



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 7

No. de práctica(s): Proyecto 2

Integrante(s):
320065570
425133840
423020252
322229916
425032224

No. de lista o brigada: 3

Semestre: 2026-1

Fecha de entrega: 27/08/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción	2
2. Marco Teórico	3
3. Desarrollo	4
4. Resultados	6
5. Conclusiones	7
6. Datos de trabajo	10

1. Introducción

Esta práctica tiene como objetivo la aplicación de los temas vistos en clase, referentes al empaquetado, encapsulamiento, polimorfismo y herencia en un proyecto funcional. Estas herramientas son prácticamente el santo grial, por decirlo de alguna forma, de la programación orientada a objetos en términos ya profesionales, ya que son las características más utilizadas en la creación de códigos, de ahí la importancia de su correcto manejo.

Se nos presenta para resolver el problema de una compañía, la cual realiza los pagos semanalmente a sus empleados, los cuales reciben dos tipos de remuneraciones; empleados asalariados que cuentan con un sueldo fijo semanalmente, y los empleados por hora, los cuales reciben un sueldo por cada hora trabajada y una remuneración al realizar horas extras. El problema consiste en crear un código que pueda generar estas dos clases de trabajadores, haciendo herencia de una clase general empleado.

A partir de estos puntos realizamos el ejercicio solicitado, sin embargo, contamos con problemas con nuestra aplicación ya que no funcionó de manera óptima, por lo tanto, en el presente escrito nuestro objetivo principal es dar una solución a las fallas que tiene nuestro programa para que sea funcional.

2. Marco Teórico

El primer concepto aplicado en el proyecto es el empaquetado; este nos facilita modularizar nuestro código. En cada módulo de nuestro código debemos usar la palabra reservada *package* para indicar en dónde pertenece [1]. Nos son útiles para agrupar clases y evitar conflictos y miembros (atributos y métodos). Está agrupado en jerarquía de paquetes con la intención de ubicar clases particulares.

El encapsulación lo implementamos para agrupar variables y métodos que operan sobre una clase [2]. Restringe el acceso directo a algunos de los componentes de un objeto y puede impedir cambios accidentales en los datos. Si se emplean modificadores de acceso con el nivel más fuerte de encapsulación (*private*) es necesario hacer uso de métodos de acceso consultores (*get*) y modificadores (*set*) para tener acceso a los atributos privados del objeto.

La herencia, como su nombre lo dice, hace que sea posible heredar métodos y atributos de una clase a otra con la intención de aplicar polimorfismo, extender funcionalidades y reutilizar código. La clase que hereda se le denomina clase padre, mientras que la clase que adquiere los métodos o atributos de la clase padre se le denomina clase hija [3]. Si queremos redefinir el comportamiento de un método en una clase en específico, debemos implementar sobreescritura de métodos y, por buena práctica, marcar antes con la palabra *@override*. Si para aplicar la herencia hacemos uso de una clase abstracta, debemos tener en cuenta los siguientes aspectos; no la podemos instanciar, podemos tener atributos, métodos abstractos y métodos concretos (métodos en los que no se está definido el comportamiento); pero si hacemos uso de métodos abstractos, debemos implementarlos en las clases hijas y especificar como es que deben actuar.

En Java, el polimorfismo es la capacidad que tiene un objeto de contestar diferente e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su llamado [4], en otras palabras, es enviar mensajes sintácticamente iguales a objetos de distintos tipos. Consiste en hacer que un objeto de una clase se comporte como cualquiera de sus subclases.

3. Desarrollo

Comenzamos creando la clase padre abstracta Empleado. Lo primero que hicimos fue agregar el empaquetado por el nombre de “package mx.unam.poo.py2”, definimos los atributos finales privados nombre, apellidoPaterno y numeroSeguro; agregamos el constructor con los parámetros de los atributos y creamos el método abstracto ingresos y, posteriormente, usamos el método toString para mostrar la información.

En la clase hija empleado asalariado de igual forma agregamos el empaquetado, usamos extends para que nuestra clase herede atributos y métodos de la clase padre y definimos el atributo privado salario semanal; usamos super para indicar que queremos adquirir los atributos de la clase Empleado, agregamos el constructor, y empleamos setters y getters para tener acceso a este atributo privado y en el set lanzamos el mensaje que no es posible poner un salario negativo si es que se pone un número menor a 0. Definimos cómo es que va a actuar el método abstracto ingresos con set y empleamos el método toString para mostrar la información, ayudándonos con super.toString() para mostrar los atributos de la clase Empleado y después agregamos los atributos generales de nuestra clase EmpleadoSemanal.

