	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): _____

Asignatura: _____

Grupo: _____

No de Práctica(s): _____

Integrante(s): 321172974

_____

_____

_____

No. de lista o brigada: _____

Semestre: _____

Fecha de entrega: _____

Observaciones: _____

_____

CALIFICACIÓN: _____

ÍNDICE

Introducción

Planteamiento del problema

Diseñar e implementar un programa en Java que permita manejar una jerarquía de clases para empleados de una compañía, donde la clase base sea Empleado y de ella deriven Manager, Desarrollador y Programador.

- Cada subclase debe tener los atributos nombre, dirección, salario y nombre de trabajo.
- Se deben implementar métodos de sobreescritura para calcular bonos, generar reportes de desempeño y manejo de proyectos (estos dos últimos se pueden representar con cadenas).

Motivación

La Herencia permite la reutilización de software, por lo que aprender a implementarla en conjunto con los anteriores conceptos vistos como paquetes, encapsulamiento, etc. hacen que nuestro paradigma[1] Orientado a objetos sea cada vez más fuerte.

Objetivos

- ☑ Implementar un sistema que solucione el problema bajo los requerimientos descritos en el Planteamiento mediante el concepto de herencia

Dicha implementación debe contar con:

- ☑ Encapsulamiento
- ☑ Definición dentro de un paquete
- ☑ Archivo jar y javadoc

Así como indicar mediante un diagrama de clases, su jerarquía

[1] **Paradigma**: Forma de pensar **[1]**

Marco Teórico

Herencia

Cuando se crea una clase, podemos optar por en vez de declarar miembros completamente nuevos, designar que la nueva clase herede los miembros de una clase existente; en otras palabras, la nueva subclase **extenderá** a la superclase.

A la nueva subclase, se le pueden añadir más campos y métodos, de tal forma que dichos objetos tengan dichas capacidades, además de las heredadas.

Ejemplo de extensión de una clase:

```
class nombreSubclase extends nombreSuperclase{...}
```

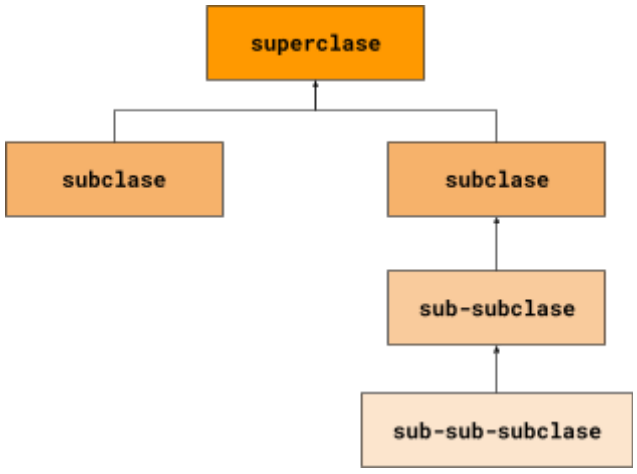


Figura 1. Jerarquía de clases

Note que la herencia tiene una dirección. (Flecha que une una subclase con su padre)

La **superclase directa** es la superclase a partir de la cual la subclase hereda en forma explícita.

Una **superclase indirecta** es cualquier clase arriba de la superclase directa en la jerarquía de clases.[2]

"La herencia es una forma de reutilización de software en la que se crea una nueva clase absorbiendo los miembros de una clase existente, y se mejoran con nuevas capacidades, o

con modificaciones en las capacidades ya existentes. Con la herencia, los programadores ahorran tiempo durante el desarrollo, al reutilizar software probado y depurado de alta calidad. Esto también aumenta la probabilidad de que un sistema se implemente con efectividad." **[2, p.379]**

Herencia jerárquica

Al momento de ocurrir la herencia, la clase ya existente que hereda sus atributos y métodos se convierte en la **superclase** y la nueva clase que recibe se extiende es la **subclase**

La herencia jerárquica ocurre cuando más de una clase hija hereda de una única clase base (padre). Esto significa que una clase padre puede tener múltiples subclases que heredan sus características y comportamientos. **[2, p.414]**

Herencia Multinivel

La herencia multinivel ocurre cuando una clase hereda de otra clase, y esa subclase a su vez es la clase base para una tercera clase. Esto forma una cadena de herencia que se extiende a lo largo de varios niveles dentro de la jerarquía de clases.

