ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ECYS | DIDT
DESARROLLO DE TRANSFERENCIA TECNOLÓGICA

Día, Fecha:	Miércoles, 01/03/2023
Hora de inicio:	07:10

Introducción a la programación y computación 1 [D]

José Eduardo Morales García

Relaciones entre clases y objetos

- En programación orientada a objetos, las clases son plantillas o moldes que definen las características y comportamientos que tendrán los objetos que se crearán a partir de ellas.
- Un objeto es una instancia concreta de una clase y representa una entidad del mundo real o conceptual.
- Las relaciones entre clases y objetos pueden ser de varios tipos, entre los más comunes se encuentran:
  - Asociación
  - Agregación
  - Composición
  - Herencia

Herencia

- Una clase puede heredar características y comportamientos de otra clase. La clase que hereda se llama subclase o clase derivada, mientras que la clase que es heredada se llama superclase o clase base. De esta manera, una subclase puede reutilizar el código de la superclase y agregar sus propias características y comportamientos.

EJEMPLO

```java
// Clase Animal (superclase)
class Animal {
  String especie;
  int edad;

  void respirar() {
    System.out.println("El animal está respirando.");
  }
}
```

```java
// Clase Perro (subclase)
class Perro extends Animal {
  String raza;

  void ladrar() {
    System.out.println("Guau guau!");
  }
}
```

```java
// Clase Gato (subclase)
class Gato extends Animal {
  String color;

  void maullar() {
    System.out.println("Miau miau!");
  }
}
```

```java
// Uso de las clases
public class Main {
  public static void main(String[] args) {
    Perro miPerro = new Perro();
    miPerro.especie = "Canino";
    miPerro.edad = 3;
    miPerro.raza = "Labrador";
    miPerro.respirar();
    miPerro.ladrar();

    Gato miGato = new Gato();
    miGato.especie = "Felino";
    miGato.edad = 2;
    miGato.color = "Negro";
    miGato.respirar();
    miGato.maullar();
  }
}
```

Composición

- una clase puede contener instancias de otras clases como miembros. Por ejemplo, una clase "Coche" puede contener una instancia de la clase "Motor". La composición permite crear objetos complejos a partir de objetos más simples y reutilizar el código de las clases contenidas.

EJEMPLO

```java
// Clase Motor
class Motor {
  int potencia;

  void encender() {
    System.out.println("El motor está encendido.");
  }

  void apagar() {
    System.out.println("El motor está apagado.");
  }
}
```

```java
// Clase Coche
class Coche {
  String marca;
  String modelo;
  Motor motor;

  Coche(String marca, String modelo, int potencia) {
    this.marca = marca;
    this.modelo = modelo;
    this.motor = new Motor();
    this.motor.potencia = potencia;
  }

  void encenderCoche() {
    motor.encender();
    System.out.println("El coche está en marcha.");
  }

  void apagarCoche() {
    motor.apagar();
    System.out.println("El coche está parado.");
  }
}
```

```java
// Uso de las clases
public class Main {
  public static void main(String[] args) {
    Coche miCoche = new Coche("Ford", "Focus", 150);
    miCoche.encenderCoche();
    miCoche.apagarCoche();
  }
}
```

Agregación

- Una clase puede contener referencias a otras clases, pero no es responsable de su ciclo de vida. Por ejemplo, una clase "Universidad" puede contener referencias a varias instancias de la clase "Estudiante". La agregación permite representar relaciones entre entidades del mundo real.

EJEMPLO

```java
// Clase Jugador
class Jugador {
  String nombre;
  int edad;

  Jugador(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
  }
}
```

```java
// Clase Equipo
class Equipo {
  String nombre;
  List<Jugador> jugadores;

  Equipo(String nombre) {
    this.nombre = nombre;
    this.jugadores = new ArrayList<Jugador>();
  }

  void agregarJugador(Jugador jugador) {
    jugadores.add(jugador);
  }
}
```

```java
// Uso de las clases
public class Main {
  public static void main(String[] args) {
    Equipo miEquipo = new Equipo("Barcelona");
    Jugador jugador1 = new Jugador("Lionel Messi", 34);
    Jugador jugador2 = new Jugador("Neymar Jr", 29);
    miEquipo.agregarJugador(jugador1);
    miEquipo.agregarJugador(jugador2);
  }
}
```

Asociación

- una clase puede interactuar con otra clase sin que exista una relación de composición o herencia. Por ejemplo, una clase "Persona" puede tener una asociación con la clase "Coche" si una persona posee un coche.

