UNIVERSIDAD NACIONAL DE INGENIERIA

INTRODUCCION A PROGRMACION

TITULO: Clase practica de las POO

INTEGRANTE: Zahid Esquivel Lainez

AÑO: 1er

CARNET: 2024-1693U

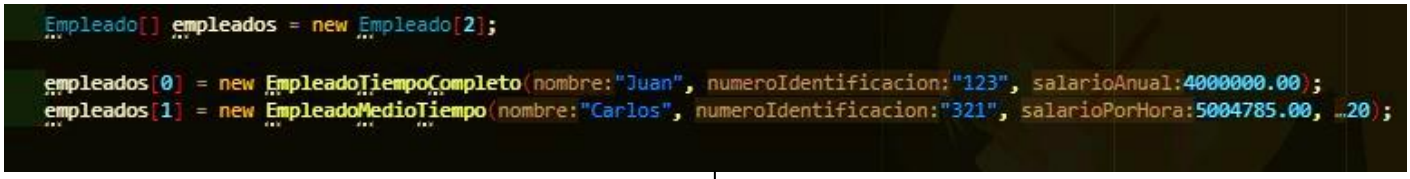
DOCENTE: Ing. Christopher Larios

FECHA DE ENTREGA: 31/5/2024

CUESTIONARIO DE LA GUIA

¿En qué parte del código usamos Polimorfismo, explique?

El polimorfismo se manifiesta en el uso de referencias de tipo **Empleado** para almacenar objetos de las subclases **EmpleadoTiempoCompleto** y **EmpleadoMedioTiempo**. Esto a mí me permite tratar a los objetos de estas subclases de manera ordenada.



Aquí, estoy creando un arreglo de **Empleado**, pero llenándolo con objetos de las subclases **EmpleadoTiempoCompleto** y **EmpleadoMedioTiempo**. Esto es un ejemplo de polimorfismo, ya que Empleado es la clase base y sus subclases pueden ser referenciadas por un tipo Empleado.

```java
for (Empleado empleado : empleados) {
    System.out.println("Empleado: " + empleado.getNombre() + "Salario mensual: $" + empleado.calcularSalario());
}
```

En este bucle, la variable empleada es de tipo **Empleado**, pero gracias al polimorfismo, puede referirse a instancias de cualquiera de sus subclases.

- Cuando el bucle empieza, empleado se asigna a **empleados**[0], que es una instancia de **EmpleadoTiempoCompleto**.
- La llamada a **empleado.calcularSalario()** invoca el método **calcularSalario()** de **EmpleadoTiempoCompleto**, porque empleado hace referencia a un objeto de esa clase.
- En la siguiente iteración, empleado se asigna a **empleados**[1], que es una instancia de **EmpleadoMedioTiempo**.
- La llamada a **empleado.calcularSalario()** invoca el método **calcularSalario()** de **EmpleadoMedioTiempo**, porque empleado ahora hace referencia a un objeto de esa clase.

```java
for (int i = 0; i < empleados.length; i++){
    System.out.println(x:"For normal");
    System.out.println("Empleado: " + empleados[i] + "Salario mensual: $" + empleados[i]);
}
```

De manera similar al primer bucle, aquí se accede a cada elemento del arreglo **empleados** mediante su variable tipo entero **i**.

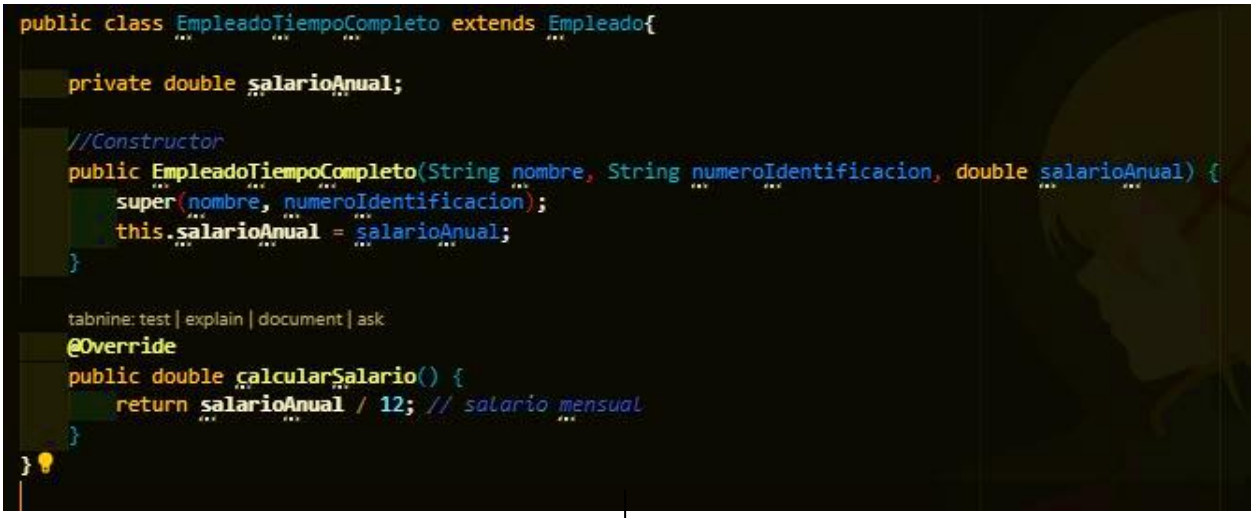
- En la primera iteración **(i = 0)**, **empleados**[i] es **empleados [0]**, que es una instancia de EmpleadoTiempoCompleto.
- La llamada a **empleados[**i] invoca el método **calcularSalario()** de EmpleadoTiempoCompleto.
- En la segunda iteración **(i = 1)**, **empleados[i]** es **empleados [1]**, que es una instancia de **EmpleadoMedioTiempo.**
- La llamada a **empleados[i]** invoca el método **calcularSalario()** de **EmpleadoMedioTiempo.**

