Laboratorio 3 - Programación Orientada a Objetos

Principios y Prácticas de Desarrollo de Software Orientado a Objetos
grupo A1

Estudiantes:
Daniel Eduardo Cobos Ayala - 2191954
Carlos Ivan Ramirez Diaz - 2172167

Presentado a:
Darío Alejandro Riaño Velandia

Bucaramanga
2023

Contenido

Introducción

En el vasto panorama de los lenguajes de programación, Java se destaca como una opción versátil y sólida para los desarrolladores de todo el mundo. Su naturaleza orientada a objetos ha permitido a los desarrolladores crear aplicaciones complejas y escalables con facilidad. En el centro del paradigma orientado a objetos de Java se encuentra un poderoso concepto conocido como "herencia".

La herencia es un componente fundamental de la programación orientada a objetos, que permite a los desarrolladores crear nuevas clases basadas en las existentes. Este concepto permite la creación de una jerarquía de clases, donde las nuevas clases heredan los atributos y comportamientos de sus clases principales y al mismo tiempo mantienen la flexibilidad para ampliarlos o anularlos.

En este laboratorio, exploramos las complejidades de la herencia en Java. Profundizaremos en su importancia, comprenderemos cómo simplifica la organización del código y seremos testigos de su aplicación en el mundo real a través de un proyecto pequeño pero ilustrativo.

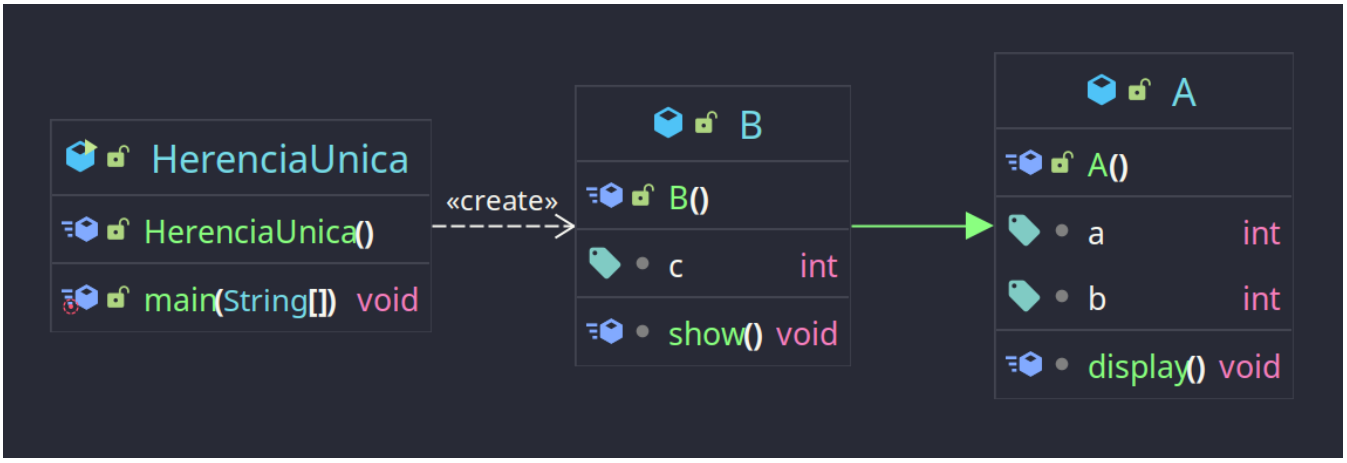
Ejercicios

- **¿Qué es la herencia en Java?**
  En Java, Herencia significa crear nuevas clases basadas en las existentes. Una clase que hereda de otra clase puede reutilizar los métodos y campos de esa clase. Además, también puede agregar nuevos campos y métodos a su clase actual.

