1-5-2023

Taller POO

Lenguaje de programación 1

Elkin Jovany Garcia Henao – Sebastian Gonzales Echavarria
CORPORACIÓN UNIVERSTIARIA REMINGTON

1. Código para el programa vehículo: camión

→ INTERFAZ VEHICULO

```
public interface Vehiculo {
   public  void encender();
   public  void verificarPuertas(int estado);
   public  void acelerar(double velocidad);
   public void detener();
}
```

→ CLASE CAMION (METODOS)

```
mport java.util.Scanner;



public class Camion implements Vehiculo {
   //variables iniciales
   boolean motor, luces;
   int estado = 0;
   //metodo encender
   public void encender() {
      motor = true;
      luces = true;
      System.out.println("Encendiendo el vehiculo...");
      dormir();
      System.out.println("motor: encendido"
            + "\nluces: encendidas"
```

```
            + "\nEl vehiculo esta encendido \n");
        estado = 1;

        //se verifican las puertas
        verificarPuertas(estado);
    }
    //metodo para verificar las puertas

    //parametro  es el estado del vehiculo (encendido)
    public void verificarPuertas(int estado) {
        //se ingresa la cantidad de puertas del vehiculo
        System.out.println("cuantas puertas tiene el camion \n");
        Scanner entradaNumPuertas = new Scanner(System.in);
        int numPuertas = entradaNumPuertas.nextInt();
        if (estado == 1) {
            System.out.println("Verificando si todas las puertas estan cerradas");
            dormir();
            System.out.println("hay alguna puerta abierta"
                    + "\n Ingresa 1= si todas las puertas estan cerradas"
                    + "\n Ingresa 2= si hay alguna puerta abierta\n");
            Scanner entradaPuertasCerradas = new Scanner(System.in);
            int verificacionPuertas = entradaPuertasCerradas.nextInt();
            if (verificacionPuertas == 1) {
                System.out.println("a que velocidad deseas conducir");
                System.out.println("Los camiones solo pueden recorrer hasta
120km/h");
                System.out.println("10, 20 , 30, 40, 50, 60, 70, 80, 90, 100, 110, 120
km/h\n");
```

```
            Scanner ingresoVelocidadDeseada = new Scanner(System.in);
            double velocidad = ingresoVelocidadDeseada.nextDouble();
            acelerar(velocidad);

        } else if (verificacionPuertas == 2) {
            System.out.println("No se puede arrancar");
            acelerar(0);  //fin condicion puertas cerradas o abiertas
        }

    }//fin condicional estado
}//fin metodo
 //inicia el metodo acelerar
public void acelerar(double velocidad) {
    double motor = velocidad;
    if (motor == 0) {
        System.out.println("Vehiculo estacionado");
    } else if (motor > 0) {

        for (int i = 0; i < motor; i++) {
            try {
                System.out.println("El vehiculo esta en marcha");
                System.out.println("incrementando la velocidad en: " + (i + 1) + "
km/h");
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
        System.out.println("El vehiculo sigue a una velocidad constante\n");

    }
}
 // metodo detener
public void detener() {
    acelerar(0);
    motor = false;
    luces = false;
    System.out.println("Estacionando vehiculo");
    dormir();
    System.out.println("El vehiculo esta estacionado\n");
}

//metodo para dar timer
public void dormir() {
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

}

//metodo para sacar menu
public void menu() {
  Scanner entradaOpcion = new Scanner(System.in);
```

```
        System.out.println("Que metodo deseas utilizar: "
                        + "\n presiona el numero correspondiente: "
                        + "\n 1: Encender/verificarPuertas"
                        + "\n 2: detener"
                        + "\n 3: finalizar el programa\n");

        int opcionElegida = entradaOpcion.nextInt();
        if(opcionElegida==1)encender();
        else if(opcionElegida==2) detener();
        else if(opcionElegida==3)System.exit(0);
    }

}
```

→ CLASE EJECUTABLE MAIN

```
public class EjecutableMain {
    public static void main(String[] args) {
        Camion objeto = new Camion();
        while(true) {
                objeto.menu();
        }
    }
}
```

2. Consulta JAVA

- ✓ Modificadores:

En Java, los modificadores son palabras clave que se utilizan para especificar el nivel de acceso y la visibilidad de las variables, métodos y clases. Los modificadores en Java se pueden clasificar en cuatro categorías principales:

Modificadores de Acceso:

public: los miembros marcados como públicos son accesibles desde cualquier clase.

private: los miembros marcados como privados solo son accesibles dentro de la misma clase.

protected: los miembros marcados como protegidos son accesibles dentro de la misma clase y dentro de las subclases.

Modificadores de No Acceso:

static: indica que un miembro pertenece a la clase en lugar de a la instancia de la clase.

final: indica que un miembro no puede ser modificado una vez que se ha inicializado.

abstract: indica que una clase o método no tiene una implementación completa y debe ser implementado por una subclase.

