UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

Programación Orientada a Objetos

Sesión 9: Diseño de diagrama de clases del UML usando herencia simple y herencia múltiple

Recordando:

¿Que vimos la sesión pasada?

Universidad Tecnológica del Perú

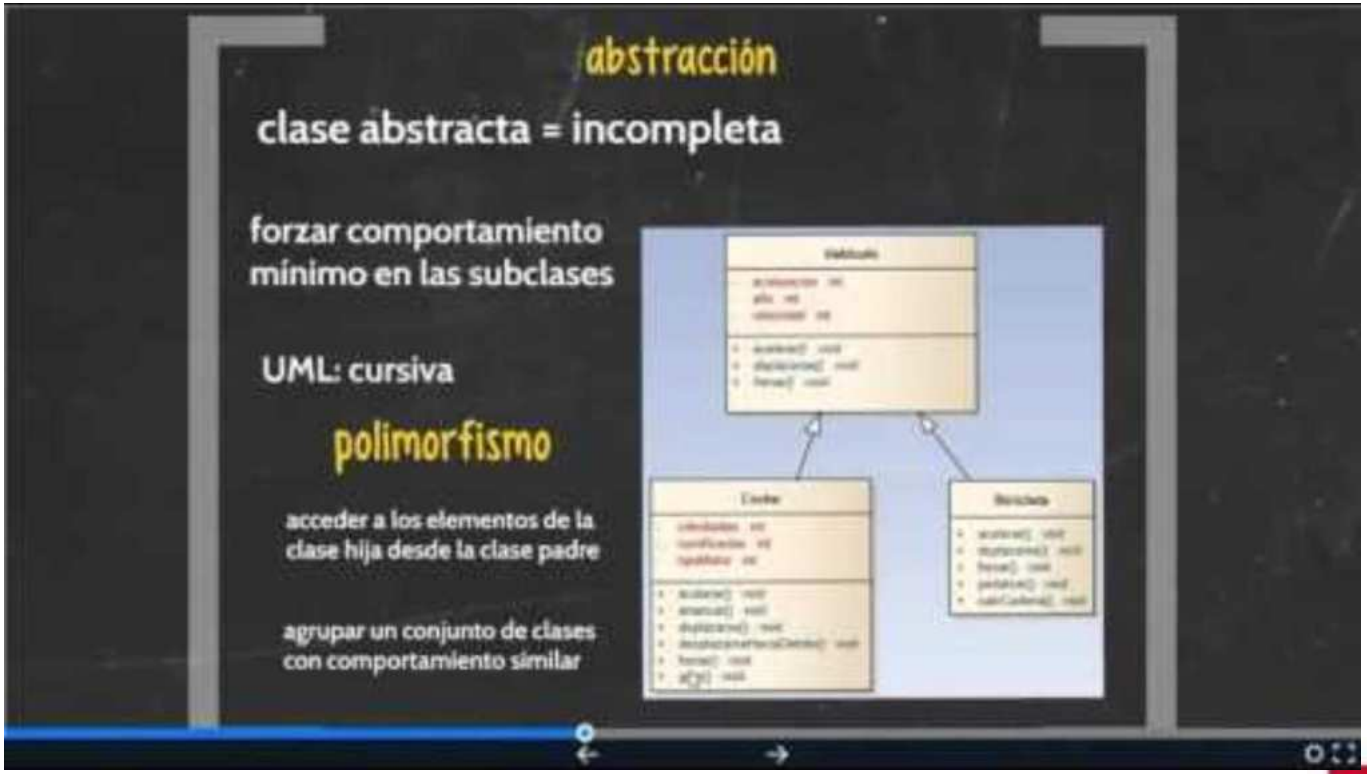
Logro de aprendizaje

Al finalizar la sesión, el estudiante comprende la naturaleza y propósito de las clases abstractas e interfaces en la solución de problemas usando Java con ejemplos prácticos.

Clase abstracta

Las clases abstractas funcionan como una clase que declara la existencia de métodos pero no su implementación. Eso es algo que haremos después en las diferentes subclases derivadas de la clase abstracta. Una clase abstracta puede contener métodos no abstractos pero al menos uno de los métodos sí debe serlo.

