	**Carátula para entrega de prácticas**
Facultad de Ingeniería	**Laboratorio de Docencia**

Laboratorio de Computación Salas A y B

Profesor: René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 1

Número de Práctica: 7

Integrante(s): 321573670

No. de Lista: N/A

Semestre: 2025-1

Fecha de Entrega: 04 / Octubre / 2024

Observaciones:

Calificación:

Índice

1. Introducción

- **Planteamiento del problema:**
  Se buscó la realización de ciertos programas con los cuales el alumnado fuera capaz de comprender, aplicar y profundizar en las distintas funcionalidades que implica la herencia de clases en Java, para ello se propuso la realización de los siguientes programas:

  - Un programa en el cual se cree una clase `Figura` con un método llamado `area()` el cual será sobrescrito en dos otras clases: `Circulo` y `Rectangulo` con las fórmulas para calcular sus respectivas áreas.
  - Un programa en el cual se cree una clase `CuentaBanco` para la cual se definan los métodos `retirar()` y `depositar()`, posteriormente se cree una clase `CuentaAhorro` en el cual se sobrescriba el método para realizar retiros de modo que no permita realizarlo si el saldo es menor que $100.
  - Un programa en el cual se defina una clase base `Empleado` de la cual se deriven las clases `Manager`, `Desarrollador` y `Programador`. En cada una de las clases debe de definirse un método para generar un reporte de desempeño, calcular un bono a su salario y mostrar el manejo de proyectos.

- **Motivación:**
  La correcta realización de los programas propuestos conlleva la comprensión de la herencia de clases en Java y el desarrollo de la capacidad para visualizar casos en los que dicha propiedad resulte útil, esto nos permite añadir complejidad a los programas y cimentar los conocimientos previamente analizados durante las clases teóricas.

- **Objetivos:**
  Se espera que el alumnado sea capaz de completar satisfactoriamente todos los programas propuestos haciendo un uso correcto de la herencia de modo que su comprensión y percepción sobre este tema se fortalezca y le permita utilizar esta herramienta de manera correcta.

2. Marco Teórico

Dentro del paradigma de programación orientado a objetos existe algo llamado herencia, este concepto es una parte fundamental del paradigma ya que permite que una clase hija herede atributos y métodos de una clase padre, facilitando la extensión de funcionalidades sin duplicar código [1]. La herencia es el mecanismo mediante el cual una clase puede heredar atributos y métodos de otra clase. Esto permite crear una jerarquía de clases que comparten comportamientos comunes, pero que también pueden tener funcionalidades específicas [2]. En el lenguaje Java, la herencia se establece con la palabra clave `extends`. Una clase puede sobrescribir (override) métodos de la clase base para adaptarlos a sus necesidades específicas.

Además de la herencia, el polimorfismo es otra característica que permite que una clase pueda ser tratada como si fuera de la clase base de la que deriva. Esto facilita el uso de métodos genéricos para manejar diferentes tipos de objetos [2].

Cuando se extiende una clase podemos realizar sobrescritura de métodos, el proceso mediante el cual una clase hija redefine un método de la clase padre con una implementación propia. Esto es útil para ajustar el comportamiento de métodos heredados sin alterar la clase base [3].

3. Desarrollo

- Ejercicio 0:
  Para este ejercicio se definió una clase base `Figura` con un método definido como `area()` para obtener el área de la figura, posteriormente se definieron dos clases hijas de `Figura`: una clase `Circulo` con sus atributos encapsulados y su constructor definido, de la misma clase se sobrescribió el método `area()` con la fórmula definida para el círculo e igualmente se sobrescribió el método `toString()` de modo que imprimiera un mensaje para denotar el área del círculo. Esta misma lógica se repitió con la clase `Rectangulo`, par ala cual nuevamente se encapsularon sus atributos, se definió un método constructor y se sobrescribieron los mismos métodos que en la clase anterior, solo que adecuados a un rectángulo.