En la otra clase hija EmpleadoPorHoras también se aplicó el empaquetado, usamos extends para indicar que se va a aplicar herencia; definimos los atributos privados salario y horas, usamos super para adquirir los atributos de la clase padre, agregamos el constructor e hicimos uso de los métodos de acceso como en la otra clase hija, y también lanzamos un mensaje en los casos donde se introduce un salario negativo y horas menores a 0 pero mayores a 180; redefinimos el método ingresos y usamos toString usando super.toString() para adquirir ese método de la clase padre y agregamos la información de los atributos particulares de esta clase.

Por último, en la clase MainApp creamos un objeto EmpleadoPorHoras y imprimimos los métodos toString y EmpleadoPorHoras, agregamos otro objeto Empleado por horas e imprimimos el método toString.

Esta secuencia de clases puede ser vista de mejor manera a través del siguiente diagrama.

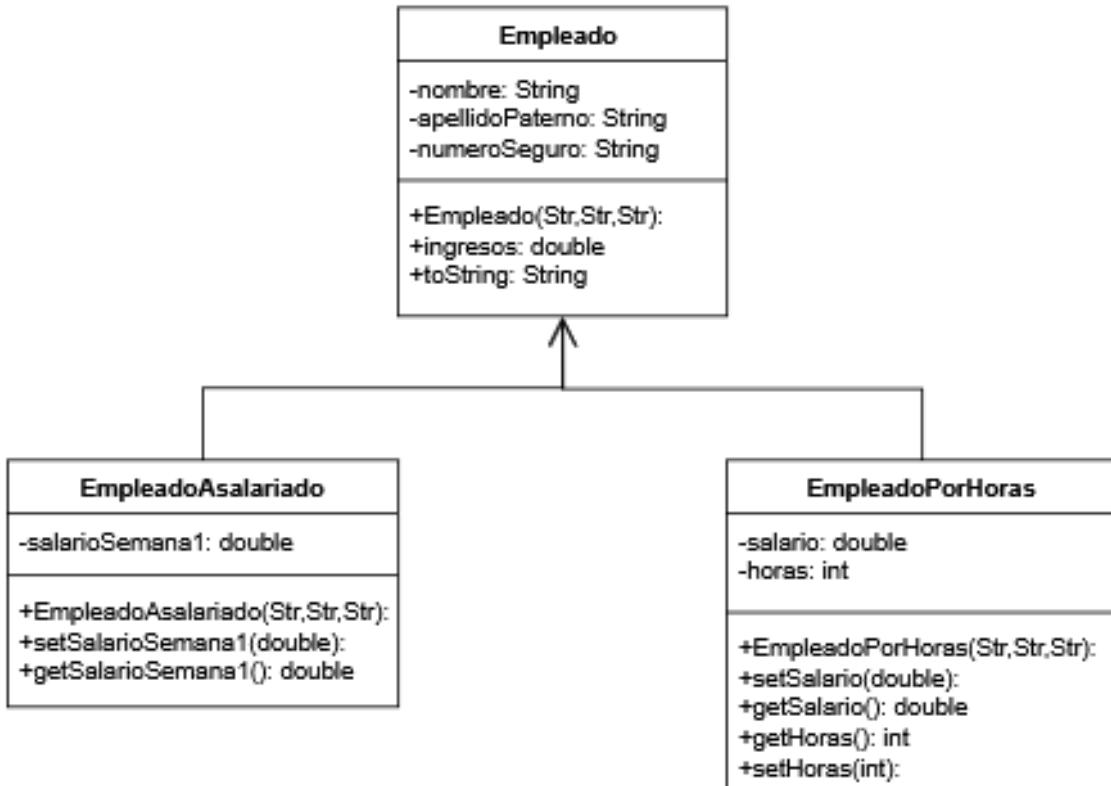


Figura 1: Diagrama de clases

Errores identificados:

Los errores identificados en que impidieron el funcionamiento de nuestra aplicación, como se presentará en la siguiente sección, se debieron a los siguientes inconvenientes:

1. La clase MainApp tiene un nombre de empaquetado distinto a los demás.
2. La clase padre **Empleado** tiene atributos estáticos finales y no se emplearon métodos de acceso a estos atributos.
3. En las clases hijas se tiene el atributo como `numeroSeguroSocial` mientras que en la clase padre se tiene a este como `numeroSeguro`.
4. La clase MainApp crea un objeto **EmpleadoPorHoras** de tipo **EmpleadoPorHoras**, no permitiendo el polimorfismo.