- **¿Por qué necesitamos la herencia de Java?**
  - **Reutilización de código:** El código escrito en la superclase es común a todas las subclases. Las clases secundarias pueden usar directamente el código de clase principal.
  - **Anulación de métodos:** la anulación de métodos solo se puede lograr a través de la herencia. Es una de las formas en que Java logra el polimorfismo de tiempo de ejecución.
  - **Abstracción:** El concepto de abstracto donde no tenemos que aportar todos los detalles se consigue a través de la herencia. La abstracción solo muestra la funcionalidad al usuario.
- **Tipos de herencia en Java**
  - Herencia única.
  - Herencia multinivel.
  - Herencia de múltiples rutas.
  - Herencia jerárquica.
  - Herencia híbrida.

Ejercicio 1 - Herencia única

La clase principal da lugar a una sola clase secundaria. Los atributos en este tipo de herencia solo descienden de una clase padre, como máximo. La reutilización del código y la implementación de nuevas características se facilitan porque los atributos descienden de una sola clase base.



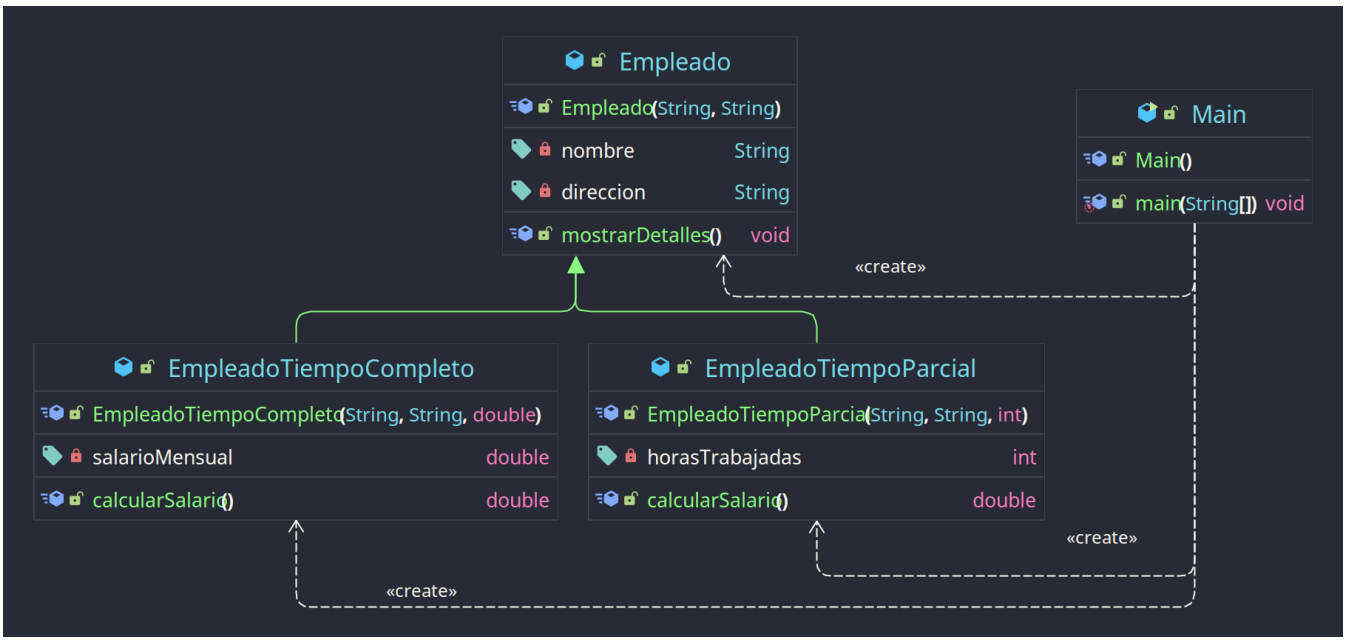
```java
package Lab3.Ej1;


new *
public class HerenciaUnica {
    new *
    public static void main(String args[])
    {
        B obj = new B();
        obj.a=10;
        obj.b=20;
        obj.c=30;
        obj.display();
        obj.show();
    }
}
```

```java
package Lab3.Ej1;


1 inheritor  new *
public class A  1 usage
{
    int a, b;  3 usages
    new *
    void display()  1 usage
    {
        System.out.println("Inside class A values ="+a+" "+b);
    }
}
```

```java
package Lab3.Ej1;


new *
public class B extends A  2 usages
{
    int c;  2 usages
    new *
    void show()  1 usage
    {
        System.out.println("Inside Class B values="+a+" "+b+" "+c);  }
}
```

