Apuntes Parcial

El main es el punto de entrada al programa y siempre tiene la misma forma

▼ Clases - Atributos y Métodos

Una clase es un plano de un objeto. Define las características (atributos) y comportamientos (métodos) que tendrá. **NO ES UN OBJETO** es la plantilla para crear el objeto

```
public class Persona {
    // Atributos (características)
    String nombre;
    int edad;

    // Métodos (comportamientos)
    public void saludar() {
        System.out.println("Hola, mi nombre es " + nombre);
    }
}
```

Atributos

Son las variables que describen el estado de un objeto

```
Persona p1 = new Persona();
p1.nombre = "Ana";
p1.edad = 25;
```

Métodos

Son las acciones que el objeto puede realizar. Son funciones que pertenecen a la clase

```
p1.saludar();  // Imprime: "Hola, mi nombre es Ana"
```

Herencia

Permite a subclases heredar atributos o métodos de una superclase. De esta forma se puede reutilizar código y modelar relaciones jerárquicas

```
class Animal {
    public void hacerSonido() {
        System.out.println("Sonido genérico de animal");
    }
}

class Perro extends Animal {
    @Override
    public void hacerSonido() {
        // Llamamos primero al comportamiento del padre
        super.hacerSonido();
        // Luego redefinimos con el comportamiento específico
        System.out.println("Guau guau");
    }
}

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

Polimorfismo

El polimorfismo significa que un mismo método puede tener diferentes comportamientos dependiendo el objeto que lo utilice

```
class Animal {
    public void hacerSonido() {
        System.out.println("Sonido genérico de animal");
    }
}

class Perro extends Animal {
    public void hacerSonido() {
        System.out.println("Guau guau");
    }
}

class Gato extends Animal {
    public void hacerSonido() {
        System.out.println("Miau miau");
    }
}

// Uso
Animal a1 = new Perro();
Animal a2 = new Gato();

a1.hacerSonido();  // "Guau guau"
a2.hacerSonido();  // "Miau miau"
```

▼ Sobrecarga

Es cuando una clase tiene métodos con el mismo nombre pero con parámetros diferentes

```
public class Calculadora {
    // Método suma con 2 enteros
    public int sumar(int a, int b) {
        return a + b;
    }

    // Método suma con 3 enteros
    public int sumar(int a, int b, int c) {
```

```
        return a + b + c;
    }

    // Método suma con decimales
    public double sumar(double a, double b) {
        return a + b;
    }
}

// Uso
Calculadora calc = new Calculadora();
System.out.println(calc.sumar(2, 3));       // 5
System.out.println(calc.sumar(2, 3, 4));    // 9
System.out.println(calc.sumar(2.5, 3.2));   // 5.7
```

▼ Interfaces

Define un contrato , métodos que una clase debe implementar

Sirve para que distintas clases tengan comportamientos en común

```
interface Vehiculo {
    void arrancar();
    void frenar();
}

class Auto implements Vehiculo {
    public void arrancar() {
        System.out.println("El auto arranca con la llave");
    }
    public void frenar() {
        System.out.println("El auto frena con los frenos ABS");
    }
}

class Bicicleta implements Vehiculo {
    public void arrancar() {
        System.out.println("La bicicleta comienza a pedalear");
    }
```

```
    public void frenar() {
        System.out.println("La bicicleta frena con los frenos manuales");
    }
}

// Uso
Vehiculo v1 = new Auto();
Vehiculo v2 = new Bicicleta();