Modificadores de Herencia:

extends: indica que una clase hereda los campos y métodos de otra clase.

implements: indica que una clase implementa una interfaz.

Modificadores de Sincronización:

synchronized: indica que un bloque de código solo puede ser accedido por un hilo a la vez.

volatile: indica que una variable puede ser modificada por múltiples hilos.

Es importante tener en cuenta que no todos los modificadores pueden aplicarse a todos los miembros. Por ejemplo, los modificadores "abstract" y "final" solo se pueden aplicar a métodos y clases, no a variables.

- ✓ Encapsulación:

La encapsulación en Java es un concepto fundamental de la programación orientada a objetos que se utiliza para proteger los datos de una clase y controlar su acceso desde el exterior. La encapsulación se logra mediante la definición de variables de instancia como privadas y la exposición de métodos públicos para acceder a ellas y modificarlas.

La encapsulación ayuda a garantizar la integridad de los datos de una clase, ya que cualquier modificación se realiza a través de métodos definidos por la clase, lo que permite la validación y verificación de los datos antes de su modificación. Además, la encapsulación permite cambiar la implementación interna de una clase sin afectar a otros componentes del programa que la utilizan, lo que aumenta la modularidad y la capacidad de mantenimiento del código.

- ✓ Paquetes Api

Los paquetes API (Application Programming Interface) son conjuntos de herramientas y bibliotecas que permiten a los desarrolladores acceder y utilizar una API de forma más fácil y eficiente en su aplicación o software.

Una API es un conjunto de protocolos, rutinas y herramientas que permiten la comunicación entre diferentes aplicaciones de software. Los paquetes API son una forma de encapsular esa funcionalidad en una biblioteca reutilizable que los desarrolladores pueden incorporar a sus propias aplicaciones. Esto les permite interactuar con una API de manera más sencilla y consistente, sin tener que escribir todo el código necesario desde cero.

Por ejemplo, una empresa podría proporcionar un paquete API que permite a los desarrolladores interactuar con su plataforma de pagos en línea. Este paquete API podría incluir funciones para procesar pagos, verificar la información de la tarjeta de crédito, obtener información de transacciones anteriores y más. Los desarrolladores podrían entonces utilizar este paquete API para integrar fácilmente la plataforma de pagos en línea en sus propias aplicaciones, sin tener que escribir todo el código necesario para realizar estas tareas desde cero

- ✓ Herencia:

En Java, la herencia es un mecanismo que permite a una clase (llamada "clase derivada" o "subclase") heredar características y comportamientos de otra clase (llamada "clase base" o "superclase").

Cuando una clase hereda de otra clase, la subclase hereda los campos y métodos de la superclase. Esto significa que la subclase puede usar los mismos campos y métodos que la superclase, así como agregar nuevos campos y métodos propios.

La herencia en Java se define utilizando la palabra clave "extends". Por ejemplo, si tenemos una clase llamada "Animal" y queremos crear una subclase llamada "Perro" que herede de la clase "Animal", podemos definirlo de la siguiente manera:

Ejemplo:

```
public class Animal {
   // campos y métodos de la clase Animal
}
public class Perro extends Animal {
   // campos y métodos de la clase Perro
}
```

En este ejemplo, la clase "Perro" hereda los campos y métodos de la clase "Animal". La clase "Perro" también puede agregar sus propios campos y métodos, que no estarán disponibles en la clase "Animal".

- ✓ Polimorfismo:

En Java, el polimorfismo se refiere a la capacidad de un objeto para tomar múltiples formas. En términos más técnicos, el polimorfismo en Java se refiere a la capacidad de un objeto de una clase para ser tratado como un objeto de una clase relacionada.

Hay dos formas principales de lograr el polimorfismo en Java: el polimorfismo de sobrecarga y el polimorfismo de anulación.

El polimorfismo de sobrecarga se produce cuando se tienen múltiples métodos con el mismo nombre en una clase, pero con diferentes parámetros. Esto permite a los objetos de la clase ser tratados de manera diferente según los argumentos pasados a los métodos.

El polimorfismo de anulación se produce cuando una subclase proporciona su propia implementación de un método que ya está definido en la clase base. Esto permite que los objetos de la subclase se traten como objetos de la clase base, pero con comportamientos específicos de la subclase.

El polimorfismo es una técnica muy útil en Java y es una de las principales características de la programación orientada a objetos. Permite escribir código más genérico y

reutilizable, lo que a su vez puede conducir a una mayor eficiencia y facilidad de mantenimiento del código.

- ✓ Clases inner (anidadas):

En programación, las clases inner anidadas son clases definidas dentro de otra clase. Estas clases pueden tener acceso a los miembros privados de la clase externa, y a su vez, la clase externa puede acceder a los miembros privados de las clases internas.

