# INSTRUCCIONES

Lee detenidamente las siguientes instrucciones, antes de abordar la realización de la prueba, con el objetivo de tener claro los aspectos a valorar y/o lo demandado en la misma.

1. Puede haber preguntas con más de una posible respuesta.
2. Indica la solución que creas correcta de cada pregunta en la tabla del final.
3. Por supuesto PREGUNTA cualquier duda o aclaración que necesites.
4. TIEMPO ESTIMADO: 60 min

**A rellenar por el alumno:**

**Nombre:**

**Apellidos:**

**Fecha:**

**A. Preguntas tipo test (elige la opción correcta)**

1. Sobre **tipos primitivos vs referencia**, indica lo correcto:  
   A) Los primitivos siempre son 0.  
   B) Los de referencia pueden ser null; los primitivos no.  
   C) Los de referencia tienen tamaño fijo; los primitivos no.  
   D) Ambos pueden ser null.
2. ¿Cuándo es más apropiado switch frente a if-else?  
   A) Cuando cada condición evalúa expresiones complejas no relacionadas.  
   B) Cuando todos los valores dependen de la misma variable.  
   C) Cuando hay que evaluar rangos numéricos.  
   D) Nunca, if-else es más legible.
3. Sobre **constructores** y encadenamiento:  
   A) this(...) debe ser la última instrucción del constructor.  
   B) super(...) y this(...) pueden llamarse en el mismo constructor.  
   C) Si no llamas a super(...), el compilador inserta super() implícito.  
   D) Un constructor puede devolver void.
4. La palabra clave **new**:  
   A) Reserva memoria, invoca el constructor y devuelve una **referencia**.  
   B) Crea siempre objetos inmutables.  
   C) Devuelve una copia por valor del objeto.  
   D) Sólo puede usarse con tipos primitivos.
5. **Métodos estáticos vs de instancia**:  
   A) Un estático puede acceder directamente a campos de instancia.  
   B) Un método de instancia puede acceder a miembros estáticos.  
   C) Los estáticos pueden ser sobrescritos.  
   D) Los de instancia pertenecen a la clase, no al objeto.
6. **Sobrecarga** y **sobrescritura**:  
   A) Sobrecargar exige mismo nombre y mismos parámetros.  
   B) Sobrescribir admite cambiar el tipo de retorno covariante.  
   C) Sobrescribir permite reducir la visibilidad (p. ej., de public a private).  
   D) Sobrecargar obliga a usar @Override.
7. En Java, la **herencia múltiple** de implementación:  
   A) Se permite con extends varias clases.  
   B) No existe; se logra vía múltiples interfaces.  
   C) No existe bajo ninguna forma.  
   D) Se permite sólo con final.
8. Dado String s = "Hola mundo"; ¿qué es cierto?  
   A) s.replace("o","O") modifica el objeto directamente.  
   B) s.substring(0,4) devuelve "Hola".  
   C) s.indexOf("z") lanza excepción.  
   D) s.toUpperCase() siempre cambia s.
9. Paso de parámetros en Java:  
   A) Siempre por referencia.  
   B) Siempre por valor (copia del **valor**).  
   C) Primitivos por valor, objetos por referencia.  
   D) Depende de la JVM.
10. **Métodos abstractos**:  
    A) Deben declararse en clases abstractas o interfaces.  
    B) Pueden tener cuerpo vacío {}.  
    C) Pueden ser private.  
    D) Pueden ser static.
11. **Clases abstractas**:  
    A) No pueden tener métodos concretos (con código).  
    B) Pueden tener constructores.  
    C) Se pueden instanciar directamente.  
    D) Deben marcarse final.
12. **¿Cuál es la diferencia principal entre una interfaz y una clase abstracta en Java?**

A) Una interfaz puede tener métodos concretos y abstractos, mientras que una clase abstracta solo puede tener métodos abstractos.

B) Una interfaz no puede tener variables de instancia, mientras que una clase abstracta puede tener variables de instancia.

C) Una interfaz puede ser instanciada directamente, mientras que una clase abstracta no puede ser instanciada directamente.

D) Una clase abstracta puede ser implementada por múltiples clases, mientras que una interfaz solo puede ser implementada por una clase.

1. Sobre **interfaces**:  
   A) Una clase sólo puede implementar una interfaz.  
   B) Una interfaz puede extender varias interfaces.  
   C) Los campos de una interfaz no son final.  
   D) Los métodos de interfaz son private por defecto.
2. **Encapsulación** con getters/setters:  
   A) Permite validar y proteger la invariancia antes de asignar.  
   B) Impide cualquier cambio de estado.  
   C) Es equivalente a usar campos public.  
   D) No aplica a clases inmutables.
3. **¿Cuál es la diferencia principal entre las interfaces List y Set en Java?**

A) List permite elementos duplicados y mantiene el orden de inserción, mientras que Set no permite elementos duplicados y no garantiza el orden de inserción.

B) List no permite elementos duplicados y no mantiene el orden de inserción, mientras que Set permite elementos duplicados y mantiene el orden de inserción.

C) List y Set permiten elementos duplicados, pero solo List mantiene el orden de inserción.

D) List y Set no permiten elementos duplicados, pero solo Set mantiene el orden de inserción.

