int x) { return 0; }**
- **Protected long blipvert(int x) { return 0; }**
- **Protected long blipvert(long x) { return 0; }**

private int blipvert(long x) { return 0; }

- Compila. Este método define una nueva sobrecarga del método blipvert en la subclase Vert. Aunque el método es private, lo cual significa que no puede ser sobrescrito, es válido para la sobrecarga. La visibilidad private no afecta la posibilidad de sobrecargar métodos ya que la sobrecarga está relacionada con la firma del método (nombre y tipos de parámetros), no con la visibilidad.

protected int blipvert(long x) { return 0; }

- Compila. Este método es una sobrecarga válida del método blipvert heredado de Blip. La firma del método cambia porque el tipo del parámetro es diferente (de int a long). La visibilidad protected es válida para la subclase Vert.

protected long blipvert(int x, int y) { return 0; }

- Compila. Este método también es una sobrecarga válida porque el número de parámetros es diferente. La firma del método cambia porque se ha añadido un segundo parámetro (int y). La visibilidad protected es válida en la subclase Vert.

public int blipvert(int x) { return 0; }

- Compila. Cambiar la visibilidad de protected a public es permitido si la firma del método permanece igual (nombre del método y tipos de parámetros). En este caso, la firma es la misma y el cambio de visibilidad de protected a public es válido.

protected long blipvert(int x) { return 0; }

- Compila. Este método cambia el tipo de retorno a long, pero la firma del método sigue siendo diferente porque el tipo de parámetro es el mismo. La visibilidad protected es adecuada en la subclase Vert.

protected long blipvert(long x) { return 0; }

- Compila. Cambiar el tipo de parámetro es una forma válida de sobrecargar el método. La visibilidad protected es válida para la subclase Vert, y la firma del método cambia porque el tipo del parámetro es diferente (de int a long).

38. What values of x, y, z will produce the following result?

```
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.print(" ");
            j++;
        } while (j < y);
        System.out.println("---");
    }
}
```

1234

1234

1234

1234

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=4, y = 2, z=3;

E. int x 2, y 3,z=4;

Queremos obtener la salida 1 2 3 4 --- 1 2 3 4 ---.

Estructura de la salida:

1 2 3 4 se imprime dos veces, seguidas de "---".

Esto significa que k debe incrementarse hasta 4, lo que implica que z = 4.

La secuencia 1 2 3 4 se imprime dos veces, lo que sugiere que el bucle do-while se ejecuta dos veces, lo que implica que y = 2.

Finalmente, la secuencia completa se repite una vez más, lo que indica que x = 2.

Para obtener la salida 1 2 3 4 --- 1 2 3 4 ---, los valores correctos de x, y, y z deben ser:

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("... ");
      }
}
```

- **The fox jump lazy ?**
- **The fox lazy ?**
- **Quick fox over lazy ?**
- **The fox lazy…**

value es 33.

Condición if (value >= 0): Se cumple porque 33 es mayor que 0.

Condición if (value != 0): Se cumple porque 33 no es 0, por lo que se imprime "the ".

Condición if (value < 10): No se cumple porque 33 no es menor que 10, no se imprime nada.

Condición if (value > 30): Se cumple porque 33 es mayor que 30, por lo que se imprime "fox ".

Condición if (value > 10): Se cumple porque 33 es mayor que 10, por lo que se imprime "lazy".

Finalmente, se imprime "..."

40. Which two declarations will compile?

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

A. Line 15.

B. Line 16.

C. Line 17.

D. Line 18.

E. Line 19.

F. Line 20.

Línea 15: Int a, b, c = 0; - Esta línea no se compilará porque Java distingue entre mayúsculas y minúsculas, y Int no es una palabra clave válida (debería ser int).

Línea 16: int a, b, c; - Esta línea se compilará. Declara tres variables enteras a, b y c sin inicialización.

Línea 17: int g, int h, int i = 0; - Esta línea no se compilará porque no se puede declarar múltiples variables del mismo tipo en una sola línea usando repetidamente la palabra clave int. Debería ser int g, h, i = 0;.

Línea 18: int d, e, f; - Esta línea se compilará. Declara tres variables enteras d, e y f sin inicialización.

Línea 19: Int k, l, m, = 0; - Esta línea no se compilará debido al mismo problema que la línea 15 (Java distingue entre mayúsculas y minúsculas, y Int no es válido).

41. What code should be inserted?

