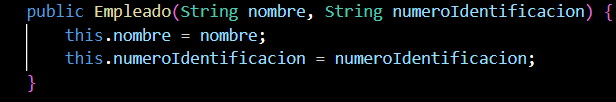
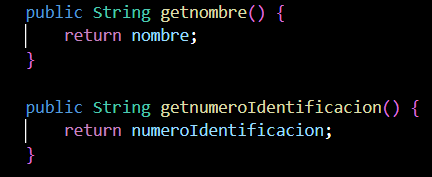


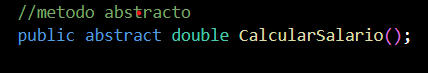
Estamos definiendo una clase llamada empleado que representa a un empleado. Tiene dos atributos privados: nombre(nombre) y numeroIdentificacion(número de identificación). Esto significa que solo se puede acceder a estos atributos dentro de la propia clase.



Esto define al constructor de la clase Empleado, que toma dos parámetros: nombre y numeroIdentificacion, ambos de tipo String. Se usa la palabra reservada this en los parámetros nombre y numeroIdentificacion para referirse específicamente la instancia del objeto actual. Esto evita ambigüedades.



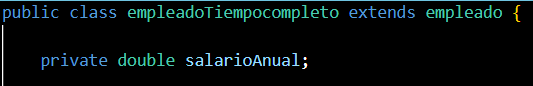
Ambos métodos devuelven el valor de la instancia, también permiten acceder a los valores de los campos privados nombre y numeroIdentificacion desde fuera de la clase.



Este método está declarado como abstracto dentro de una clase empleado. La palabra clave abstract indica que el método no tiene implementación en esta clase.

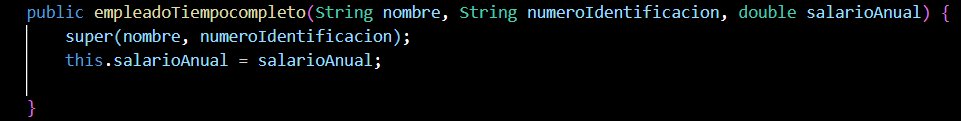
El método abstracto CalcularSalario está definido para devolver un valor de tipo double, que representa el salario que se va a calcular de la clase empleado.

Segunda parte



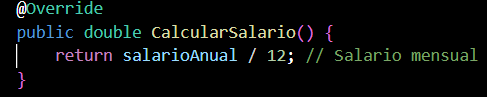
Se esta definiendo la clase EmpleadoTiempoCompleto que hereda los atributos de la clase Empleado. Se agrega un nuevo atributo salarioAnual a la clase EmpleadoTiempoCompleto, que representa el salario anual del empleado. El atributo salarioAnual es privado, lo que significa que solo puede ser accedido y modificado dentro de la clase EmpleadoTiempoCompleto.

Es de tipo double, lo que indica que puede almacenar valores decimales para representar el salario anual del empleado. Esto permite definir objetos que representan los empleados a tiempo completo y proporciona la capacidad de almacenar su salario anual.



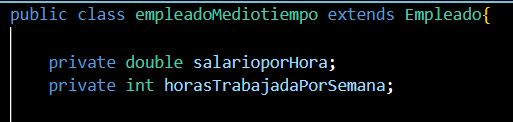
Se declara el constructor de la clase EmpleadoTiempoCompleto que toma tres parámetros: nombre, numeroIdentificacion y salarioAnual.

La palabra clave super llama al constructor de la clase base Empleado, pasando los parámetros nombre y numeroIdentificacion. Esto asegura una correcta inicialización de la clase hija, permitiendo una construcción adecuada de los objetos.



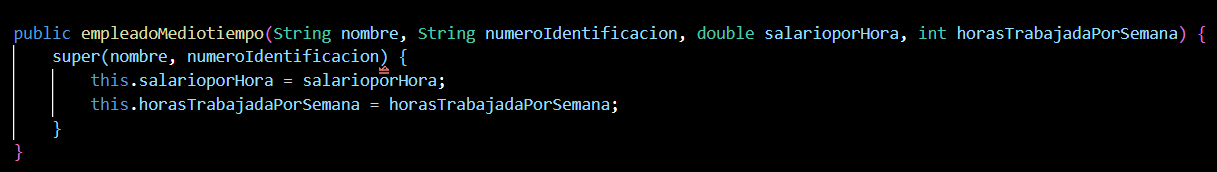
El @Override indica que el método sobrescribe el método de la clase base. En este caso, CalcularSalario de la clase base Empleado. El método devuelve un valor de tipo double, que representa el salario anual. Este método define cómo se calcula el salario del empleado a tiempo completo, proporcionando una implementación concreta del método abstracto CalcularSalario definido en la clase base.

Tercera parte



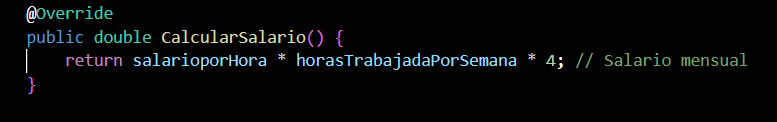
Se define una nueva clase EmpleadoMedioTiempo que hereda de la clase Empleado.Declara un atributo privado salarioporHora que representa el salario por hora del empleado. También declara un atributo privado horasTrabajadaPorSemana que representa el número de horas trabajadas por semana por el empleado.