v1.arrancar();  // "El auto arranca con la llave"
v2.arrancar();  // "La bicicleta comienza a pedalear"
```

▼ Relaciones

Permiten representar cómo se vinculan las entidades dentro de un sistema. Se pueden clasificar en 3 grupos: Generalización , Asociación y Dependencia

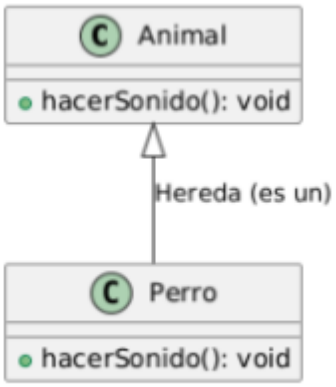
Relaciones de Generalización

Relación jerárquica entre tipos. Una subclase hereda de una superclase. También se aplica cuando una clase implementa una interfaz. Estas relaciones:

- No presentan cardinalidad especifica
- Se aplica entre tipos y no objetos

Relación de Herencia o Extensión

Una subclase hereda atributos y comportamientos de una superclase. Se encuentra representada por la relación de tipo “es un”.



Relación de Implementación

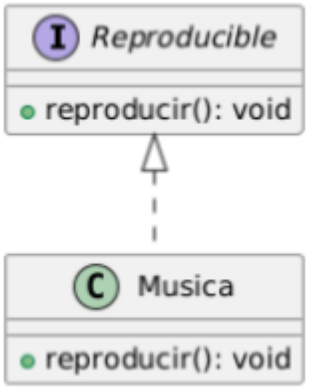
Cuando una clase se compromete a implementar una interfaz, la interfaz define el qué debe implementar la clase pero no especifica el cómo. Esto deja abierta la posibilidad de aplicar polimorfismo sobre los métodos definidos.

Las interfaces en la POO establecen un contrato: define las operaciones disponibles (el qué), peor no su comportamiento (el cómo). Por lo tanto:

1. La clase concreta define el cómo implementar esos métodos , de la forma que mejor se adapte a su lógica.
2. El código cliente puede interactuar con la interfaz sin conocer, ni depender directamente, de la clase concreta que la implementa

Así una interfaz resulta más flexible que una clase abstracta o una superclase

1. Soportas múltiples implementaciones . Una clase puede implementar varias interfaces
2. No impone estructura interna , como si lo hace la superclase
3. La clase solo implementa métodos públicos sin implementación
4. Es útil cuando es necesaria flexibilidad y bajo acoplamiento



Relaciones de Asociación

indica que una clase conoce o interactúa con otra

Las clases se refieren entre sí como parte de su definición

1. Puede existir cardinalidad
2. Puede ser unidireccional o bidireccional

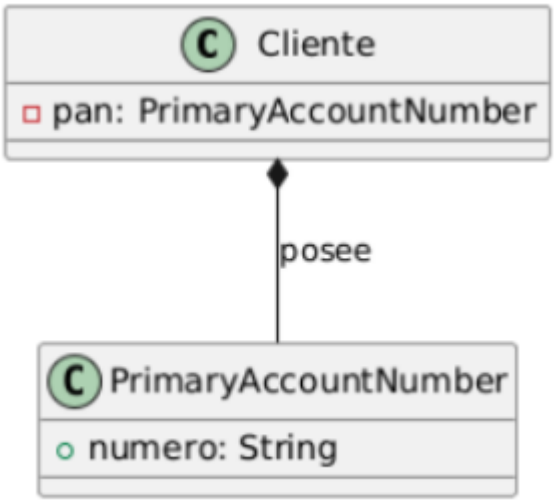
3. La clase solo implementa métodos públicos sin implementación
4. Es útil cuando es necesaria flexibilidad y bajo acoplamiento

Relación de Composición

Cuando una clase contiene a otra forma esencial e inseparable. Una no tiene razón de ser sin la otra.

Ejemplo:

- Una página no existe sin su libro
- Un sueldo no puede existir sin empleado (Siempre debe ser abonado a un empleado por lo que siempre estará asociado a un empleado)
- Una tarjeta de crédito/debito su número (PAN - Primary Account Number) siempre debe estar asociado a un titular. Si por alguna razón este da de baja su relación con la entidad emisora, el PAN se daría de baja de forma automática junto al cliente. El ciclo de vida del PAN depende totalmente del ciclo de vida del todo



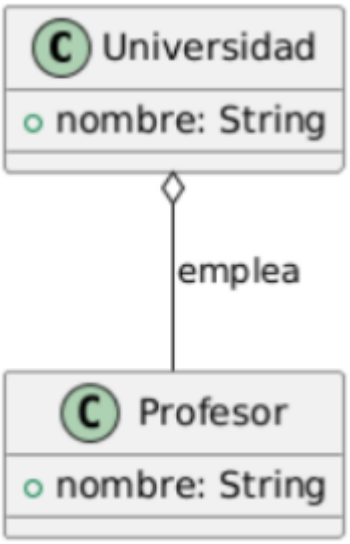
Relación de Agregación (relación de tipo debil)