Aunque el concepto de **herencia múltiple no es soportado en Java,** sin embargo se puede simular algunos de los beneficios de ella con las interfaces, la de multinivel es aquella que se va generando con la herencia simple y se van teniendo distintos niveles de herencia. **[1]**

[2] **Jerarquía de clases**: Define las relaciones de herencia entre las clases

Desarrollo

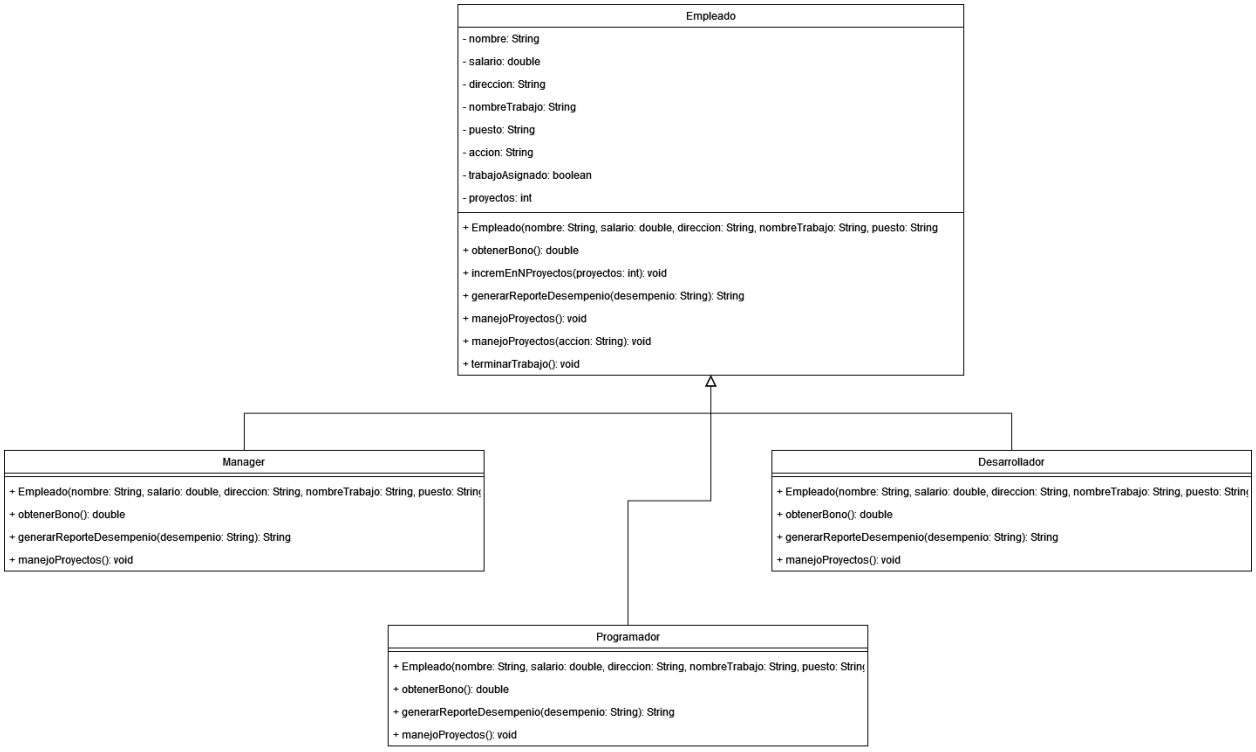
Ejercicios en clase

`Ejercicio0`	
Planteamiento	**Solución**
Programa con clase base Figura con método area, y las clases derivadas *Rectángulo* y *Circulo* que sobreescriban al método área para calcularla debidamente para cada figura	Se define la clase Figura con solo un método: area() Mediante herencia se extiende dicha clase a dos: Rectangulo y Circulo, donde cada una sobreescribe el método área para su particular cálculo. Cada clase hija también tiene sus getters y setters, además de sobreescribir el método toString() para poder imprimir un mensaje personalizado al momento de su invocación indirecta, sobre todo en System.out

`Ejercicio1`	
Planteamiento	**Solución**
Programa que tenga la clase CuentaBanco con los métodos depositar y retirar. Se crea una clase derivada CuentaAhorro con el método sobreescrito de retirar que previene retiros si la cuenta no posee saldo mayor a 100	Se define la clase CuentaBanco con sus setters y getters para las variables de acceso privado numeroCuenta y saldo. Se definen los métodos de depositar y retirar. En la clase heredera, CuentaAhorro, se hace una sobreescritura del método retirar, dado su peculiar comportamiento. Se hacen pruebas en el main de que todo funciona correctamente