Ejercicio 2 - Herencia multinivel

Se refiere a una cadena de herencia en la que una clase hija hereda de una clase padre, y luego otra clase hija hereda de la primera clase hija, creando una jerarquía en niveles. Esto permite que las clases hijas hereden propiedades y comportamientos de múltiples niveles hacia arriba en la jerarquía. Aquí tienes un ejemplo:



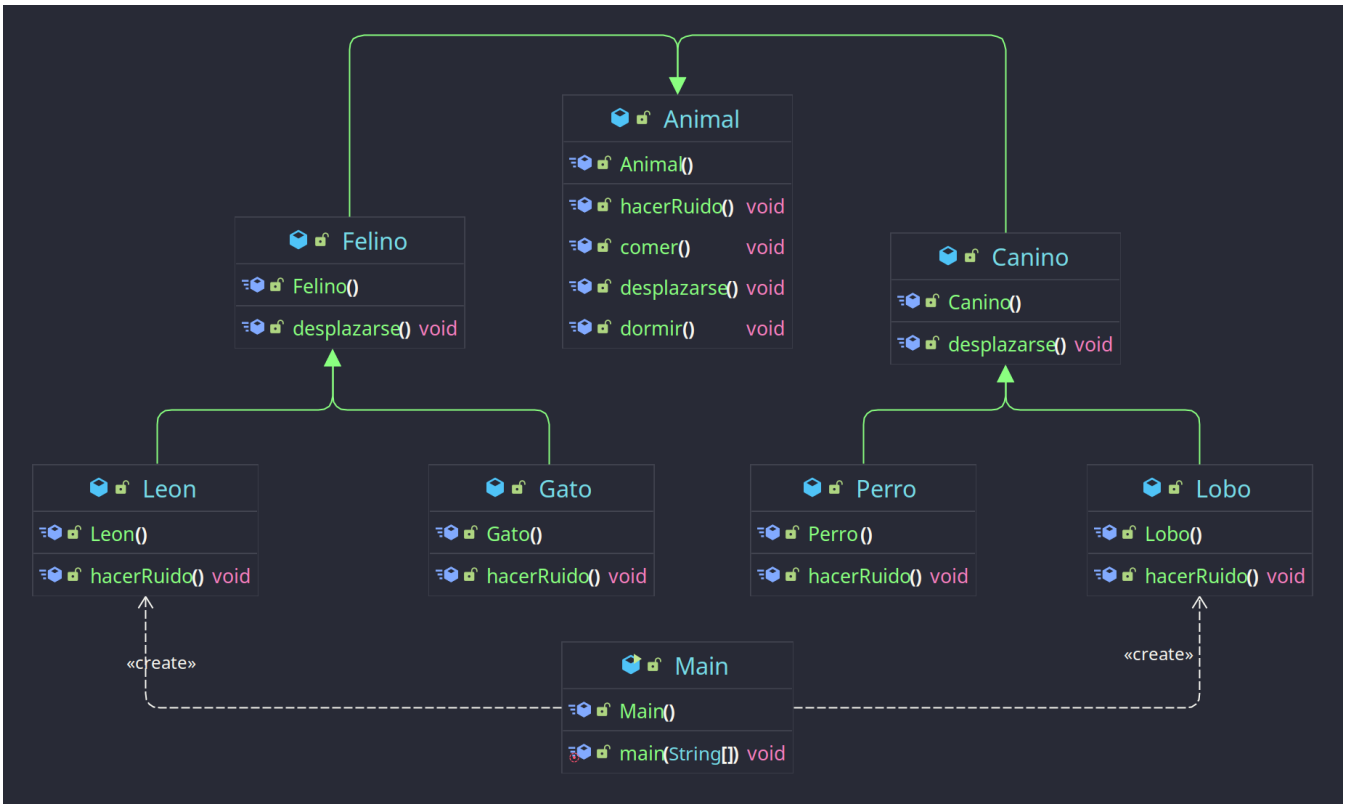

```java
package Lab3.Ej2;

public class HerenciaMultiple {
    public static void main(String[] args) {
        Empleado empleado1 = new Empleado( nombre: "Juan", direccion: "Calle A");
        Empleado empleado2 = new EmpleadoTiempoCompleto( nombre: "Ana", direccion: "Calle B", salarioMensual: 5000);
        Empleado empleado3 = new EmpleadoTiempoParcial( nombre: "Pedro", direccion: "Calle C", horasTrabajadas: 20);

        Empleado[] empleados = {empleado1, empleado2, empleado3};

        for (Empleado empleado : empleados) {
            empleado.mostrarDetalles();

            if (empleado instanceof EmpleadoTiempoCompleto) {
                EmpleadoTiempoCompleto empleadoTiempoCompleto = (EmpleadoTiempoCompleto) empleado;
                System.out.println("Salario: $" + empleadoTiempoCompleto.calcularSalario());
            } else if (empleado instanceof EmpleadoTiempoParcial) {
                EmpleadoTiempoParcial empleadoTiempoParcial = (EmpleadoTiempoParcial) empleado;
                System.out.println("Salario: $" + empleadoTiempoParcial.calcularSalario());
            }

            System.out.println();
        }
    }
}
```

```java
package Lab3.Ej2;

2 inheritors
public class Empleado {  8 usages
    private String nombre;  2 usages
    private String direccion;  2 usages

    public Empleado(String nombre, String direccion) {  3 usages
        this.nombre = nombre;
        this.direccion = direccion;
    }

    // Getters y setters para nombre y direccion

    // Método para imprimir detalles del empleado
    public void mostrarDetalles() {  1 usage
        System.out.println("Nombre: " + nombre);
        System.out.println("Dirección: " + direccion);
    }
}
```

```java
package Lab3.Ej2;

public class EmpleadoTiempoCompleto extends Empleado {  4 usages
    private double salarioMensual;  2 usages

    public EmpleadoTiempoCompleto(String nombre, String direccion, double salarioMensual) {  1 usage
        super(nombre, direccion);
        this.salarioMensual = salarioMensual;
    }

    // Getter y setter para salarioMensual

    // Método para calcular el salario
    public double calcularSalario() { return salarioMensual; }
}
```

```java
package Lab3.Ej2;

public class EmpleadoTiempoParcial extends Empleado {  4 usages
    private int horasTrabajadas;  2 usages

    public EmpleadoTiempoParcial(String nombre, String direccion, int horasTrabajadas) {  1 usage
        super(nombre, direccion);
        this.horasTrabajadas = horasTrabajadas;
    }

    // Getter y setter para horasTrabajadas

    // Método para calcular el salario
    public double calcularSalario() { return horasTrabajadas * 10.0; // Supongamos un salario de $10 por hora }
}
```

