Objetivo de la tarea

Aplicar conceptos fundamentales de la Programación Orientada a Objetos (POO) en Java, incluyendo herencia entre paquetes, polimorfismo, uso de modificadores de acceso, sobreescritura de métodos, buenas prácticas de organización de código y el manejo de colecciones, mediante análisis, preguntas de reflexión y prácticas con entrada de datos.

1. ¿Qué es el polimorfismo?

El polimorfismo, en programación orientada a objetos, se refiere a la capacidad de un objeto para tomar muchas formas. Esto significa que un método puede comportarse de manera diferente según el objeto específico que lo invoque, incluso si esos objetos comparten una interfaz común o provienen de una jerarquía de herencia. En esencia, permite que un solo nombre de método sea utilizado para realizar acciones que son lógicamente similares, pero que requieren implementaciones distintas dependiendo del tipo de objeto.

Existen varios tipos de polimorfismo, incluyendo el polimorfismo de sobrecarga (donde múltiples métodos con el mismo nombre pero diferentes parámetros existen en la misma clase), y el polimorfismo de inclusión o de subtipo (donde una subclase puede ser tratada como una instancia de su superclase, permitiendo que un método definido en la superclase se comporte de manera especializada en las subclases). Esta característica fundamental promueve la flexibilidad, la reusabilidad del código y la extensibilidad en el diseño de software.

3. Sobreescritura de métodos con @Override

La subclase puede redefinir un método heredado para modificar su comportamiento.?

La sobreescritura de métodos es un concepto fundamental en la programación orientada a objetos donde una subclase redefine un método que ha heredado de su superclase. Esto permite que la subclase modifique o especialice el comportamiento de ese método para adaptarlo a sus propias necesidades, sin alterar la firma (nombre, tipo de retorno y parámetros) del método original. De esta manera, cuando se llama a un método sobreescrito en un objeto de la subclase, se ejecuta la implementación específica de la subclase.

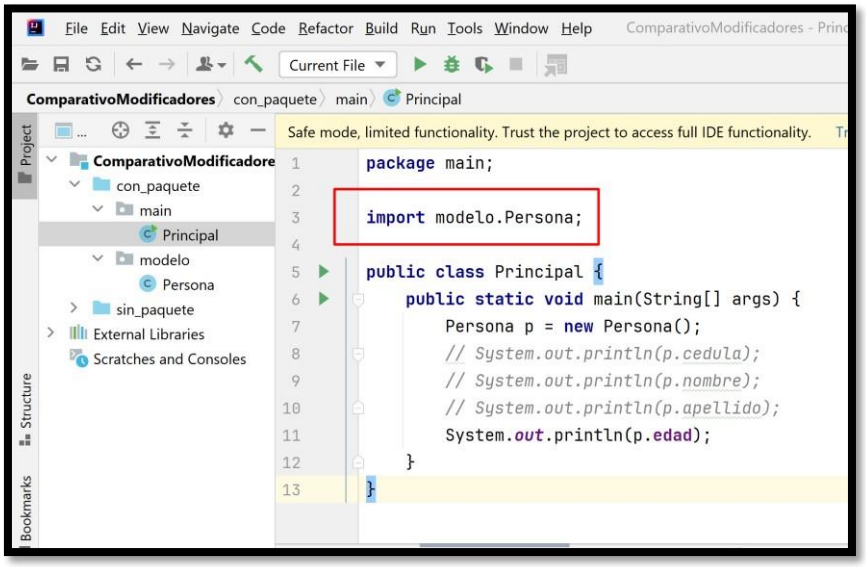
La anotación @Override, común en lenguajes como Java, es crucial para la sobreescritura. No solo mejora la legibilidad del código al indicar claramente que un método está redefiniendo uno heredado, sino que también proporciona una verificación en tiempo de compilación. Esto significa que el compilador asegura que el método realmente sobreescribe uno existente en la superclase, evitando errores comunes como errores tipográficos en el nombre o los parámetros, y facilitando así el mantenimiento y la robustez del software.

4. Por que se pone import modelo.Persona;

Porque estamos importando la clase Persona del paquete modelo.

5. Que sucede si omite import modelo.Persona;

No podría acceder a los métodos, constructores, clases hijas, ni nada que esté vinculado con la clase Persona del paquete modelo.