Ejercicios de Tarea

Diagramas UML



Consideraciones

Para la construcción de una solución dada la libertad creativa que representa la implementación de los métodos de generar reportes de desempeño y manejo de proyectos, se estableció la siguiente lógica:

Existe una clase base: `Empleado.java`
Cada empleado tiene, además de los atributos de *nombre*, *salario*, *dirección y nombre de Trabajo*, los atributos:

`accion`	Describe en qué está trabajando dicho Empleado
`trabajoAsignado`	Booleano que indica si el empleado tiene un trabajo asignado
`proyectos`	El número de trabajos terminados (Al instanciarse, se inicializa en 0

La idea es que todo Empleado empiece con 0 proyectos o trabajos hechos, mediante el método de `manejoProyectos(String accion)`, se establezca que el **Empleado** está trabajando en lo que contenga la variable '`accion`' [3]
Cada vez que se imprima la sobrecarga de `manejoProyectos()`, lo que se imprimirá será una línea que indica que el empleado está trabajando la acción guardada (en forma de cadena) en `accion`.
Para poder decir que dicha actividad ha acabado, se ejecuta el método `terminarTrabajo()`, que lo que hace es resetear las variables e incrementar la variable de `proyectos`. Esto para que, cuando se llame el método `generarReporteDesempenio(String)`, se pueda imprimir el número de actividades (`proyectos`) realizados.[4]

De esta forma se sigue una secuencia:

- Establecer un manejo de Proyecto a un Empleado (**`manejoProyectos(String)`**)
- Terminar dicho proyecto: (**terminarTrabajo()**)
- *Repetir anteriores para establecer más proyectos terminados, (si se llama a **`manejoProyectos()`**, se imprime en pantalla que está haciendo dicho empleado en ese momento, es decir, se imprime el valor de la variable `accion`.*
- Generar reporte de desempeño (**`generarReporteDesempenio(String)`**)

Tablas de funciones

Presentes en las carpetas de Javadocs[5]

Pruebas

Para los 3 tipos de usuarios: Manager, Desarrollador y Programador, se probaron las acciones siguientes:

- Imprimir Bono
- Generar reportes (en varios momentos)
- Manejo de Proyectos (distintos proyectos, empezarlos y terminarlos)

[3] Esto mediante el uso de setters
[4] **Oportunidad de implementación**: establecer un rango para que de acuerdo al número de actividades, el mensaje del reporte no tenga que ser ingresado por el usuario, sino que se genera automáticamente, ejemplo: 1-3 actividades: Desempeño Bajo; 4-6 actividades: Desempeño bueno.
[5] Las carpetas llevan por nombre la clase que documentan

Resultados

Ejercicios hechos en clase

Ejercicio 1

```
El area del rectangulo es: 30.0

El area del circulo es: 78.53981633974483
```

Ejercicio 2

```
Se crea un objeto CuentaBanco (C/b No. CB1234) con un saldo inicial de $500
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 inicial de $ 450
Se retiran $300 de la cuenta CA1000
Saldo nuevo: 150.0

 Creando un objeto CuentaAhorro (C/a No. CA1001) con un saldo inicial de $300
Se intenta retiro por $250...
Se requiere un saldo de al menos 100
Saldo actual: 300.0
```

Ejercicios de tarea

`Manager`

Creación de Manager y generación de Reporte con el Empleado sin trabajos realizados

```
Reporte de desempenio del Manager Juan: Nulo.
(0 Trabajos supervisados y finalizados)
```

Impresión de Bonos

```
Bono: 1500.0
```

Manejo de proyectos. Al no tener algún proyecto en curso, como lo indica el método, imprime una acción por default.

```
Coordinando con otros departamentos
```

Llamada al método manejoProyectos() con la sobrecarga del argumento como cadena de accion; posteriomente se llama al método terminarTrabajo()

```
Supervisando Buscador para la empresa
Terminando trabajo en Buscador para la empresa
```