Ejercicio 3 - Herencia multinivel




```java
package Lab3.Ej3;

Daniel Cobos *
public class HerenciaMultinivel {
    Daniel Cobos
    public static void main(String[] args) {
        System.out.println("Lobo");
        Lobo elLobo = new Lobo();
        elLobo.hacerRuido();
        elLobo.comer();
        elLobo.desplazarse();

        System.out.println();

        System.out.println("Leon");
        Leon elLeon = new Leon();
        elLeon.hacerRuido();
        elLeon.comer();
        elLeon.desplazarse();
    }
}
```

```java
package Lab3.Ej3;

6 inheritors  Daniel Cobos
public class Animal {  2 usages

    Daniel Cobos
    public void comer() { System.out.println("Yumi Yumi"); }

    Daniel Cobos
    public void dormir() { System.out.println("Zzzzzzzzz"); }

    2 overrides  Daniel Cobos
    public void desplazarse() { System.out.println("Caminando por las calles ...."); }

    4 overrides  Daniel Cobos
    public void hacerRuido() { System.out.println("Haciendo ruido ..."); }

}
```

```java
package Lab3.Ej3;

2 inheritors  Daniel Cobos
public class Canino extends Animal {  2 usages
    Daniel Cobos
    public void desplazarse() { System.out.println("Desplazamiento en manada"); }
}
```

```java
package Lab3.Ej3;

2 inheritors  Daniel Cobos
public class Felino extends Animal {  2 usages
    Daniel Cobos
    public void desplazarse() { System.out.println("Desplazamiento solitario"); }
}
```

```java
package Lab3.Ej3;

Daniel Cobos
public class Gato extends Felino{  no usages
    Daniel Cobos
    public void hacerRuido() { System.out.println("Miau Miau"); }
}
```