4. Resultados

```
/Users/A31ZP/Code/POO-3-TEAM/Practicas/Proyecto2/mx/unam/fi/poo/py2/EmpleadoPorHoras.java:3: error: cannot find symbol
public class EmpleadoPorHoras extends Empleado {
    ^
symbol: class Empleado
MainApp.java:6: error: cannot access EmpleadoPorHoras
    ^
    EmpleadoPorHoras empleado1 = new EmpleadoPorHoras("Juan", "Perez", "1256789", 45, 50.0);
                                ^
bad source file: /Users/A31ZP/Code/POO-3-TEAM/Practicas/Proyecto2/mx/unam/fi/poo/py2/EmpleadoPorHoras.java
file does not contain class mx.unam.fi.poo.py2.EmpleadoPorHoras
Please remove or make sure it appears in the correct subdirectory of the sourcepath.
/Users/A31ZP/Code/POO-3-TEAM/Practicas/Proyecto2/mx/unam/fi/poo/py2/EmpleadoPorHoras.java:40: error: method does not override or implement a method from a supertype
@Override
^
/Users/A31ZP/Code/POO-3-TEAM/Practicas/Proyecto2/mx/unam/fi/poo/py2/EmpleadoPorHoras.java:55: error: method does not override or implement a method from a supertype
@Override
^
/Users/A31ZP/Code/POO-3-TEAM/Practicas/Proyecto2/mx/unam/fi/poo/py2/EmpleadoPorHoras.java:58: error: cannot find symbol
    "\n" + super.toString() +
    ^
symbol: variable super
location: class EmpleadoPorHoras
5 errors
error: compilation failed
```

Figura 2: Complilación fallida del proyecto

```
---- Empleado Por Horas ----
Juan Solis
numero de seguro social: 0283
Salario por hora: $2500.0
Horas trabajadas: 60
Ingresos totales: $175000.0
---- Empleado Asalariado ---
Ernesto Cruz
numero de seguro social: 9345
Salario semanal: $2500.0
```

Figura 3: Complilación de la solucion propuesta

5. Conclusiones

En la sección de desarrollo se enumeraron los errores presentes en el código realizado inicialmente, errores los cuales se pueden visualizar en la figura 2 presente en la sección de resultados.

Para resolver estos errores, redactamos una solución para el código, además de una serie de propuestas para su mejora.

Solución propuesta y mejoras:

1. Cambiar el empaquetado de la clase MainApp por package mx.unam.poo.py2.
2. Hacer que los atributos de la clase padre solo sean privados y aplicar los métodos de acceso para acceder a estos.

```
private String nombre, apellidoPaterno, numeroSeguroSocial;
```

3. Como mejora podemos hacer que en la clase Empleado semanal en el método `toString` se agregue el método `ingresos()` en vez de `getSalarioSemanal()`
4. Cambiar el nombre del atributo `NumeroSeguro` por `NumeroSeguroSocial`
5. Crear un arreglo de tipo `Empleado` para crear objetos de tipo `Empleado` por horas y `EmpleadoAsalariado` de la siguiente manera:

```
Empleado[] empleado = new Empleado[2];
```

```
empleado[0] = new EmpleadoPorHoras("Juan", "Solis", "0283", 60, 2500.0);  
empleado[1] = new EmpleadoAsalariado("Mario", "Cruz", "9345", 2500.0);
```

e imprimir solo el método `toString`.

6. A manera de implementación, en la clase `MainApp` sería buena idea realizar un menú, que permita al usuario agregar y eliminar trabajadores por su tipo de salario, buscar y mostrar un trabajador en específico, así como ver y eliminar la lista de trabajadores; este sistema presentado a partir del siguiente diagrama.