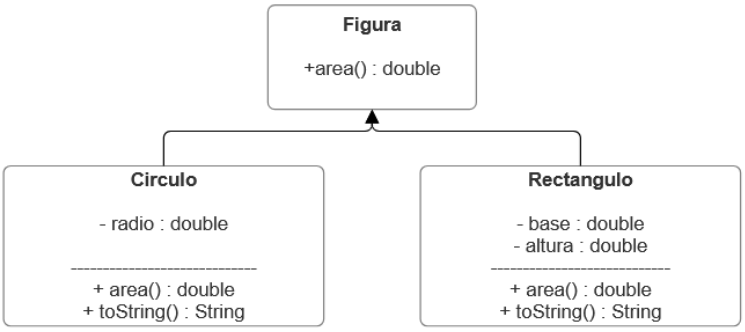


Figura 1: UML de clases de Ejercicio 0.

- Ejercicio 1
  Para este programa se creó una clase base denominada `CuentaBanco` con un constructor, atributos encapsulados, métodos para acceder a dichos atributos y métodos definidos tanto para realizar un retiro como para realizar un depósito. Posteriormente se definió la clase `CuentaAhorro` que extiende a `CuentaBanco`, en dicha clase se definió el constructor reutilizando el de la clase padre y también se sobrescribe el método `retirar()` para que no permita el retiro de dinero a menos que el saldo sea mayor a 100. Finalmente se definió la clase principal en la cual se crearon dos objetos de ambas clases y se realizaron ciertas operaciones.

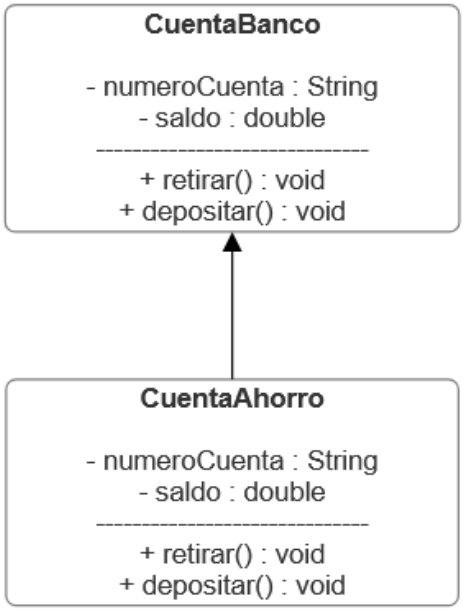


Figura 2: UML de las clases de Ejercicio 1.

- Practica 7:
  En este ejercicio se definió clase `Empleado` con atributos `nombre`, `direccion`, `trabajo` y `salario`, cad auno de estos atributos encapsulados y con sus respectivos métodos para acceder a ellos, haciendo uso de dichos métodos se contruyó un método constructor. Además se definieron los métodos: `calcularBono()` con el cual se calcula el bono del salario del empleado, el nuevo salario y se regresa dicho bono, `generarReporteDesempeño()` con el cual se imprime el desempeño del empleado y el método `manejoProyectos()` con el cual se imprime el proyecto que maeja el empleado. Finalmente se agregó la funcionalidad para imprimir toda la información del empleado.

  Tomando la clase `Empleado` como base se definieron las clases `Manager`, `Desarrollador` y `Programador`, cada una de estas clases con constructores que hacen uso del constructor de `Empleado` y sobrescribiendo los métodos definidos con la finalidad de acomodarse a cada uno de los casos.

  Finalmente se definió un método principal en el que se definieron ins-

tancias para cada una de las clases hijas de `Empleado` haciendo uso de cada método constructor, después se llamó cada uno de los métodos definido para cada clase de modo que se mostrara la información de cada objeto, su desempeño, los projectos relacionados a esta persona y el bono que se le otorgó.

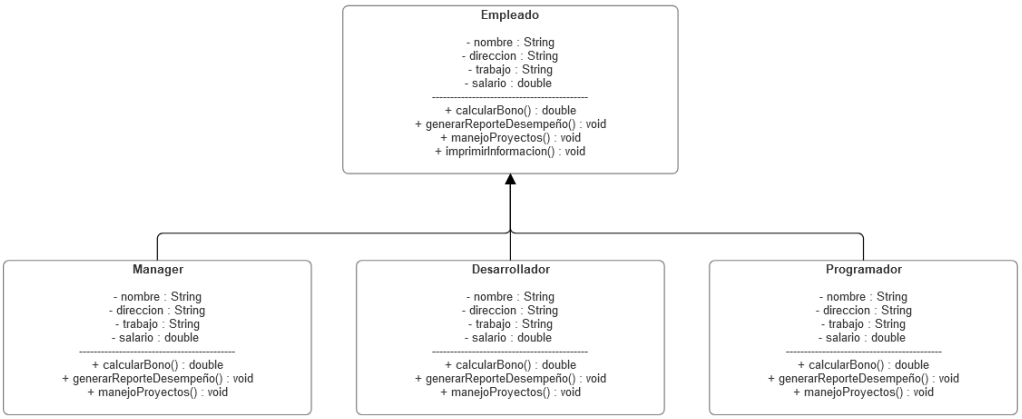


Figura 3: UML para las clases de Practica 7.