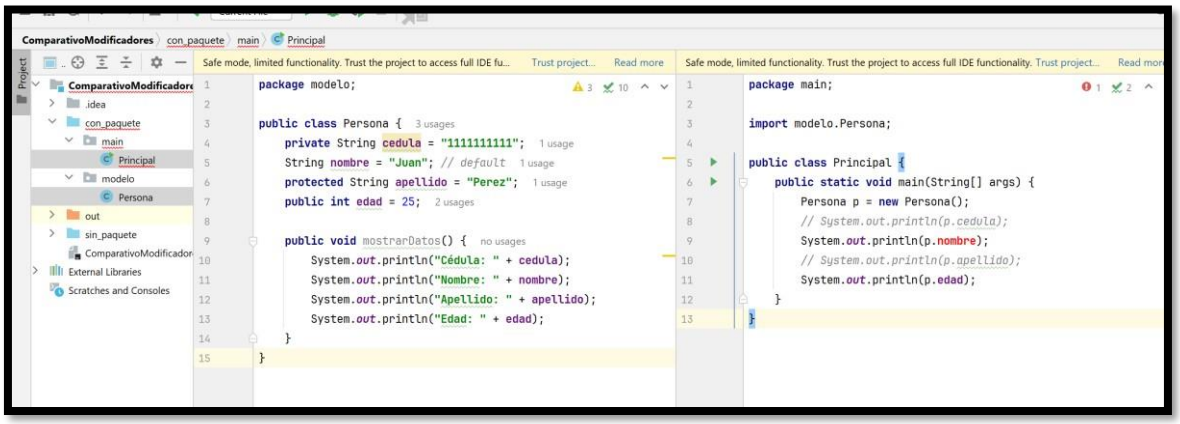


6. Analice que está pasando en el código – Que acción se desea realizar, explicar con detalle si tiene algún problema y que solución se debe aplicar

Se ve que el sistema está creando un nuevo objeto Persona y se llama p, también estamos intentando imprimir la edad de este objeto, en la creación del objeto no se agregaron parámetros para el constructor, por lo que no habría aparentemente una edad, a menos que en el constructor se defina sin parámetros y nos refleje alguna edad como dato quemado, lo cual no es lo mejor. Para solucionarlo tendríamos que hacer un constructor con parámetros para enviar desde la instanciación los datos para estos parámetros, y así tener un objeto bien creado.

7. Revisar si en la captura dispone de herencia en esta practica

No, no dispone de herencia, no se extiende con “extend” ni se está llamando con “super” lo que quiere decir que no estamos usando herencia en ninguna parte de este fragmento de código.



---

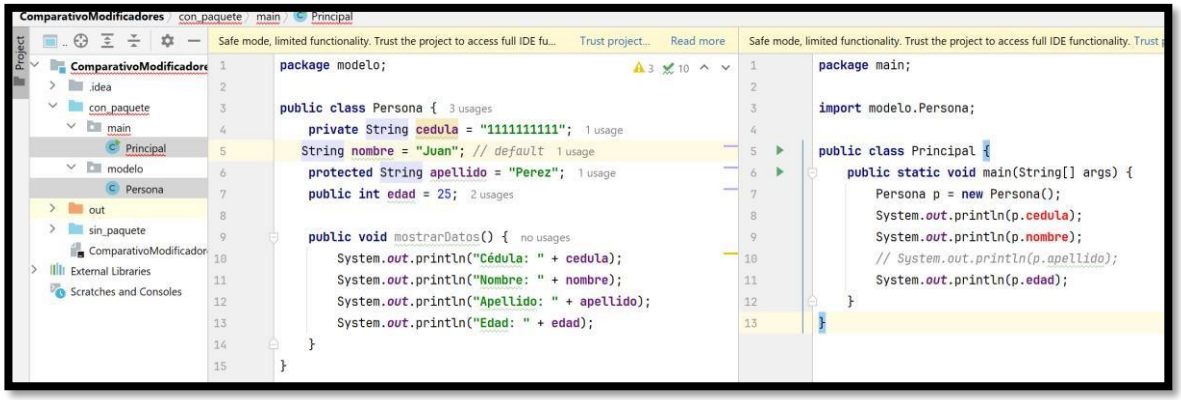
8 . Cuantos errores hay en esta practica y cuales son, indique cuales serían las soluciones?