**Pregunta 1:**

**Considerando el siguiente código, ¿cuál será la salida y por qué?**

class Animal {

    void comer() {

        System.out.println("Animal comiendo");

    }

}

class AnimalCarnivoro {

    void comer () {

        System.out.println("Animal comiendo carne");

    }

    void cazar() {

        System.out.println("Animal carnívoro cazando");

    }

}

class Lobo extends AnimalCarnivoro {

    void comer () {

        System.out.println("Animal comiendo carne en manada");

    }

}

public class Test {

    public static void main(String[] args) {

        Animal a = new Lobo ();

        if (a instanceof AnimalCarnivoro) {

            Lobo l = (Lobo) a;

            l.comer();

            l.cazar();

        } else {

            a.comer();

        }

    }

}

A) Error de compilación en l.cazar()  
B) Animal comiendo carne en manada  
Animal carnívoro cazando

C) se produce la excepción ClassCastException en la línea Lobo l = (Lobo) a;

D) Animal comiendo carne   
Animal carnívoro cazando

**Pregunta 2:**

**Considere el siguiente código. ¿Cuál será la salida de la ejecución del código?**

public class Test {

    public static void main(String[] args) {

        int a = 5;

        modifyPrimitive(a);

        System.out.println("a = " + a);

        MyObject obj = new MyObject();

        obj.value = 5;

        modifyObject(obj);

        System.out.println("obj.value = " + obj.value);

    }

    public static void modifyPrimitive(int x) {

        x = 10;

    }

    public static void modifyObject(MyObject o) {

        o.value = 10;

    }

}

class MyObject {

    int value;

}

A) a = 5  
obj.value = 5

B) a = 5  
obj.value = 10

C) a = 10  
obj.value = 5

D) a = 10  
obj.value = 10

**Pregunta 3:**

**Qué tres palabras reservadas irían en los puntos donde aparece \_\_\_\_\_\_\_\_\_\_ (por orden de aparición)**

\_\_\_\_\_\_\_ class Animal {

    abstract void makeSound();

    void sleep() {

        System.out.println("Sleeping...");

    }

}

class Dog \_\_\_\_\_\_\_\_ Animal {

    void makeSound() {

        System.out.println("Bark");

    }

}

public class Test {

    public static \_\_\_\_\_\_\_\_ main(String[] args) {

        Animal myDog = new Dog();

        myDog.makeSound();

        myDog.sleep();

    }

}

A) public, implements, String

B) interface, implements, void

C) abstract, extends, void

D) \*vacio, \*vacio, String

\*vacio=no poner nada

**Pregunta 4:**

**Qué tres palabras reservadas irían en los puntos donde aparece \_\_\_\_\_\_\_\_\_\_ (por orden de aparición)**

interface Animal {

    void \_\_\_\_\_\_\_\_\_\_();

    default void sleep() {

        System.out.println("Sleeping...");

    }

}

\_\_\_\_\_\_\_\_\_ Dog implements Animal {

    public void makeSound() {

        System.out.println("Bark");

    }

    public void eat() {

        System.out.println("eating");

    }

}

public class Test {

    public static void main(String[] args) {

        Animal myDog = new \_\_\_\_\_\_\_();

        myDog.makeSound();

        myDog.sleep();

    }

}

A) makeSound, class, Dog()

B) makeSound, class, Animal()

C) eat, class, Dog()

D) eat, public, Animal()

**Pregunta 5:**

Considere el siguiente código. **¿Cuál será la salida de la ejecución del código?**

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

class Persona {

    private String dni;

    private String nombre;

    private int edad;

    public Persona(String dni, String nombre, int edad) {

        this.dni = dni;

        this.nombre = nombre;

        this.edad = edad;

    }

    // Getters

    public String getDni() {

        return dni;

    }

    public String getNombre() {

        return nombre;

    }

    public int getEdad() {

        return edad;

    }

}

public class Test {

    public static void main(String[] args) {

        Set<Persona> personas = new HashSet<>();

        // Añadir personas al Set

Persona p = new Persona("12345678A", "Alice", 30);

        personas.add(p);

        personas.add(new Persona("87654321B", "Bob", 25));

        personas.add(new Persona("12345678A", "Carlos", 40)); // DNI duplicado

personas.add(p);

        // Imprimir el contenido del Set

        for (Persona persona : personas) {

            System.out.println(persona.getDni());

        }

    }

}

A) 12345678, 87654321B, 12345678A

B) 12345678, 87654321B

C) 12345678, 87654321B, 12345678A, null

D) Hay un error de compilación en la línea:

Persona p = new Persona("12345678A", "Alice", 30);

|  |  |  |  |
| --- | --- | --- | --- |
| Pregunta | Respuesta | Pregunta | Respuesta |
| Pregunta 1 |  | Pregunta 11 |  |
| Pregunta 2 |  | Pregunta 12 |  |
| Pregunta 3 |  | Pregunta 13 |  |
| Pregunta 4 |  | Pregunta 14 |  |
| Pregunta 5 |  | Pregunta 15 |  |
| Pregunta 6 |  | Pregunta 16 |  |
| Pregunta 7 |  | Pregunta 17 |  |
| Pregunta 8 |  | Pregunta 18 |  |
| Pregunta 9 |  | Pregunta 19 |  |
| Pregunta 10 |  | Pregunta 20 |  |