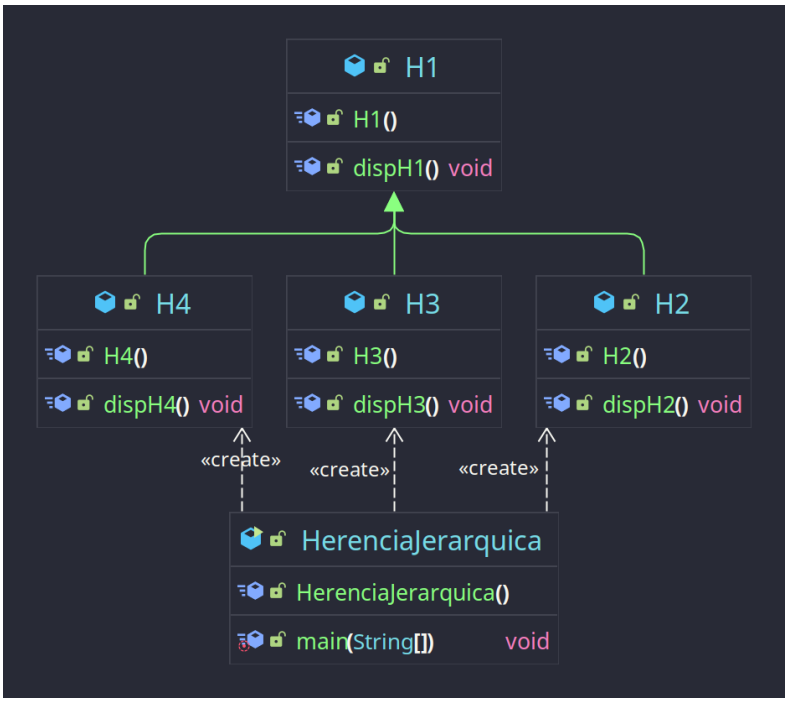
```java
package Lab3.Ej3;

Daniel Cobos
public class Leon extends Felino{  2 usages
    Daniel Cobos
    public void hacerRuido() { System.out.println("Grrrrrrr"); }
}
```

```java
package Lab3.Ej3;

Daniel Cobos
public class Lobo extends Canino{  2 usages
    Daniel Cobos
    public void hacerRuido() { System.out.println("Aauuuuuuu"); }
}
```

```java
package Lab3.Ej3;

Daniel Cobos
public class Perro extends Canino{  no usages
    Daniel Cobos
    public void hacerRuido() { System.out.println("Guau Guau"); }
}
```