4. Resultados

- Ejercicio 0:

  Código:

```java
package mx.unam.fi.poo.g1.p70;

public class Ejercicio0{
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Rectangulo rectangulo = new Rectangulo(base:3.0,altura:10.0);
        System.out.println(rectangulo);

        System.out.println(x:"");

        Circulo circulo = new Circulo(radio:5.0);

        System.out.println(circulo);

    }
}
```

Figura 4: Clase principal en la que se instancia cada una de las clases definidas y se calculan sus áreas.

```java
package mx.unam.fi.poo.g1.p70;

public class Figura {
    public double area(){
        return 0.0;
    }
}
```

Figura 5: Definición de la clase `Figura`

```java
package mx.unam.fi.poo.g1.p70;

public class Circulo extends Figura{
    private double radio;

    public Circulo(double radio){
        setRadio(radio);
    }

    public void setRadio(double radio){
        this.radio = radio;
    }

    public double getRadio(){
        return this.radio;
    }

    @Override
    public double area(){
        return Math.PI * this.radio * this.radio;
    }

    @Override
    public String toString(){
        return "El área del círculo es: " + this.area();
    }
}
```

Figura 6: Definición de la clase `Cicrculo` que extiende a `Figura`, con su respectivo contructor, métodos para acceder a su atributo y la sobrescritura de los métodos para calcular el área y `toString()`.

```java
package mx.unam.fi.poo.g1.p70;

public class Rectangulo extends Figura{
    private double base,altura;

    public Rectangulo(double base, double altura){
        setBase(base);
        setAltura(altura);
    }

    public double getBase(){
        return base;
    }

    public void setBase(double base){
        this.base = base;
    }

    public double getAltura(){
        return altura;
    }

    public void setAltura(double altura){
        this.altura = altura;
    }

    @Override
    public double area(){
        return this.base * this.altura;
    }

    @Override
    public String toString(){
        return "El área del rectángulo es: " + this.area();
    }

}
```

Figura 7: Definición de la clase `Rectángulo` que extiende a la clase `Figura`, se define su constructor, métodos para acceder a sus atributos y la sobrescritura del método para calcular el área y el método `toString()`.

Ejecución:

```
El área del rectángulo es: 30.0

El área del círculo es: 78.53981633974483
```

Figura 8: Ejecución del programa de Ejercicio 0.

- Ejercicio 1:

  Código:

```java
package mx.unam.fi.poo.g1.p71;

public class Ejercicio1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        System.out.println(x:"Se crea un objeto CuentaBanco (C/b No. CB1234)");
        CuentaBanco CB1234 = new CuentaBanco(numeroCuenta:"CB1234",saldo:500.00);
        System.out.println(x:"Se depositan $1000 a la cuenta CB1234");
        CB1234.depositar(cantidad:1000.0);
        System.out.println("Saldo nuevo: "+CB1234.getSaldo());
        System.out.println(x:"Se retiran $600 de la cuenta CB1234");
        CB1234.retirar(cantidad:600.00);
        System.out.println("Saldo nuevo: "+CB1234.getSaldo());
        System.out.println(x:"\nCreando un objeto CuentaAhorro (C/a No. CA1000) con un saldo de $400.");
        CuentaAhorro CA1000 = new CuentaAhorro(numeroCuenta:"CA1000",saldo:400.00);
        System.out.println(x:"Se retiran $300 de la cuenta CA1000");
        CA1000.retirar(cantidad:300.00);
        System.out.println("Saldo nuevo: "+CB1234.getSaldo());
        System.out.println(x:"\n Creando un objeto CuentaAhorro (C/a No. CA1001) con un saldo de $300.");
        CuentaAhorro CA1001 = new CuentaAhorro(numeroCuenta:"CA1001", saldo:300);
        CA1001.retirar(cantidad:250.0);
        System.out.println("Sueldo actual; " + CA1001.getSaldo());

    }
}
```