Error 1: Acceso a miembros privados (cedula) desde otra clase.

Solución: Crear getters y setters para ese atributo privado.

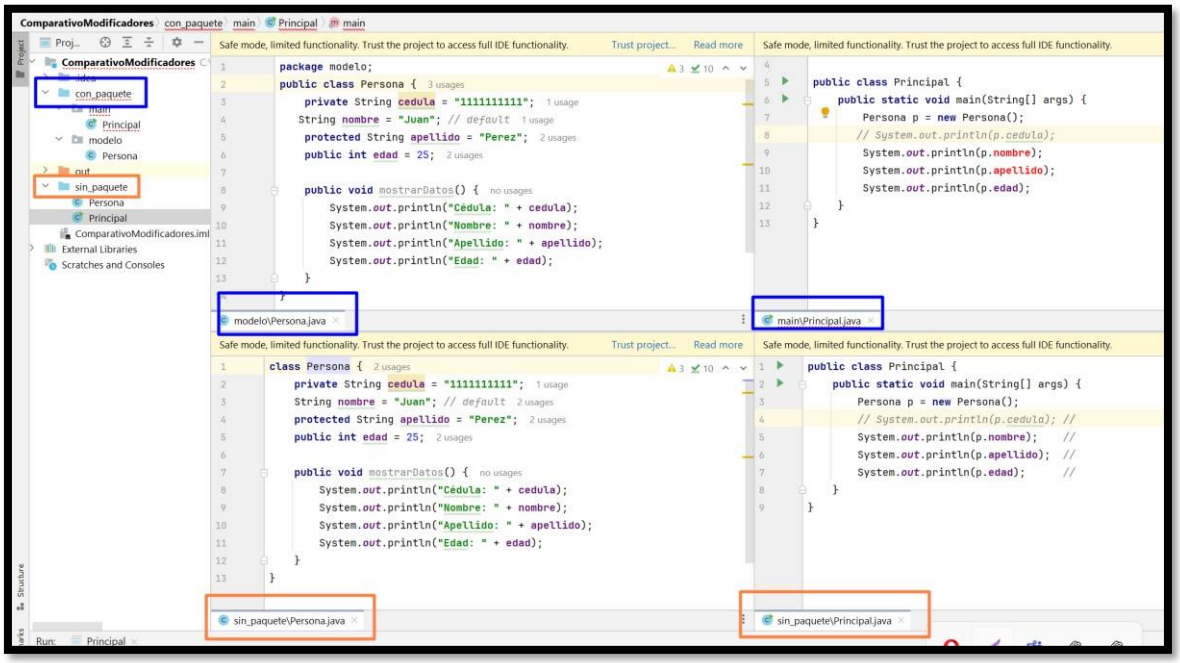
Error 2: Acceso a miembro con visibilidad por defecto (package-private) desde otro paquete.

Solución: Crear un método getter público para nombre en la clase Persona.



9. Si están las mismas líneas de programación en las dos practicas porque existe errores- Analice y explique la solución

Porque la una tiene paquetes separados y la otra dentro de los mismos paquetes.



10. **¿Qué ocurre si un método sobrescrito no usa @Override?**

- El método sí se sobrescribirá si la firma es correcta, pero **no se validará en tiempo de compilación**.
- **Riesgo**: si hay un error en la firma, Java **creará un método nuevo** en vez de sobrescribir.
- Por eso se recomienda **usar siempre @Override**, ya que ayuda al compilador a validar correctamente la intención de sobreescritura.

11. **¿Qué significa "separar por responsabilidad y paquetes"?**

Organizar el código según la función de cada clase.