En resumen:

El polimorfismo permite que **empleado.calcularSalario()** llame al método correcto dependiendo del tipo real del objeto (ya sea **EmpleadoTiempoCompleto** o **EmpleadoMedioTiempo**). Esto es posible porque **EmpleadoTiempoCompleto** y **EmpleadoMedioTiempo** implementan el método **calcularSalario()** de maneras diferentes.

¿En qué parte del código usamos Herencia, explique?

La herencia se aplica al definir las clases **EmpleadoTiempoCompleto** y **EmpleadoMedioTiempo** como subclases de Empleado.



```java
public class EmpleadoTiempoCompleto extends Empleado{

    private double salarioAnual;

    //Constructor
    public EmpleadoTiempoCompleto(String nombre, String numeroIdentificacion, double salarioAnual) {
        super(nombre, numeroIdentificacion);
        this.salarioAnual = salarioAnual;
    }

    tabnine: test | explain | document | ask
    @Override
    public double calcularSalario() {
        return salarioAnual / 12; // salario mensual
    }
}
```

La clase **EmpleadoTiempoCompleto** extiende **Empleado**, heredando sus atributos y métodos, además, define su propio atributo **salarioAnual** y proporciona una implementación para el método abstracto **calcularSalario().**

```java
public class EmpleadoMedioTiempo extends Empleado{

    private double salarioPorHora;
    private int horasTrabajadasPorSemana;

    //Constructor

    public EmpleadoMedioTiempo(String nombre, String numeroIdentificacion, double salarioPorHora, int horasTrabajadasPorSemana) {
        super(nombre, numeroIdentificacion);
        this.salarioPorHora = salarioPorHora;
        this.horasTrabajadasPorSemana = horasTrabajadasPorSemana;
    }

    @Override
    public double calcularSalario() {
        return salarioPorHora * horasTrabajadasPorSemana * 4; // salario mensual
    }

}
```

La clase EmpleadoMedioTiempo también extiende Empleado, heredando sus atributos y métodos. Define sus propios atributos **salarioPorHora** y **horasTrabajadasPorSemana,** y proporciona su propia implementación del método **calcularSalario()**.

¿En qué parte del código usamos Encapsulamiento, Explique?

Tenemos el encapsulamiento en la clase **Empleado** los atributos nombre y **numeroIdentificacion** están declarados como **private**, y significa que no pueden ser accedidos directamente desde fuera de la clase Empleado; esto es una aplicación directa del encapsulamiento.

Pero también tenemos **Encapsulamiento** en las Subclases **EmpleadoTiempoCompleto** y **EmpleadoMedioTiempo.**

private double salarioAnual;} El atributo **salarioAnual** es **private,** lo que significa que solo puede ser accedido dentro de la clase **EmpleadoTiempoCompleto**.

```java
public class EmpleadoTiempoCompleto extends Empleado{

    private double salarioAnual;

    //Constructor
    public EmpleadoTiempoCompleto(String nombre, String numeroIdentificacion, double salarioAnual) {
        super(nombre, numeroIdentificacion);
        this.salarioAnual = salarioAnual;
    }

    tabnine: test | explain | document | ask
    @Override
    public double calcularSalario() {
        return salarioAnual / 12; // salario mensual
    }
}
```

Similar a **EmpleadoTiempoCompleto**, el constructor y el método **calcularSalario** permiten interactuar con los atributos de **EmpleadoMedioTiempo** de manera controlada.

¿En qué parte del código usamos Abstracción, Explique?

Lo implementación del Método calcularSalario():

Clase Abstracta Empleado:

- ❖ Define una plantilla abstracta para representar a los empleados.
- ❖ Oculta los detalles específicos de cada tipo de empleado y se enfoca en la funcionalidad común.

Método Abstracto calcularSalario():

- ❖ Define un método abstracto en la clase Empleado para calcular el salario.
- ❖ No proporciona una implementación concreta, permitiendo que las subclases proporcionen su propia implementación.

Implementación Específica en las Subclases:

- EmpleadoTiempoCompleto y EmpleadoMedioTiempo implementan sus propias versiones del método calcularSalario(), lo que permite calcular el salario de acuerdo con las características específicas de cada tipo de empleado.