Figura 9: Definición del método principal en el que se crean instancias de cada una de las clases definidas y se realizan operaciones con los métodos definidos.

```java
package mx.unam.fi.poo.g1.p71;


public class CuentaBanco {

    private String numeroCuenta;
    private double saldo;

    public CuentaBanco(String numeroCuenta, double saldo){
        setNumeroCuenta(numeroCuenta);
        setSaldo(saldo);
    }

    public double getSaldo(){
        return this.saldo;
    }

    public void setSaldo(double saldo){
        this.saldo = saldo;
    }

    public String getNumeroCuenta(){
        return this.numeroCuenta;
    }

    public void setNumeroCuenta(String numeroCuenta){
        this.numeroCuenta = numeroCuenta;
    }
```

Figura 10: Definición de la clase `CuentaBanco` con su contructor y métodos para accesar a los atributos encapsulados.

```java
public class CuentaBanco {
    public void retirar(double cantidad){
            this.saldo -= cantidad;
        } else {
            System.out.println(x:"Fondos insuficientes...");
        }
    }

    public void depositar(double cantidad){
        this.saldo += cantidad;
    }

}
```

Figura 11: Métodos de `CuentaBanco` para realizar retiros y depósitos.

```java
package mx.unam.fi.poo.g1.p71;

public class CuentaAhorro extends CuentaBanco {
    public CuentaAhorro(String numeroCuenta,double saldo){
        super(numeroCuenta,saldo);
    }

    @Override
    public void retirar(double cantidad){
        if(getSaldo() - cantidad < 100){
            System.out.println(x:"Se requiere un saldo de al menos $100");
        } else {
            super.retirar(cantidad);
        }
    }


}
```

Figura 12: Definición de `CuentaAhorro` que extiende a `CuentaBanco` en la cual se define su método constructor haciendo uso del de su clase padre y se sobrescribe el método retirar.

Ejecución:

```
Se crea un objeto CuentaBanco (C/b No. CB1234)
Se depositan $1000 a la cuenta CB1234
Saldo nuevo: 1500.0
Se retiran $600 de la cuenta CB1234
Saldo nuevo: 900.0

Creando un objeto CuentaAhorro (C/a No. CA1000) con un saldo de $400.
Se retiran $300 de la cuenta CA1000
Saldo nuevo: 900.0

 Creando un objeto CuentaAhorro (C/a No. CA1001) con un saldo de $300.
Se requiere un saldo de al menos $100
Sueldo actual; 300.0
```

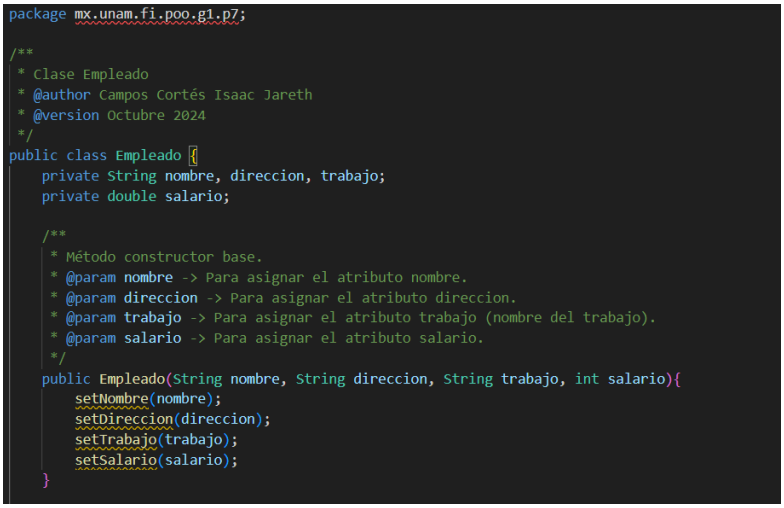
Figura 13: Ejecución del código de Ejercicio 1.