EJEMPLO

```java
// Clase Persona
class Persona {
  String nombre;
  int edad;

  Persona(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  void adoptarMascota(Mascota mascota) {
    System.out.println(nombre + " ha adoptado a " + mascota.nombre);
  }
}
```

```java
// Clase Mascota
class Mascota {
  String nombre;
  String tipo;

  Mascota(String nombre, String tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
  }
}
```

```java
// Uso de las clases
public class Main {
  public static void main(String[] args) {
    Persona persona1 = new Persona("Juan", 35);
    Mascota mascota1 = new Mascota("Firulais", "Perro");
    persona1.adoptarMascota(mascota1);
  }
}
```

Polimorfismo

- El polimorfismo es una característica importante de la programación orientada a objetos que permite que un objeto pueda ser tratado como si fuera de un tipo diferente al suyo. En Java, el polimorfismo se logra a través del uso de clases y métodos abstractos, interfaces y clases que implementan esas interfaces.

Sobrecarga de métodos

- La sobrecarga de métodos en Java es una característica que permite definir varios métodos con el mismo nombre en la misma clase, siempre y cuando tengan diferentes parámetros. Esto significa que una clase puede tener dos o más métodos con el mismo nombre, pero cada uno con diferentes tipos de parámetros.

EJEMPLO

```java
public class EjemploSobrecarga {

    public int sumar(int a, int b) {
        return a + b;
    }

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

    public String sumar(String a, String b) {
        return a + b;
    }

    public static void main(String[] args) {
        EjemploSobrecarga ejemplo = new EjemploSobrecarga();
        System.out.println(ejemplo.sumar(2, 3)); // imprime 5
        System.out.println(ejemplo.sumar(2.5, 3.5)); // imprime 6.0
        System.out.println(ejemplo.sumar("Hola ", "Mundo")); // imprime "Hola Mund
    }
}
```

Virtualización

- El polimorfismo en Java se refiere a la capacidad de un objeto para ser tratado como un objeto de su superclase o interfaz. La virtualización es una técnica que permite simular o emular recursos o comportamientos físicos en un entorno virtual. En Java, el polimorfismo y la virtualización se utilizan juntos para crear objetos que se comportan de manera diferente según su tipo real.

EJEMPLO

```java
public abstract class Animal {
    public abstract void hacerSonido();
}
```

```java
public class Gato extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("Miau!");
    }
}
```

```java
public class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("Guau!");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Perro();
        Animal animal2 = new Gato();

        animal1.hacerSonido(); // imprime "Guau!"
        animal2.hacerSonido(); // imprime "Miau!"
    }
}
```

Construcciones abstractas

- En programación, una construcción abstracta se refiere a un elemento de código que no tiene una implementación concreta y se utiliza para definir una estructura o comportamiento genérico que luego puede ser implementado por clases concretas. Las construcciones abstractas son muy utilizadas en lenguajes de programación orientados a objetos, como Java, para definir clases abstractas e interfaces.

Clase abstracta

- Una clase abstracta en Java es una clase que no se puede instanciar directamente, sino que se utiliza como una superclase para otras clases. Las clases abstractas se utilizan para definir una estructura común para un conjunto de subclases relacionadas.

EJEMPLO

```java
public abstract class Animal {
    protected String nombre;

    public Animal(String nombre) {
        this.nombre = nombre;
    }

    public abstract void hacerSonido();

    public String getNombre() {
        return nombre;
    }
}
```

```java
public class Perro extends Animal {
    public Perro(String nombre) {
        super(nombre);
    }

    @Override
    public void hacerSonido() {
        System.out.println("Guau!");
    }
}
```

```java
public class Gato extends Animal {
    public Gato(String nombre) {
        super(nombre);
    }

    @Override
    public void hacerSonido() {
        System.out.println("Miau!");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Animal perro = new Perro("Fido");
        Animal gato = new Gato("Pelusa");

        System.out.println(perro.getNombre() + " hace:");
        perro.hacerSonido();

        System.out.println(gato.getNombre() + " hace:");
        gato.hacerSonido();
    }
}
```

Interface

- En Java, una interfaz es una colección de métodos abstractos y constantes que pueden ser implementados por cualquier clase que implemente la interfaz. Las interfaces se utilizan para definir un contrato que una clase debe cumplir para poder utilizarse en cierto contexto.

EJEMPLO

```
public interface Forma {
    double getArea();

    double getPerimetro();
}
```

Parte práctica