Ambos atributos son privados, lo que significa que solo pueden ser accedidos y modificados dentro de la clase EmpleadoMedioTiempo. Esta clase permite representar empleados que trabajan a medio tiempo.



Se declara un constructor para la clase EmpleadoMedioTiempo que toma cuatro parámetros: nombre, numeroIdentificacion, salarioporHora y horasTrabajadaPorSemana. La palabra super Llama al constructor de la clase base Empleado para inicializar los atributos heredados nombre y numeroIdentificacion.

Se inicializa los atributos salarioporhora y horasTrabajadaPorSemana de la clase EmpleadoMedioTiempo la palabra this se usa para evitar ambigüedades.



@Override Indica que este método sobrescribe un método con la misma firma en la clase base Empleado. Se define el método CalcularSalario que debe devolver un valor de tipo double. Se calcula el salario mensual multiplicando salarioporHora por horasTrabajadaPorSemana y luego por 4 (asumiendo 4 semanas en un mes). Y se proporciona una implementación específica de cómo se calcula el salario mensual para un empleado de medio tiempo.

Cuarta parte



Se declara un arreglo de Empleado con capacidad para 2 elementos. Se crean dos objetos, uno de EmpleadoTiempoCompleto y otro de EmpleadoMedioTiempo, con los datos proporcionados. Los valores de salarioAnual y salarioPorHora son números (double), no cadenas de texto.

Se utiliza un bucle for-each para iterar sobre cada elemento del arreglo empleados. Para cada Empleado, se llama a los métodos getNombre y calcularSalario para obtener e imprimir el nombre y el salario anual y mensual.

**¿En qué parte del código usamos Polimorfismo, explique?**

En este código, el polimorfismo se manifiesta cuando se trata a los objetos de las clases derivadas (EmpleadoTiempoCompleto y EmpleadoMedioTiempo) como si fueran instancias de su clase base Empleado. Esto permite que el código sea más flexible y extensible.

**¿En qué parte del código usamos Herencia, explique?**

Se usa en la clase EmpleadoTiempoCompleto. Se extiende (hereda de) Empleado. Tiene un atributo adicional salarioAnual. Se usa el constructor de la superclase Empleado para inicializar nombre y numeroIdentificacion.

Se usa en la clase EmpleadoMedioTiempo. Se extiende (hereda de) Empleado. Tiene atributos adicionales salarioPorHora y horasTrabajadasPorSemana. Luego se usa el constructor de la superclase Empleado para inicializar nombre y numeroIdentificacion.

La herencia permite a las clases EmpleadoTiempoCompleto y EmpleadoMedioTiempo reutilizar los atributos y métodos de la clase base Empleado, reduciendo la duplicación de código y facilitando la extensión del comportamiento de los empleados de manera específica según el tipo de empleo (tiempo completo o medio tiempo).

**¿En qué parte del código usamos Encapsulamiento, explique?**

El encapsulamiento se usa para proteger los datos dentro de las clases y proporcionar métodos de acceso controlado a esos datos. Los atributos nombre y numeroIdentificacion en la clase Empleado son privados para evitar el acceso directo desde fuera de la clase.

Los métodos getNombre() y getNumeroIdentificacion() proporcionan una forma controlada de acceder a los atributos privados.

El encapsulamiento asegura que los datos internos de un objeto no puedan ser modificados arbitrariamente desde el exterior, lo que ayuda a mantener la integridad y coherencia del objeto.

El encapsulamiento es fundamental para la robustez y mantenibilidad del código, ya que permite cambios internos en la implementación de una clase sin afectar el código que usa esa clase.

**¿En qué parte del código usamos abstracción, explique?**

La abstracción en el código se utiliza principalmente en la definición de la clase abstracta Empleado y en la forma en que esta clase se emplea para representar empleados de diferentes tipos (tiempo completo y medio tiempo) sin preocuparse por los detalles específicos de su cálculo de salario.

La clase Empleado es abstracta, lo que significa que no se puede instanciar directamente. Está diseñada para proporcionar una interfaz común (atributos y métodos) para todas las subclases que representarán tipos específicos de empleados.

calcularSalario: Este método es abstracto, lo que obliga a las subclases a proporcionar una implementación específica para el cálculo del salario.

EmpleadoTiempoCompleto y EmpleadoMedioTiempo. Estas subclases extienden la clase abstracta Empleado y proporcionan implementaciones específicas del método calcularSalario.

En el método main de la clase App, se crean instancias de EmpleadoTiempoCompleto y EmpleadoMedioTiempo, pero se manejan utilizando referencias de tipo Empleado.

El código no necesita conocer los detalles de cómo se calcula el salario para cada tipo específico de empleado, solo sabe que cada Empleado tiene un método calcularSalario que puede ser llamado. Esto es un uso claro de la abstracción, ya que se está trabajando con el concepto general de un "Empleado" sin preocuparse por las implementaciones específicas de los subtipos.

La abstracción en el código permite manejar empleados de diferentes tipos de manera uniforme y extensible, facilitando el mantenimiento y la extensión del código.