- Practica 7:

  Código:

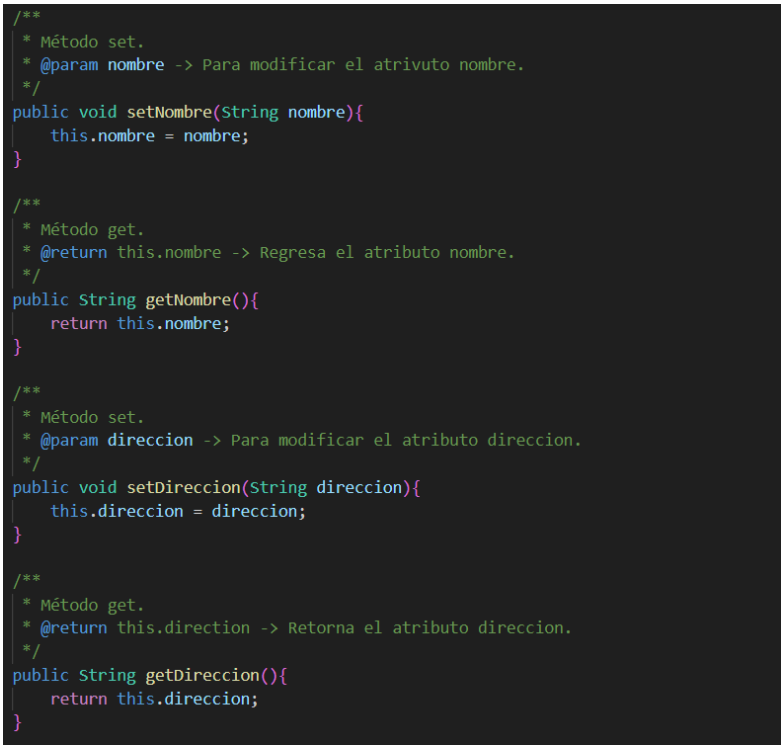
```java
public class Practica7 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Manager manager = new Manager(nombre:"Georgina Pérez", direccion:"Tejamamil 58, CDMX", trabajo:"Web Dev Manager", salario:35000);

        Desarrollador desarrollador = new Desarrollador(nombre:"Juan Velazco", direccion:"Jilotepec 54, Edo. Mex.", trabajo:"Mobile Developer", salario:25000);

        Programador programador = new Programador(nombre:"Federica flores", direccion:"Monte verde 37, CDMX", trabajo:"Junior Programmer", salario:18000);

        System.out.println(x:"A continuación se presenta la información de 3 empleados a quienes se les otorgó un bono a su salario.\n");

        manager.imprimirInformacion();
        manager.generarReporteDesempeño(desempeño:"Excelente");
        manager.manejoProyectos();
        System.out.println("\nEl bono dado a "+manager.getNombre() + " equivalió a: $"+manager.calcularBono());
        System.err.println("Su salario después del bono fue de: $"+manager.getSalario()+"\n");

        desarrollador.imprimirInformacion();
        desarrollador.generarReporteDesempeño(desempeño:"Bueno");
        desarrollador.manejoProyectos();
        System.out.println("\nEl bono dado a "+desarrollador.getNombre() + " equivalió a: $"+desarrollador.calcularBono());
        System.err.println("Su salario después del bono fue de: $"+desarrollador.getSalario()+"\n");

        programador.imprimirInformacion();
        programador.generarReporteDesempeño(desempeño:"Sobresaliente");
        programador.manejoProyectos();
        System.out.println("\nEl bono dado a "+programador.getNombre() + " equivalió a: $"+programador.calcularBono());
        System.err.println("Su salario después del bono fue de: $"+programador.getSalario()+"\n");

    }
}
```

Figura 14: Código de la clase principal en la cual se crean objetos de cada clase definida, se imprime su información, reporte de desepeño, los proyectos en los que se involucra y la bonificación a su salario.



```java
package mx.unam.fi.poo.g1.p7;

/**
 * Clase Empleado
 * @author Campos Cortés Isaac Jareth
 * @version Octubre 2024
 */
public class Empleado {
    private String nombre, direccion, trabajo;
    private double salario;

    /**
     * Método constructor base.
     * @param nombre -> Para asignar el atributo nombre.
     * @param direccion -> Para asignar el atributo direccion.
     * @param trabajo -> Para asignar el atributo trabajo (nombre del trabajo).
     * @param salario -> Para asignar el atributo salario.
     */
    public Empleado(String nombre, String direccion, String trabajo, int salario){
        setNombre(nombre);
        setDireccion(direccion);
        setTrabajo(trabajo);
        setSalario(salario);
    }
```

