CLASE ESPEJO

Estructura de datos

Master. Jesús Ariel González Bonilla

LinkedIn
https://www.linkedin.com/in/jesusarielgonzalezbonilla/

GitHub
https://github.com/ariel5253

CORHUILA
CORPORACIÓN UNIVERSITARIA DEL HUILA
Vigilada Mineducación

CORPORACIÓN UNIVERSITARIA DEL HUILA - CORHUILA
"Diseño y prestación de servicios de docencia, investigación y extensión de programas de pregrado, aplicando todos los requisitos de las normas ISO implementadas en sus sedes Neiva y Pitalito"

ITSON
UNIVERSIDAD

Algoritmos y Programación

Master.
Jesús Abraham Zazueta Castillo.

LinkedIn

GitHub

Optimizar tú código

Enlaces de interés: Puede acceder al siguiente enlace, el cual va a tener el resumen de la jornada académica.

https://github.com/code-corhuila/clase-espejo-2024-b.git

Conceptos básicos

Condicionales (if/else) Permiten tomar decisiones basadas en una condición lógica.

```
int x = 10;

if (x > 10) {
    System.out.println("El número es mayor que 10");
} else {
    System.out.println("El número es 10 o menor");
}
```

Conceptos básicos

Estructura Switch Utilizada para evaluar múltiples condiciones de manera más clara.

```
String color = "rojo";

switch (color) {
    case "rojo":
        System.out.println("El color es rojo");
        break;
    case "azul":
        System.out.println("El color es azul");
        break;
    default:
        System.out.println("Color desconocido");
        break;
}
```

Conceptos básicos

Bucles (for, while, do-while) Se usan para repetir un bloque de código.

```
for (int i = 0; i < 5; i++) {
    System.out.println("Iteración " + i);
}
```

```
int i = 0;
while (i < 5) {
    System.out.println("Iteración " + i);
    i++;
}
```

```
int i = 0;
do {
    System.out.println("Iteración " + i);
    i++;
} while (i < 5);
```

Estructuras de Datos Esenciales en Java

Bucles (for, while, do-while) Se usan para repetir un bloque de código.

Listas (ArrayList)

```
import java.util.ArrayList;

ArrayList<String> frutas = new ArrayList<>();
frutas.add("manzana");
frutas.add("naranja");
System.out.println(frutas.get(0)); // Salida: "manzana"
frutas.remove("naranja");
```

Arreglos (Arrays)

```
String[] frutas = {"manzana", "naranja", "plátano"};
System.out.println(frutas[0]);
frutas[2] = "fresa";
System.out.println(frutas[2]);
```

Estructuras de Datos Esenciales en Java

Mapas (HashMap)

```
import java.util.HashMap;

HashMap<String, Integer> mapa = new
HashMap<>();
mapa.put("clave1", 10);
mapa.put("clave2", 20);
System.out.println(mapa.get("clave1"));
```

Estructuras de Datos Esenciales en Java

Pilas (Stack)

```
import java.util.Stack;

Stack<Integer> pila = new Stack<>();
pila.push(1);
pila.push(2);
System.out.println(pila.pop()); // Salida: 2
```

Colas (Queue)

```
import java.util.LinkedList;
import java.util.Queue;

Queue<Integer> cola = new LinkedList<>();
cola.add(1);
cola.add(2);
System.out.println(cola.poll()); // Salida: 1
```

Estructuras de Datos Esenciales en Java

Conjuntos (HashSet)

```
import java.util.HashSet;

HashSet<Integer> conjunto = new HashSet<>();
conjunto.add(1);
conjunto.add(2);
conjunto.add(1); // Ignorado, ya que 1 ya existe
System.out.println(conjunto); // Salida: [1, 2]
```

Estructuras de Datos Esenciales en Java

Listas Enlazadas (LinkedList)

```
import java.util.LinkedList;

LinkedList<String> listaEnlazada = new LinkedList<>();
listaEnlazada.add("Nodo1");
listaEnlazada.add("Nodo2");
System.out.println(listaEnlazada.get(1)); // Salida: "Nodo2"
```

POO

```
// Definición de la clase
class Persona {
    String nombre;
    int edad;

    // Método de la clase
    void saludar() {
        System.out.println("Hola, mi nombre es " + nombre);
    }
}

// Clase principal
public class Main {
    public static void main(String[] args) {
        // Creación de un objeto
        Persona persona1 = new Persona();
        persona1.nombre = "Juan";
        persona1.edad = 25;
        persona1.saludar(); // Salida: Hola, mi nombre es Juan
    }
}
```

Clases y Objetos

- **Clase**: Es una plantilla o molde que define las propiedades y comportamientos de los objetos.
- **Objeto**: Es una instancia de una clase.

```
class CuentaBancaria {
    private double saldo;

    // Getter
    public double getSaldo() {
        return saldo;
    }

    // Setter
    public void depositar(double cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        }
    }
}

public class Main {
    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria();
        cuenta.depositar(1000);
        System.out.println("Saldo: " + cuenta.getSaldo()); // Salida: Saldo:
1000.0
    }
}
```


Encapsulamiento

Concepto: Restringe el acceso directo a los atributos de una clase y los controla mediante métodos (**getters** y **setters**).

```java
// Clase base (superclase)
class Animal {
    void comer() {
        System.out.println("Este animal está comiendo");
    }
}

// Clase derivada (subclase)
class Perro extends Animal {
    void ladrar() {
        System.out.println("El perro está ladrando");
    }
}

public class Main {
    public static void main(String[] args) {
        Perro perro = new Perro();
        perro.comer();  // Heredado de Animal
        perro.ladrar(); // Método propio
    }
}
```

POO

Herencia

Concepto: Permite que una clase (subclase) herede propiedades y métodos de otra clase (superclase).

Polimorfismo

Concepto: Permite que un objeto adopte múltiples formas. Principalmente se logra a través de **sobrecarga** y **sobrescritura** de métodos.

```
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido");
    }
}

class Gato extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El gato dice: Miau");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Gato(); // Polimorfismo
        miAnimal.hacerSonido(); // Salida: El gato dice: Miau
    }
}
```

```
class Calculadora {
    int sumar(int a, int b) {
        return a + b;
    }

    double sumar(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println(calc.sumar(3, 5));    // Salida: 8
        System.out.println(calc.sumar(3.5, 5.5)); // Salida: 9.0
    }
}
```

POO

```
abstract class Figura {
   abstract double calcularArea(); // Método abstracto
}

class Circulo extends Figura {
   private double radio;

   Circulo(double radio) {
      this.radio = radio;
   }

   @Override
   double calcularArea() {
      return Math.PI * radio * radio;
   }
}

public class Main {
   public static void main(String[] args) {
      Figura miCirculo = new Circulo(5);
      System.out.println("Área: " + miCirculo.calcularArea()); //
Salida: Área: 78.53981633974483
   }
}
```

```
abstract class Figura {
   abstract double calcularArea(); // Método abstracto
}
```

Abstracción

- **Concepto**: Oculta detalles complejos al usuario y expone solo lo esencial.
- Se logra con clases **abstractas** o **interfaces**.

```
interface Vehiculo {
   void conducir();
}

class Coche implements Vehiculo {
   public void conducir() {
      System.out.println("Conduciendo un coche");
   }
}

public class Main {
   public static void main(String[] args) {
      Vehiculo miCoche = new Coche();
      miCoche.conducir(); // Salida: Conduciendo un coche
   }
}
```

Resolver reto

Adjunto encuentre el ejercicio a resolver.