Además, las clases inner anidadas pueden ser de dos tipos:

Clases internas estáticas: Estas clases son miembros estáticos de la clase externa y se pueden acceder a ellos sin necesidad de crear una instancia de la clase externa. Para acceder a estas clases, se utiliza la sintaxis "ClaseExterna.ClaseInterna".

Clases internas no estáticas: Estas clases no son miembros estáticos de la clase externa y solo se pueden acceder a ellas a través de una instancia de la clase externa. Para acceder a estas clases, se utiliza la sintaxis "instanciaDeClaseExterna.ClaseInterna".

- ✓ Clases abstractas:

Las clases abstractas son clases en programación que no se pueden instanciar directamente, sino que se utilizan como plantillas o modelos para crear otras clases. Una clase abstracta proporciona una definición de métodos y propiedades, pero deja los detalles de su implementación a las clases hijas o subclases que la heredan.

En otras palabras, una clase abstracta define la interfaz y el comportamiento general que deben tener todas las subclases que la implementan, pero no proporciona una implementación completa de los métodos. Los métodos abstractos declarados en la clase abstracta deben ser implementados por las subclases concretas, que deben proporcionar una implementación concreta.

Las clases abstractas se utilizan para establecer una jerarquía de clases en la que las subclases comparten ciertos comportamientos o características comunes, pero tienen diferencias específicas. Al definir una clase abstracta, se puede garantizar que todas las subclases que la implementen tengan un conjunto mínimo de métodos y propiedades, lo que puede ayudar a mejorar la coherencia y la organización del código.

- ✓ Interface:

En Java, una interface es un tipo de objeto que define un conjunto de métodos (sin implementación) que una clase concreta debe implementar si implementa la interface. En

otras palabras, una interface define un contrato que debe cumplir cualquier clase que implemente dicha interface.

Las interfaces se definen mediante la palabra clave "interface" en Java, seguida del nombre de la interface y las firmas de los métodos que deben implementarse. Por ejemplo, la siguiente es la definición de una interface simple en Java:

```
public interface MiInterface {
   void metodo1();
   String metodo2(int valor);
}
```

En este ejemplo, se define la interface "MiInterface" con dos métodos, "metodo1" y "metodo2". Ambos métodos no tienen cuerpo ni implementación, solo especifican la firma del método (nombre, tipo de retorno y parámetros).

Cuando una clase implementa una interface, debe proporcionar una implementación para todos los métodos de la interface. Por ejemplo, la siguiente es la definición de una clase que implementa la interface "MiInterface":

```
public class MiClase implements MiInterface {
   public void metodo1() {
      System.out.println("Implementación de metodo1");
   }
   public String metodo2(int valor) {
      return "Implementación de metodo2 con valor " + valor;
   }
}
```

En este ejemplo, la clase "MiClase" implementa la interface "MiInterface" y proporciona una implementación para los métodos "metodo1" y "metodo2".

Las interfaces en Java son una herramienta poderosa para el diseño de software orientado a objetos, ya que permiten definir contratos claros y definir relaciones entre diferentes clases de manera flexible.

3. Paquetes de java para:

- ✓ QR:

ava proporciona una serie de bibliotecas y paquetes que pueden utilizarse para generar y leer códigos QR. Algunos de los paquetes más populares son:

ZXing (pronunciado "zebra crossing"): es una biblioteca de código abierto para la generación y lectura de códigos QR. ZXing es muy popular y cuenta con una gran cantidad de recursos y documentación disponible.

QRGen: es una biblioteca Java para la generación de códigos QR. Es fácil de usar y puede utilizarse para generar códigos QR con diferentes tamaños, formatos y niveles de corrección de errores.

QRCoder: es una biblioteca Java para la generación de códigos QR. Es compatible con diferentes tipos de datos, incluyendo texto, URL, correo electrónico, SMS y vCard.

QR-Code-Encoder-for-Java: es una biblioteca Java para la generación de códigos QR. Proporciona una API sencilla para generar códigos QR con diferentes niveles de corrección de errores y tamaños.

Además de estos paquetes, hay muchos otros disponibles que también pueden utilizarse para trabajar con códigos QR en Java.

- ✓ Bar Code:

Existen varias bibliotecas en Java para la generación y lectura de códigos de barras. Algunas de las más populares son las siguientes:

ZXing: es una biblioteca de código abierto para la generación y lectura de códigos de barras en Java. Soporta varios tipos de códigos de barras, incluyendo QR, Code 128, EAN-13, y UPC-A.

Barbecue: es otra biblioteca de código abierto que permite la generación de códigos de barras en Java. Soporta varios tipos de códigos de barras, incluyendo Code 39, Code 128, y UPC-A.

iText: es una biblioteca de código abierto para la creación de documentos PDF en Java, que también incluye soporte para la generación de códigos de barras. Soporta varios tipos de códigos de barras, incluyendo QR, Code 39, Code 128, y EAN-13.