Figura 15: Se define la clase `Empleado` con sus atributos encapsulados, métodos para acceder a dichos atributos, su método constructor y métodos para imprimir su información, calcular el bono a su salario, imprimir el reporte de desempeño y de manejo de proyectos.



```java
/**
 * Método set.
 * @param nombre -> Para modificar el atrivuto nombre.
 */
public void setNombre(String nombre){
    this.nombre = nombre;
}

/**
 * Método get.
 * @return this.nombre -> Regresa el atributo nombre.
 */
public String getNombre(){
    return this.nombre;
}

/**
 * Método set.
 * @param direccion -> Para modificar el atributo direccion.
 */
public void setDireccion(String direccion){
    this.direccion = direccion;
}

/**
 * Método get.
 * @return this.direction -> Retorna el atributo direccion.
 */
public String getDireccion(){
    return this.direccion;
}
```

Figura 16: Métodos setters y getters para nombre y dirección.

```java
/**
 * Método set.
 * @param trabajo -> Para modificar el atributo trabajo.
 */
public void setTrabajo(String trabajo){
    this.trabajo = trabajo;
}

/**
 * Método get.
 * @return this.trabajo -> Retorna el atributo trabajo.
 */
public String getTrabajo(){
    return this.trabajo;
}

/**
 * Método set.
 * @param salario -> Para modificar el atributo salario.
 */
public void setSalario(double salario){
    this.salario = salario;
}

/**
 * Método get.
 * @return this.salario -> Regresa el atributo salario.
 */
public double getSalario(){
    return this.salario;
}
```

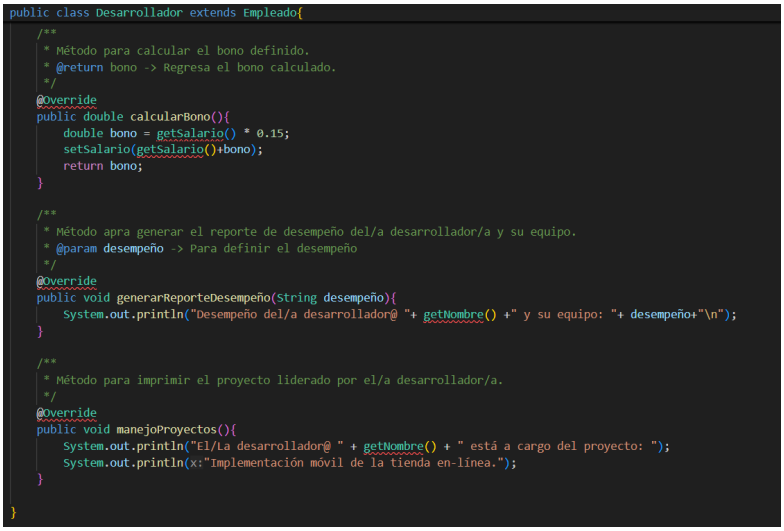
Figura 17: Métodos setters y getters para trabajo y salario.

```java
/**
 * Método para calcular el bono definido.
 * @return bono -> Regresa el bono calculado.
 */
public double calcularBono(){
    double bono = getSalario() * 0.5;
    setSalario(getSalario()+bono);
    return bono;
}

/**
 * Método para generar un reporte de desempeño.
 * @param desempeño -> Definición del desempeño del empleado.
 */
public void generarReporteDesempeño(String desempeño){
    System.out.println("Desempeño: "+ desempeño);
}

/**
 * Método para presentar los proyectos siendo manejados.
 */
public void manejoProyectos(){
    System.out.println("El/La emplead@" + getNombre() + "no está manejando ningún proyecto");
}

public void imprimirInformacion(){
    System.out.println("Nombre: "+getNombre());
    System.out.println("Dirección: "+ getDireccion());
    System.out.println("Puesto: "+getTrabajo());
    System.out.println("Salario: "+getSalario());
}
```

Figura 18: Definición de los métodos `calcularBono()`, `generarReporteDesempeño()`, `manejoProyectos()` y `imprimirInformacion()`.