12. Una clase **abstracta** es aquella que no puede *instanciar*

Aplique un ejemplo

```
// Archivo: FiguraGeometrica.java

package com.ejemplo.formas; // Opcional: Paquete donde se ubicará la clase


public abstract class FiguraGeometrica {


   protected String nombre; // Atributo común a todas las figuras


   public FiguraGeometrica(String nombre) {

      this.nombre = nombre;

   }


   // Método abstracto: No tiene implementación en la clase abstracta.

   // Las subclases concretas DEBEN proporcionar su propia implementación.

   public abstract double calcularArea();


   // Método abstracto: No tiene implementación en la clase abstracta.
```

```
    // Las subclases concretas DEBEN proporcionar su propia implementación.
    public abstract double calcularPerimetro();

    // Método concreto: Tiene implementación en la clase abstracta.
    // Las subclases pueden heredarlo y usarlo directamente, o sobreescribirlo.
    public void mostrarInformacion() {
        System.out.println("Soy una figura geométrica llamada: " + nombre);
    }

    // Método getter para el nombre (método concreto)
    public String getNombre() {
        return nombre;
    }
}
```

Si – No

13. Existen métodos abstracto? Sí.
14. Usar protected para herencia favorece encapsulamiento controlado? Sí.
15. Aplicar @Override Evita errores de redefinición y que es eso? Sí.

16. ¿Qué ocurre si un atributo private se intenta acceder directamente desde una subclase? No.

17.

Ejercicio de análisis 1: Interpretación de código

Código:

```
public class Persona {
    protected String nombre;
    public void saludar() {
        System.out.println("Hola, soy " + nombre);
    }
}

public class Empleado extends Persona {
    private String cargo = "Cajero";
    public void mostrarCargo() {
        System.out.println("Cargo: " + cargo);
    }
}
```

Pregunta: ¿Qué ocurre si se llama `saludar()` desde una instancia de `Empleado` que nunca definió `nombre`?

Saldría: “Hola, soy null”.

18. ¿Cuál es la diferencia entre @Override y sobrecarga de métodos?

La principal diferencia radica en su propósito y mecanismo: `@Override` se utiliza para la **sobreescritura de métodos** (method overriding), lo que implica que una subclase redefine la implementación de un método *heredado* de su superclase, manteniendo la misma firma (nombre, tipo de retorno y parámetros) y permitiendo el polimorfismo en tiempo de ejecución. Por otro lado, la **sobrecarga de métodos** (method overloading) ocurre dentro de la *misma clase* (o en una subclase añadiendo nuevos métodos sobrecargados) y consiste en definir múltiples métodos con el *mismo nombre* pero con *diferentes listas de parámetros* (cantidad, tipo o orden), lo que permite que una función tenga distintos comportamientos según los argumentos que se le pasen, sin relación de herencia para su distinción.

19. Cual es la opción correcta y porqué

Opcion 1: Porque es la correcta y punto.

Característica	Sobrecarga (Overloading)	Sobreescritura (Overriding)
Requiere herencia	No	Sí
Misma firma	No	Sí
Mismo nombre	Sí	Sí
Aplicación	En misma clase	Entre superclase y subclase

Opcion 2

Característica	Sobrecarga (Overloading)	Sobreescritura (Overriding)
Requiere herencia	SI	NO
Misma firma	SI	NO
Mismo nombre	SI	SI
Aplicación	En misma clase	Entre superclase y subclase

20. Cómo se llama esta acción y se aplica solo en herencia?

Sobrecarga

Ocurre cuando una clase tiene múltiples métodos con el mismo nombre pero diferentes parámetros.

```
class Calculadora {
    public int sumar(int a, int b) {
        return a + b;
    }

    public double sumar(double a, double b) {
        return a + b;
    }

    public int sumar(int a, int b, int c) {
        return a + b + c;
    }
}
```

21. Como se llama esta acción y se aplica solo en herencia
No sé, no hay nada referente a la pregunta.

22. ¿Qué incluye una colección?

En Java, una colección se refiere al Java Collections Framework (JCF), que es un conjunto de interfaces, clases y algoritmos diseñados para almacenar y manipular grupos de objetos de manera eficiente. Este framework incluye interfaces fundamentales como List (para colecciones ordenadas con duplicados), Set (para colecciones sin duplicados), Queue (para elementos que esperan procesamiento) y Map (para pares clave-valor), junto con sus respectivas implementaciones concretas como ArrayList, HashSet o HashMap, además de algoritmos comunes para tareas como ordenar o buscar.

23.

Ocurre cuando una subclase redefine un método que ya existe en su superclase.

```java
class Persona {
    public void saludar() {
        System.out.println("Hola desde Persona");
    }
}

class Cliente extends Persona {
    @Override
    public void saludar() {
        super.saludar();
        System.out.println("Hola desde Cliente");
    }
}
```

Sobreescritura (Override)