Barcode4J: es otra biblioteca de código abierto para la generación de códigos de barras en Java. Soporta varios tipos de códigos de barras, incluyendo Code 128, EAN-13, y UPC-A.

Estas son solo algunas de las bibliotecas disponibles para la generación y lectura de códigos de barras en Java. Cada biblioteca tiene sus propias características y ventajas, por lo que es importante investigar y comparar antes de elegir una para su proyecto.

- ✓ Bluetooth:

En Java existen varias librerías y paquetes que permiten la comunicación con dispositivos Bluetooth. Algunos de ellos son:

javax.bluetooth: Este paquete proporciona las clases necesarias para realizar la comunicación con dispositivos Bluetooth. Incluye clases para la búsqueda de dispositivos, la conexión y la transferencia de datos.

bluecove: Es una implementación de la API de Java para Bluetooth, que permite la comunicación con dispositivos Bluetooth a través de diferentes plataformas, incluyendo Windows, Linux y Mac.

BlueZ: Es una implementación de Bluetooth para sistemas Linux, que incluye una interfaz de programación de aplicaciones (API) para Java.

AvetanaBluetooth: Es un paquete de Java que proporciona una API para la comunicación Bluetooth, y que soporta dispositivos tanto en modo cliente como en modo servidor.

BlueNRG: Es un paquete que proporciona una API para la comunicación Bluetooth de baja energía (BLE), y que es compatible con dispositivos que soportan esta tecnología.

Cada uno de estos paquetes tiene sus propias características y limitaciones, por lo que es importante evaluar cuál es el más adecuado para cada caso de uso específico.

- ✓ WHatsapp:

Existen varios paquetes de Java que se pueden utilizar para trabajar con la API de WhatsApp. A continuación, se mencionan algunos de los paquetes más populares:

Yowsup: Es una biblioteca de Python que puede ser utilizada para interactuar con la API de WhatsApp. También existe una versión para Java llamada Yowsup2.

Smack: Es una biblioteca de Java que puede ser utilizada para construir aplicaciones de mensajería instantánea y chat, incluyendo aplicaciones de WhatsApp.

WhatsApi: Es una biblioteca de Java que puede ser utilizada para interactuar con la API de WhatsApp. Actualmente, esta biblioteca está en desuso y no se recomienda su uso.

Es importante tener en cuenta que el uso de estas bibliotecas para interactuar con la API de WhatsApp puede violar los términos de servicio de WhatsApp, y podría llevar a la suspensión de la cuenta o incluso a acciones legales. Por lo tanto, es importante revisar cuidadosamente los términos de servicio de WhatsApp antes de utilizar cualquier paquete de Java para interactuar con la plataforma.

4. Como saber el tipo de dato de una variable en java

En Java, puedes conocer el tipo de dato de una variable usando el operador "instanceof" o el método "getClass()".

Usando el operador "instanceof":

```
if (variable instanceof String) {
   // La variable es de tipo String
}
else if (variable instanceof Integer) {
   // La variable es de tipo Integer
}
else if (variable instanceof Double) {
   // La variable es de tipo Double
}
// y así sucesivamente para otros tipos de datos
```

Usando el método "getClass()":

```
Class tipoDeDato = variable.getClass();

if (tipoDeDato == String.class) {
   // La variable es de tipo String
```

```
}
else if (tipoDeDato == Integer.class) {
    // La variable es de tipo Integer
}
else if (tipoDeDato == Double.class) {
    // La variable es de tipo Double
}
// y así sucesivamente para otros tipos de datos
```

Ten en cuenta que el operador "instanceof" y el método "getClass()" solo funcionan con tipos de datos de referencia (clases), no con tipos de datos primitivos como int, double, etc. Para los tipos de datos primitivos, puedes usar métodos de la clase correspondiente para realizar conversiones de tipo.

5. Manejo de cadenas y RegEx Java

En Java, las cadenas son secuencias de caracteres. El manejo de cadenas en Java incluye una variedad de métodos y funciones incorporadas para manipular y procesar cadenas, como concatenación, comparación, búsqueda y reemplazo de subcadenas, entre otros.

Además, Java también ofrece soporte para expresiones regulares (RegEx), que son patrones de búsqueda de texto que pueden ser utilizados para encontrar y manipular subcadenas específicas dentro de una cadena. Las expresiones regulares se crean utilizando una sintaxis especial y se pueden utilizar en combinación con métodos de cadenas incorporados para realizar operaciones avanzadas de búsqueda y manipulación de texto.

El manejo de cadenas y RegEx es una habilidad importante en la programación Java, especialmente en aplicaciones que implican el procesamiento de datos de texto y la validación de entradas del usuario.