Una clase contiene y agrupa a otras sin embargo estas pueden existir de forma independiente.

Ejemplo: Universidad y profesores. La universidad agrega profesores para su funcionamiento pero no los crea ni destruye. El ciclo de vida de cada profesor es independiente al de la universidad. La universidad “emplea” un profesor

El todo utiliza las partes y estas pueden existir de forma independiente

1. un estudiante existe por fuera de la Universidad (el sueldo y el PAN no)
2. Puede estar relacionado con varias universidades a la vez o carreras
3. Si la universidad o curso desaparece, el estudiante o profesor continua con su vida



Relación de Asociación Simple

Representa que una clase conoce a otra y puede interactuar con ella, no implica propiedad, composición ni agrupación lógica. Solo indica que hay un vinculo estructural o de colaboración entre ambas clases

1. Pueden ser bidireccionales y tener cardinalidad
2. No existe dependencia existencial . Una clase puede existir sin la otra

Por ejemplo: Alumno se inscribe en un curso, tanto el alumno como el curso pueden seguir existiendo y ni el alumno es dueño del curso ni viceversa. Sin embargo se conocen.

Se utiliza cuando una clase necesita conocer a otra pero sin implicar pertenencia , exclusividad ni dependencia del ciclo de vida

Relaciones de Dependencia

Se trata de un vínculo temporal y débil . No existe una relación permanente entre clases, representa que una clase requiere de otra para funcionar

1. No son parte del objeto
2. Existe acoplamiento funcional, la clase requiere a la otra para realizar su función
3. No llevan cardinalidad
4. Tienen variaciones de fuerza

Usos más comunes:

1. Llamadas a servicios externos
2. Inyección de dependencias
3. Librerías o componentes temporales
4. Configuraciones opcionales
5. Plugins

Es un vínculo de acoplamiento funcional, no estructural. No implica propiedad ni composición

Dependencia Fuerte

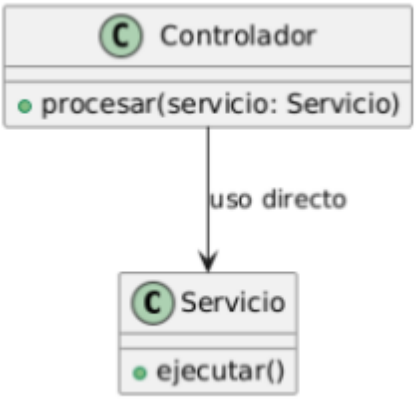
Cuando una clase usa otra de forma directa , concreta y obligatoria sin margen de flexibilidad o reemplazo

Se manifiesta cuando:

1. La clase se pasa como parámetro obligatorio
2. Se invoca sin validación previa
3. El sistema falla si esta no existe

Esto implica alto acoplamiento y dependencia funcional fuerte. La clase queda vinculada de forma permanente, entonces los cambios en la clase proveedora afectan a la consumidora

Se debe evitar siempre que se pueda este tipo de dependencias

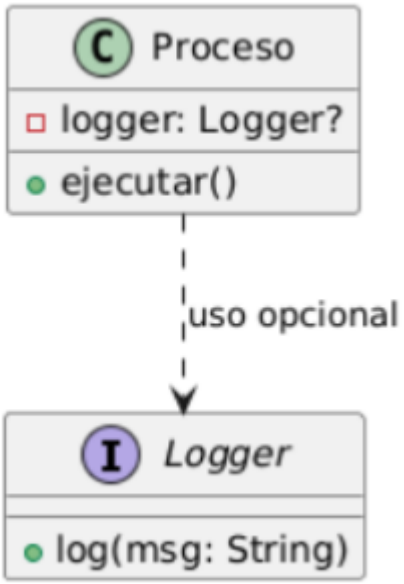


Dependencia Débil

Una clase puede usar otra, pero no la requiere para funcionar correctamente

Suele aparecer cuando existe verificación previa al uso. Se usan interfaces en lugar de clases , se aplican mecanismos de inyección de dependencias

La clase se ocupa de su lógica sin recurris a otra para realizar su función. Favorece la flexibilidad, extensibilidad y sostenibilidad del sistema.


Proceso
logger: Logger?
ejecutar()
uso opcional
Logger
log(msg: String)