Asignación de tarea, conclusión de la tarea y generación de Reporte

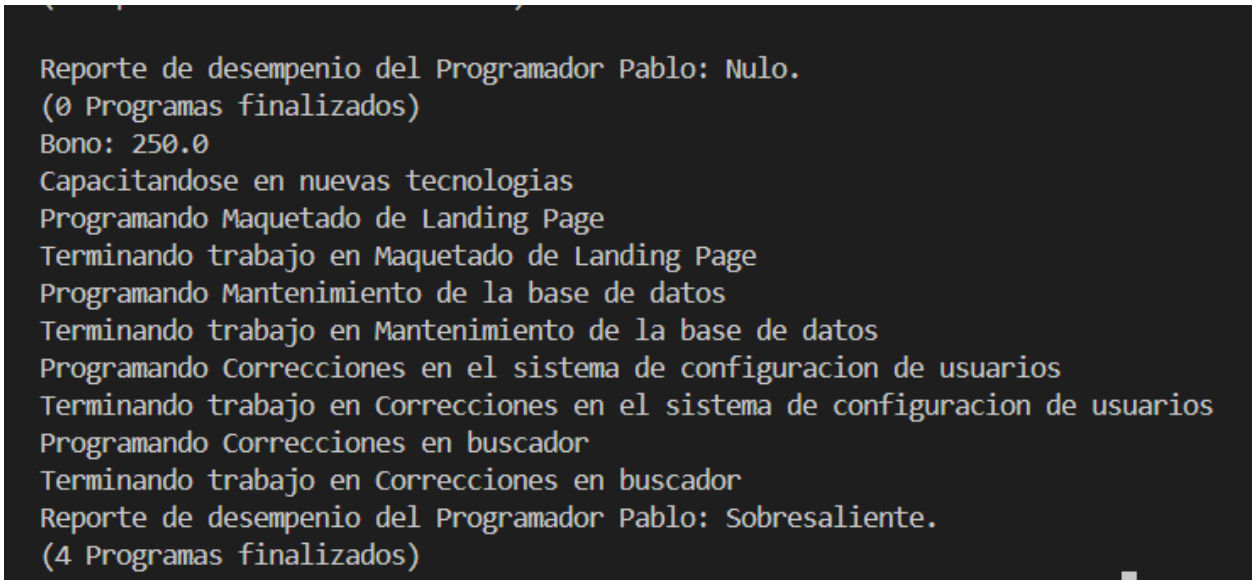
```
Supervisando Buscador para la empresa
Terminando trabajo en Buscador para la empresa
Reporte de desempenio del Manager Juan: Bueno.
(2 Trabajos supervisados y finalizados)
```

`Desarrollador`

Ciclo de pruebas completas

```
Reporte de desempenio del Desarrollador Pedro: Nulo.
(0 Implementaciones finalizadas)
Bono: 800.0
Corrigiendo errores del sistema
Diseniando Sistema de control de inventarios
Terminando trabajo en Sistema de control de inventarios
Diseniando Conectar base de datos con interfaz grafica
Terminando trabajo en Conectar base de datos con interfaz grafica
Diseniando API para la empresa
Terminando trabajo en API para la empresa
Reporte de desempenio del Desarrollador Pedro: Regular.
(3 Implementaciones finalizadas)
```

`Programador`
Ciclo de pruebas completas



Conclusiones

En un sistema amplio, las entidades tienden a parecerse o repetir alguno de sus comportamientos, no es para menos, ya que usualmente trasladamos el enfoque de objeto tangente y viviente al plano abstracto donde se emulan en algún software.

Este traslado mantiene conceptos fundamentales como lo son la esencia misma del objeto, ya sea inanimado o no, siendo estos sus características o atributos y funciones miembro o métodos. Cuando un concepto es base o muy general, tienden a existir objetos que toman dicha generalidad, la concretan e implementan otras en su definición, la acción de implementar aspectos de otros objetos se conoce como herencia, ya que una objeto extiende las funcionalidades ya atributos de otro.

En la programación, especialmente en el paradigma orientado a objetos, la herencia resulta un recurso indispensable para la generación de sistemas donde se presenta una clara jerarquía de comportamientos, donde la generalización de uno puede ser concretada en otro, la llamada sobreescritura.

Dado que se comprende la herencia simple, a multinivel y se implementa con clases que sobreescriben métodos que pueden ya no tener el mismo comportamiento para cuando la herencia llega a dicho nivel y la redefinen para su concreto funcionamiento, además de cubrir con los requerimientos adicionales, los objetivos de las práctica fueron cubiertos.

Referencias

[1] "Programación Orientada a Objetos" notas de clase de 1323, División de Ingeniería Eléctrica, Facultad de Ingeniería de la Universidad Nacional Autónoma de México, Verano 2024.

[2] P. Deitel y H. Deitel. *CÓMO PROGRAMAR EN JAVA*. Sétima edición. México: Pearson Educación México, 2008. ISBN: 978-970-26-1190-5.