```
4. public class Bark {
5.    // Insertar código aquí - Línea 5
6.    public abstract void bark();
7. }
8.
9. // Insertar código aquí - Línea 9
10. public void bark() {
11.    System.out.println("woof");
12. }
13. }
```

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{

En la Línea 5 se necesita definir una clase abstracta Dog que contiene el método abstracto bark(). Este método debe ser implementado por cualquier clase que extienda Dog.

En la Línea 9, la clase Poodle debe extender Dog y proporcionar una implementación concreta del método bark().

Correcta, abstract class Dog es una declaración válida para contener métodos abstractos, y Poodle puede extender Dog.

42. Which statement initializes a stringBuilder to a capacity of 128?

A. StringBuilder sb = new String("128");

B. StringBuilder sb = StringBuilder.setCapacity(128);

C. StringBuilder sb = StringBuilder.getInstance(128);

D. StringBuilder rsb = new StringBuilder(128);

Opción A: Incorrecta. Esta opción crea un objeto String con el valor "128", pero no tiene nada que ver con StringBuilder.

Opción B: Incorrecta. StringBuilder no tiene un método setCapacity() que se pueda usar de esta manera. El método para ajustar la capacidad existe, pero no se utiliza de esta forma.

Opción C: Incorrecta. StringBuilder no tiene un método estático getInstance().

Opción D: Correcta. Esta opción utiliza el constructor de StringBuilder que permite establecer la capacidad inicial a 128.

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;
    }
}

public class Main {
    public static void main(String[] args) {
        Map<MyKeys, String> 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());
    }
}
```

- **2**
- **3**
- **4**
- **Compilation fails.**

La respuesta es 4 porque, aunque la clase MyKeys sobrescribe el método equals, no sobrescribe el método hashCode. En Java, un HashMap utiliza tanto equals como hashCode para determinar la unicidad de las claves. Dado que MyKeys no sobrescribe hashCode, los objetos m1, m2, m3 y m4 se consideran diferentes incluso si tienen el mismo valor de key. Por lo tanto, cada llamada a m.put agrega una nueva entrada al mapa, resultando en cuatro entradas distintas.

44. What changes will make this code compile?

```
class X {
    X() {}  // Constructor
    private void one() {}  // Método privado
}

public class Y extends X {
    Y() {}  // Constructor
    private void two() {
        one();  // Llamando al método privado 'one()'
    }
    public static void main(String[] args) {
        new Y().two();  // Creando una instancia de Y y llamando a
'two()'
    }
}
```

A. Adding the public modifier to the declaration of class X.

B. Adding the protected modifier to the X() constructor.

C. Changing the private modifier on the declarationof the one() method to protected.

D. Removing the Y() constructor.

E. Removing the private modifier from the two() method.

Opción A: No es necesario hacer la clase X pública para que el código compile. Por defecto, una clase sin modificador tiene un nivel de acceso "package-private" y puede ser accedida por otras clases en el mismo paquete. No es necesario.

Opción B: El constructor X() no necesita ser protected para que el código compile, ya que la clase Y ya tiene acceso al constructor de X al estar en el mismo paquete. No es necesario.

Opción C: Correcta. El problema en el código es que one() es un método privado en X, lo que significa que no es accesible en la clase Y, incluso si Y hereda de X. Cambiar one() a protected permitiría que Y acceda a este método.

Opción D: Eliminar el constructor Y() no es necesario y no resolvería el problema de acceso al método one(). No es necesario.

Opción E: Eliminar el modificador private del método two() en Y no es relevante pa

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)).

Esta es la opción correcta. El método equals en la clase String compara el contenido de las dos cadenas. Como ambas cadenas tienen el mismo contenido "Bob", esta comparación devolvería true

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;

Opción A:

Válida. p1 es una instancia de ClassB, que es una subclase de ClassA. Se permite la asignación de una subclase a una referencia de superclase.

Opción B:

Inválida. p1 es una referencia de tipo ClassB y p2 es una instancia de ClassC. No se puede asignar una referencia de una clase que no es una subclase o superclase sin casting, y en este caso, ClassC y ClassB no tienen una relación de herencia directa entre sí.

Opción C:

Válida. p4 es una instancia de ClassC y p2 también es de tipo ClassC. Se permite esta asignación directa.

Opción D:

Inválida. p1 es una instancia de ClassB, que no es compatible con ClassC. Intentar hacer un casting de ClassB a ClassC producirá una ClassCastException en tiempo de ejecución.