Ejercicio 4 - Herencia jerárquica

La herencia jerárquica en Java se refiere a una estructura de herencia en la que varias clases hijas heredan de una única clase padre. Cada clase hija tiene la misma clase padre en la jerarquía. Aquí tienes un ejemplo:



```java
package Lab3.Ej4;

new *
public class HerenciaJerarquica {
    new *
    public static void main(String args[])
    {
        H2 h2 = new H2();
        h2.dispH2();
        h2.dispH1();

        H3 h3 = new H3();
        h3.dispH3();
        h3.dispH1();

        H4 h4 = new H4();
        h4.dispH4();
        h4.dispH1();
    }
}
```

```java
package Lab3.Ej4;

3 inheritors  new *
public class H1 {  3 usages
    new *
    public void dispH1()  3 usages
    {
        System.out.println("Method of ClassH1");
    }
}
```

```java
package Lab3.Ej4;

new *
public class H2 extends H1 {  2 usages
    new *
    public void dispH2() { System.out.println("Method of ClassH2"); }
}
```

```java
package Lab3.Ej4;

new *
public class H3 extends H1 {  2 usages
    new *
    public void dispH3()  1 usage
    {
        System.out.println("Method of ClassH3");
    }
}
```

```java
package Lab3.Ej4;


new *
public class H4 extends H1 {  2 usages
    new *
    public void dispH4() { System.out.println("Method of ClassH4"); }
}
```

Evidencia

Variables Polimórficas: Las variables polimórficas son variables que pueden hacer referencia a objetos de diferentes tipos en una jerarquía de clases. Esto significa que una variable puede apuntar a objetos de clases diferentes en tiempo de ejecución, lo que permite la implementación del polimorfismo. El polimorfismo permite que diferentes objetos respondan de manera diferente a un mismo método, lo que es fundamental en la programación orientada a objetos.

Resultados obtenidos de la semana: Durante la semana de estudio, se ha explorado el concepto de herencia en el contexto de Java, así como su importancia y aplicaciones. La herencia es un componente fundamental de la programación orientada a objetos que permite la creación de nuevas clases basadas en las clases existentes. Esto facilita la reutilización de código, la anulación de métodos y la organización jerárquica de las clases.

Lecciones aprendidas

Durante este período de estudio y práctica, el equipo de trabajo ha adquirido valiosas lecciones que han fortalecido nuestras habilidades en programación orientada a objetos, específicamente en el contexto de Java. Una de las lecciones más significativas ha sido el refinamiento en la creación de diagramas UML (Unified Modeling Language), lo que ha mejorado nuestra capacidad para visualizar y representar las relaciones y estructuras entre las clases en un proyecto de manera más efectiva.

Además, el equipo de trabajo ha profundizado en la comprensión del concepto de herencia en Java. Ahora estamos familiarizados con cómo usar la palabra clave extends para establecer relaciones de herencia entre clases. Esta habilidad nos ha permitido reutilizar el código de clases existentes y extender sus funcionalidades, lo que es fundamental para construir sistemas de software más robustos y eficientes.

El polimorfismo, otro concepto clave, también se ha convertido en un recurso más claro y poderoso en nuestra caja de herramientas de programación. Ahora comprendemos cómo diferentes objetos pueden responder de manera diferente a un mismo método, lo que mejora la flexibilidad y la adaptabilidad de un sistema.

Además, hemos aprendido a utilizar la palabra clave super() para llamar al constructor de la clase padre desde una subclase. Esto es esencial para inicializar adecuadamente los atributos heredados de la superclase en la subclase, lo que garantiza un funcionamiento correcto de la jerarquía de clases.

Por supuesto, durante este proceso de aprendizaje, también se han presentado desafíos y errores. Estos incluyen errores de sintaxis, errores lógicos y problemas específicos relacionados con la herencia. Sin embargo, cada uno de estos desafíos se ha abordado con una revisión cuidadosa del código y ajustes necesarios.

Referencias

[1] *Inheritance (The Java™ Tutorials >Learning the Java Language > Interfaces and Inheritance)*. (n.d.). https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

[2] GeeksforGeeks. (2023). Inheritance in java. *GeeksforGeeks*. https://www.geeksforgeeks.org/inheritance-in-java/

[3] Pankaj. (2022). Inheritance in Java example. *DigitalOcean*. https://www.digitalocean.com/community/tutorials/inheritance-java-example

[4] *Java polymorphism*. (n.d.). https://www.w3schools.com/java/java_polymorphism.asp

[5] GeeksforGeeks. (2023a). Polymorphism in Java. *GeeksforGeeks*. https://www.geeksforgeeks.org/polymorphism-in-java/