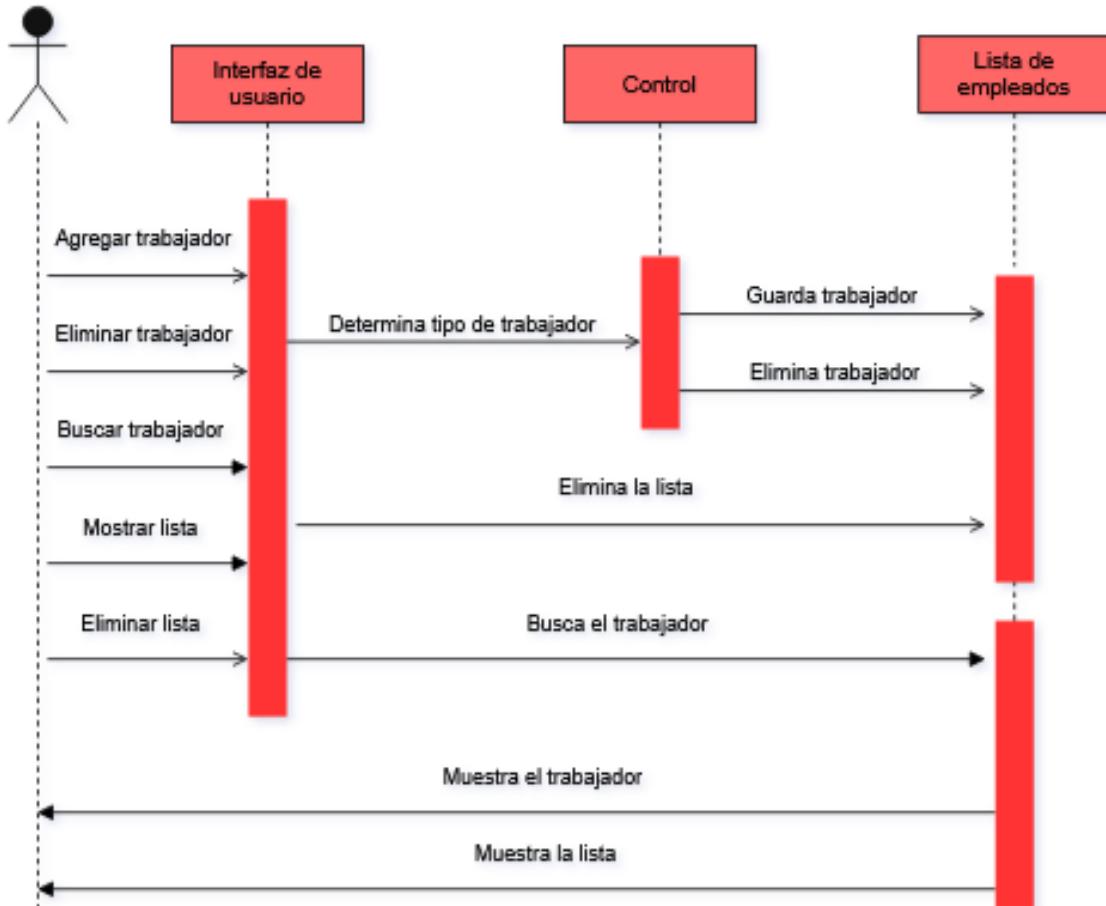


Figura 4: Diagrama de secuencia para la implementación propuesta.

Estas propuestas son solamente una parte de todo lo que puede realizarse a partir de la problemática establecida, ya que la aplicación puede ser mejorada de muchas maneras más, pero de manera concreta, se pudo realizar satisfactoriamente el buen funcionamiento de la aplicación, razón la cual era nuestro objetivo principal.

Referencias

- [1] Campus MVP. *Paquetes en Java: qué son, para qué se utilizan y cómo se usan.* 2023. Disponible en: <https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx>
- [2] DataCamp. *Encapsulación en Java.* 2024. Disponible en: <https://www.datacamp.com/es/doc/java/encapsulation>
- [3] W3Schools. *Java Inheritance.* 2024. Disponible en: https://www.w3schools.com/java/java_inheritance.asp
- [4] NTT Data. *Polimorfismo en Java: Programación Orientada a Objetos.* 2024. Disponible en: <https://ifgeekthen.nttdata.com/s/post/polimorfismo-en-java-programacion-orientada-objetos>

6. Datos de trabajo

Usuarios de Git del equipo:

Usuario de Git	Número de cuenta
SantiPichardo	322229916
thborty	425133840
eduarAG	320065570
CasvelGit	423020252
GT-8	425032224

Trabajo en el reporte:

- Introducción: Colaborada por 423020252
- Marco teórico: Colaborado por 322229916
- Desarrollo y resultados: Colaborado por 425032224
- Conclusiones: Colaborado por 425133840
- Diagramas: Colaborado por 320065570

Trabajo en el código:

- Clase Empleado: Colaborada por 425032224
- Clase EmpleadoAsalariado: Colaborada por 423020252
- Clase EmpleadoPorHoras: Colaborada por 425133840
- Clase MainApp: Colaborada por 320065570