Opción E:

Válida. p3 es una referencia de tipo ClassA que apunta a un objeto de tipo ClassB. Es posible hacer un casting de p3 a ClassB sin problemas porque p3 realmente apunta a un objeto de tipo ClassB.

Opción F:

Válida. p4 es una referencia de tipo ClassA que apunta a un objeto de tipo ClassC. Se puede hacer un casting de p4 a ClassC sin problemas porque p4 realmente apunta a un objeto de tipo ClassC.

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.

color = "Red", por lo tanto, se ejecutará el caso "Red":

Imprime: "Found Red"

Como no hay una declaración break después de System.out.println("Found Red");, el código continuará ejecutando los siguientes casos:

Imprime: "Found Blue"

Imprime: "Found White"

Después de case "White":, hay un break, por lo que la ejecución del switch se detiene antes de llegar al bloque default.

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. 2357

B. 2753

C. 7532

D. Compilation fails.

Clase MySort:

El método compare recibe dos enteros (x y y) y los compara de forma que devuelve un valor positivo si y es mayor que x. Esto hace que el arreglo se ordene en orden descendente.

Ordenación del arreglo primes:

El arreglo {2, 7, 5, 3} será ordenado por Arrays.sort utilizando la lógica de MySort, es decir, en orden descendente. Por lo tanto, el arreglo se convertirá en {7, 5, 3, 2}.

Impresión:

El ciclo for imprimirá los elementos del arreglo ordenado, separados por espacios.

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();  // Esto imprimirá 20
    }
}
```

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

El método calc toma un entero num como parámetro y establece la variable de instancia this.num en num * 10.
Cuando se llama a obj.calc(2), this.num se establece en 2 * 10, que es 20.
Luego, obj.printNum() imprime el valor de this.num, que ahora es 20.
Entonces, la salida del programa será 20.

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.

Para que la clase Greet compile y se ejecute correctamente, debes asegurarte de que el compilador pueda encontrar la clase KeyStroke desde el paquete handy.dandy en el que está definida. Aquí están las modificaciones correctas:

A. Línea 8 reemplazada por handy.dandy.KeyStroke stroke = new KeyStroke();

Esto especifica el paquete completo de la clase KeyStroke, asegurando que el compilador pueda encontrarla.
E. import handy.dandy.*; añadido después de la línea 1.

Esto importa todas las clases del paquete handy.dandy, permitiendo que puedas usar KeyStroke directamente en la clase Greet.
F. import handy.dandy.KeyStroke; añadido después de la línea 1.

Esto importa específicamente la clase KeyStroke, lo que te permite usarla directamente sin tener que escribir el nombre completo del paquete cada vez.

Estas modificaciones, hechas de manera independiente, permitirán que la clase Greet compile y ejecute correctamente. Aquí tienes un resumen de las respuestas correctas:

52. 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.

Instanciación de Cougar:

- Al crear una instancia de Cougar con new Cougar(), primero se ejecuta el constructor de la superclase Feline, que imprime "feline ".
- Luego se ejecuta el constructor de Cougar, que imprime "cougar "

Método go:

- Dentro del método go, type es asignado a "c". Esto afecta al atributo type de la instancia de Cougar porque type es una variable de instancia que se hereda de Feline.
- this.type ahora es "c", ya que type se ha modificado en la instancia de Cougar.
- super.type también es "c" porque super.type y this.type se refieren al mismo atributo en la instancia de Cougar. La asignación a this.type también modifica super.type ya que ambos apuntan al mismo atributo.

Impresión en go:

- El método go imprime this.type + super.type. Dado que ambos son "c", el resultado es "c c".

53. What is the result?

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

public class Camel implements Rideable {
   int weight = 2;
   String getGait() {
      return mph + ", lope"; // Error de compilación: mph no está definido
   }

   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.

Error de compilación: El método getGait() en la clase Camel intenta usar una variable mph que no está definida en la clase, lo que causará un error de compilación.

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

```
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;
```

Opción a: La segunda línea es válida, pero la primera línea tiene un error de sintaxis.

Opción b: La inicialización de array2D es válida, pero la declaración de array3D es incorrecta.

Opción c: La primera línea tiene un error de tipo; la segunda línea es válida.

Opción d: La asignación es incorrecta debido a tipos incompatibles.

Por lo tanto, las inicializaciones válidas serían:

Opción b (para array2D y la declaración de array3D corregida a int[][][] array3D sería válida)

Opción c (si se corrige la declaración de array3D a int[][] array2D)