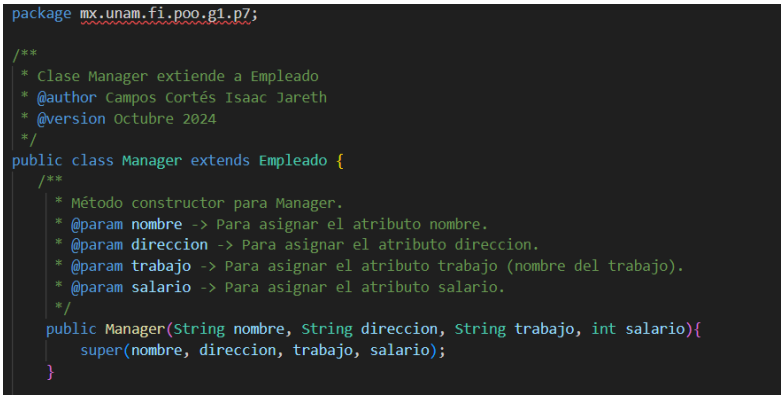
```java
package mx.unam.fi.poo.g1.p7;

/**
 * Clase Desarrollador extiende a Empleado
 * @author Campos Cortés Isaac Jareth
 * @version Octubre 2024
 */
public class Desarrollador extends Empleado{

    /**
     * Método constructor para Desarrollador.
     * @param nombre -> Para asignar el atributo nombre.
     * @param direccion -> Para asignar el atributo direccion.
     * @param trabajo -> Para asignar el atributo trabajo (nombre del trabajo).
     * @param salario -> Para asignar el atributo salario.
     */
    public Desarrollador(String nombre, String direccion, String trabajo, int salario) {
        super(nombre, direccion, trabajo, salario);
    }
```

Figura 19: Código de la clase `Desarrollador` que extiende a `Empleado`, se define su método constructor y se sobrescribe cada uno de los métodos para calcular el bono, generar el reporte y manejar proyectos.



```java
public class Desarrollador extends Empleado{
    /**
     * Método para calcular el bono definido.
     * @return bono -> Regresa el bono calculado.
     */
    @Override
    public double calcularBono(){
        double bono = getSalario() * 0.15;
        setSalario(getSalario()+bono);
        return bono;
    }

    /**
     * Método apra generar el reporte de desempeño del/a desarrollador/a y su equipo.
     * @param desempeño -> Para definir el desempeño
     */
    @Override
    public void generarReporteDesempeño(String desempeño){
        System.out.println("Desempeño del/a desarrollador@ "+ getNombre() +" y su equipo: "+ desempeño+"\n");
    }

    /**
     * Método para imprimir el proyecto liderado por el/a desarrollador/a.
     */
    @Override
    public void manejoProyectos(){
        System.out.println("El/La desarrollador@ " + getNombre() + " está a cargo del proyecto: ");
        System.out.println(x:"Implementación móvil de la tienda en-línea.");
    }

}
```

Figura 20: Sobreescritura de los métodos para el bono, desempeño y manejo de proyectos.



```java
package mx.unam.fi.poo.g1.p7;

/**
 * Clase Manager extiende a Empleado
 * @author Campos Cortés Isaac Jareth
 * @version Octubre 2024
 */
public class Manager extends Empleado {
    /**
     * Método constructor para Manager.
     * @param nombre -> Para asignar el atributo nombre.
     * @param direccion -> Para asignar el atributo direccion.
     * @param trabajo -> Para asignar el atributo trabajo (nombre del trabajo).
     * @param salario -> Para asignar el atributo salario.
     */
    public Manager(String nombre, String direccion, String trabajo, int salario){
        super(nombre, direccion, trabajo, salario);
    }
```

Figura 21: Definición de la clase `Manager` que extiende a la clase `Empleado` de la cual se define su constructor, y la sobrescritura de los mismos métodos que en `Desarrollador`.

Figura 22: Métodos para el bono, desempeño y manejo de proyectos, ajustados para la clase `Manager`

