Dylan Fernández / Sebastián mora

INFORME

La práctica consta de cinco actividades: las cuatro primeras forman parte del mismo paquete y la quinta es independiente.

Las actividades de la 1 a 4 se encuentran en el mismo paquete llamado Tpart1, que tiene una superclase llamada Vehículo y tres subclases llamadas Auto, Moto y Camioneta, que heredan atributos y métodos de la superclase. También contamos con un archivo principal que crea instancias, demuestra la ejecución y el polimorfismo. Estas clases también implementan la herencia y la anulación.

La clase Auto hereda de Vehículo y agrega un método propio llamado tipoDeCombustible, que imprime el tipo de combustible que usa el auto.

la clase Moto también hereda de Vehículo, pero además de un método de combustible, sobrescribe el método encender para personalizar el mensaje y mostrar que se trata de una moto.

La clase Camioneta sobrescribe igualmente el método encender e incluye información adicional como la capacidad de carga y el tipo de tracción, con un método extra para mostrar esos datos.

La clase Main es la que ejecuta el programa. En ella se crean instancias de Auto, Moto y Camioneta, y se llaman a sus métodos específicos para mostrar cómo cada uno funciona de manera diferente. Luego, se utiliza un arreglo de tipo Vehículo para almacenar los distintos objetos y se recorre con un ciclo, llamando al método encender de cada elemento. Aquí se evidencia el polimorfismo, ya que, aunque todos los objetos están guardados como tipo Vehículo, cada uno responde según su propia implementación del método.

Cuando el programa se ejecuta, en la consola se muestran mensajes que indican que cada vehículo ha sido encendido, el tipo de combustible que usan y, en el caso de la camioneta, su capacidad de carga y tracción. Finalmente, al recorrer el arreglo de vehículos, vuelve a mostrarse el encendido de cada uno, pero con el mensaje específico de su clase.

En conclusión, este código demuestra de manera sencilla cómo funciona la herencia en Java, cómo las subclases pueden sobrescribir métodos de la clase padre y cómo el polimorfismo permite trabajar con diferentes objetos de manera uniforme, pero conservando el comportamiento particular de cada uno.

informe de la Actividad 5

En esta práctica trabajamos el tema de la herencia en Java, aplicando los conceptos de superclase, subclases, polimorfismo y sobreescritura de métodos. El ejercicio de la actividad 5 consistía en desarrollar un sistema para gestionar distintos tipos de empleados, en este caso un gerente, un desarrollador y un administrativo, siguiendo el diagrama UML propuesto en la guía.

Para resolverlo se creó primero una clase base llamada Empleado, que tiene los atributos y métodos comunes como nombre, id y salario básico. A partir de esa clase se derivaron las tres subclases: Gerente, que incluye un bono anual; Desarrollador, que trabaja con el número de proyectos asignados; y Administrativo, que considera las horas extra realizadas. Cada una de estas clases sobrescribe el método de cálculo del salario para adaptarlo a su propia lógica, mostrando así cómo funciona la especialización en la herencia.

Además de las clases de empleados, se creó una clase llamada GestionPersonal, que tiene un método polimórfico llamado generarInforme. Este método recibe una lista de empleados sin importar si son gerentes, desarrolladores o administrativos, y genera un informe general con la información de cada uno. Gracias al polimorfismo, el método es capaz de tratar de la misma manera a diferentes tipos de empleados, pero ejecutando el comportamiento específico de cada clase.

Finalmente, en la clase Main se probaron los objetos, creando instancias de cada tipo de empleado, agregándolos a la lista y generando el informe. Al ejecutar el programa, se mostró cómo los salarios se calculan de forma diferente según el tipo de empleado, pero el sistema los gestiona a todos de manera unificada.

Con esta actividad se pudo comprobar de forma práctica la utilidad de la herencia para organizar mejor el código, evitando repeticiones innecesarias y facilitando la extensión de funcionalidades. También se entendió la importancia de la sobrescritura de métodos, que permite que cada subclase defina su propio comportamiento, y el polimorfismo, que brinda flexibilidad al trabajar con diferentes tipos de objetos bajo una misma referencia. En conclusión, la práctica permitió reforzar la comprensión de la programación orientada a objetos y su aplicación en problemas reales como la gestión de personal en una empresa.

PREGUNTAS

¿Cuál es la diferencia entre una superclase y una subclase en Java?

Superclase: es la clase "general" de la cual otras heredan. Define atributos y métodos comunes. Subclase: es una clase más específica que hereda de la superclase y puede añadir o modificar comportamientos.

¿Cómo se implementa la herencia en Java utilizando la palabra clave extends?

En Java, la herencia se implementa con extends. Una clase solo puede extender una clase padre (herencia simple).

¿Qué es la sobreescritura de métodos y cómo se utiliza en herencia?

Sobreescribir un método significa redefinir un método heredado para que tenga un comportamiento diferente en la subclase. Se usa la anotación @override para indicar que se está sobrescribiendo aunque ya no es necesario

¿En qué casos es recomendable utilizar herencia sobre composición?

Herencia: cuando existe una clara relación “extends” (Auto extends Vehículo, Moto extends vehículo). Composición: cuando la relación es “atributo” (Auto atributo Motor, Camioneta atributo Remolque).

¿Cómo se manifiesta el polimorfismo en un programa con herencia en Java?

El polimorfismo permite que una misma referencia de tipo superclase pueda apuntar a objetos de diferentes subclases, ejecutando el método correcto.

¿Cuáles son los problemas comunes al usar herencia y cómo pueden evitarse?

Aunque la herencia es una herramienta poderosa en Java, puede traer consigo algunos inconvenientes si no se utiliza correctamente. Uno de los problemas más frecuentes es la aplicación incorrecta de la herencia, cuando se fuerza una relación entre clases que no es realmente del tipo “es un”, lo cual genera jerarquías incoherentes y poco lógicas. También puede presentarse un acoplamiento fuerte, ya que las subclases dependen en exceso de la superclase, lo que hace que cualquier cambio en ella obligue a modificar todas las clases que heredan de la misma. Esto, a su vez, conlleva a una falta de flexibilidad, pues cuando la jerarquía es muy rígida, se dificulta la ampliación o adaptación del código a nuevas necesidades. Para evitar estos problemas, es recomendable usar la herencia únicamente cuando exista una relación clara de tipo “es un”, mientras que en otros casos es preferible recurrir a la composición. Además, se aconseja diseñar jerarquías simples y coherentes, y en situaciones que requieran mayor desacoplamiento, optar por el uso de interfaces.