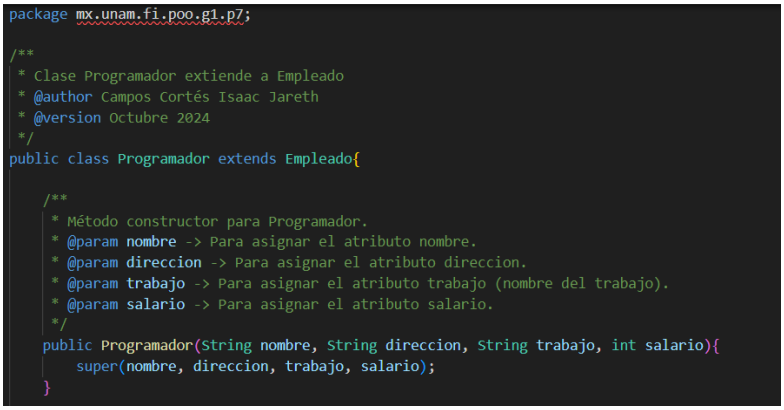


Figura 23: Código para definir la clase `Programador` que extiende la clase `Empleado` con su método constructor y la sobrescritura de los métodos para calcular el bono, realizar el reporte de desempeño y el manejo de proyectos.

```java
/**
 * Método para calcular el bono definido.
 * @return bono -> Regresa el bono calculado.
 */
@Override
public double calcularBono(){
    double bono = getSalario() * 0.10;
    setSalario(getSalario()+bono);
    return bono;
}

/**
 * Método apra generar el/a reporte de desempeño del programador/a.
 * @param desempeño -> Para definir el desempeño
 */
@Override
public void generarReporteDesempeño(String desempeño){
    System.out.println("Desempeño del/a programador/a "+ getNombre() +": "+ desempeño+"\n");
}

/**
 * Método para imprimir el proyecto del que forma parte el/a programador/a.
 */
@Override
public void manejoProyectos(){
    System.out.println("El/La programador@ " + getNombre() + " forma parte del proyecto: ");
    System.out.println(x:"Modernización de las bibliotecas \"legacy\" de la compañía.");
}
```

Figura 24: Sobrescritura de los métodos para el bono, desempeño y manejo de proyectos, modificados para las finalidades de la clase.

Ejecución:

```
A continuación se presenta la información de 3 empleados a quienes se les otorgó un bono a su salario.

Nombre: Georgina Pérez
Dirección: Tejamamil 58, CDMX
Puesto: Web Dev Manager
Salario: 35000.0
Desempeño de los equipos manejados por Georgina Pérez: Excelente

El/La manager Georgina Pérez está manejando los siguientes proyectos:
1)Desarrollo de una página web
2)Reparación del servidor 345-G
3)Restructuración de la bse de datos 4A

El bono dado a Georgina Pérez equivalió a: $7000.0
Su salario después del bono fue de: $42000.0

Nombre: Juan Velazco
Dirección: Jilotepec 54, Edo. Mex.
Puesto: Mobile Developer
Salario: 25000.0
Desempeño del/a desarrollador@ Juan Velazco y su equipo: Bueno

El/La desarrollador@ Juan Velazco está a cargo del proyecto:
Implementación móvil de la tienda en-línea.

El bono dado a Juan Velazco equivalió a: $3750.0
Su salario después del bono fue de: $28750.0
```

Figura 25: Ejecución del código.

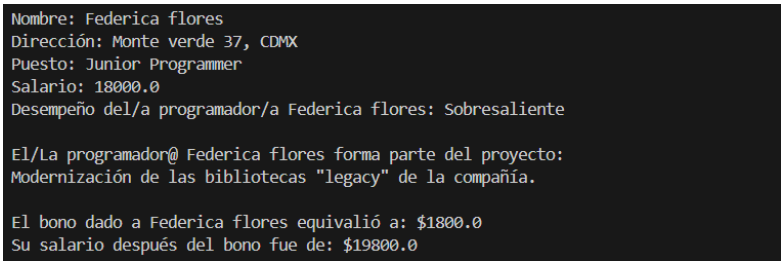


Figura 26: Ejecución del código.

5. Conclusiones

La realización de la práctica nos permitió concretar el conocimiento adquirido sobre la herencia de clases, la sobrescritura de métodos y la manera de implementar estos conceptos en un programa haciendo uso de Java. Además fuimos capaces de relaizar todos los programas de la manera deseada. En pocas palabras, los objetivos se cumplieron de forma exitosa.

Referencias

[1] H. Schildt, *Java: The Complete Reference*, 12th. McGraw-Hill Education, 2022. dirección: `https://www.mhprofessional.com`.

[2] H. D. Paul Deitel, *Java: How to Program*, 11th. Pearson, 2017. dirección: `https://www.pearson.com`.

[3] C. S. Horstmann, "Core Java Volume I–Fundamentals," *Core Java*, vol. I, 2020. dirección: `https://www.pearson.com`.