Clase abstracta



https://www.youtube.com/watch?v=8UhPB6dc5WQ

Universidad Tecnológica del Perú

```java
public abstract class Producto  {
    private int precio;

    public Producto(int precio) {
      this.precio = precio;
    }
   public int getPrecio(){
      return this.precio;
   }
    public abstract void getName();
}

public class Banana extends Producto{

    public Banana(){
       super(200);
    }

    @Override
    public String getName(){
      return "banana";
    }
}
```

Clase abstracta

Para verlo más claro lo mejor es fijarnos en el siguiente ejemplo:

UTP Universidad Tecnológica del Perú

Clase abstracta

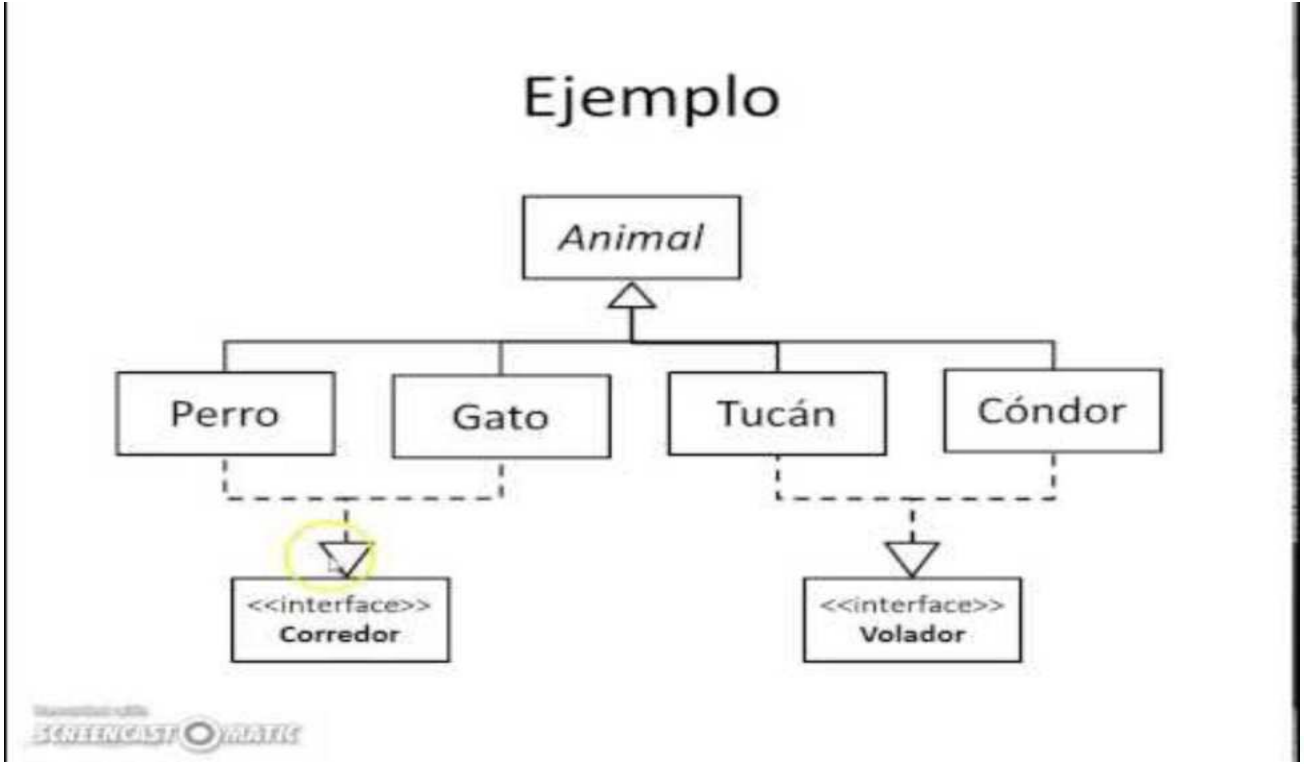
Interpretación:

- Como puedes ver creamos una clase abstracta “Producto” donde declaramos un método abstracto llamado “getName” pero éste no está implementado en esa clase sino que que es en la clase “Banana” donde definimos el método y su comportamiento.
- En otras palabras, lo que hacemos es crear una “anotación” que nos recuerde que debemos definir sí o sí la implementación de dicho método en cada nueva clase que definamos a continuación y que dependa de nuestra clase abstracta.

Interfaces

Los Interfaces son unas clases completamente abstractas que contiene sólo una colección de métodos abstractos y propiedades constantes.
Al igual que en las clases abstractas en un Interface se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describen las acciones que puede llevar a cabo un determinado objeto.

Interfaces



https://www.youtube.com/watch?v=_CzsJr3JT0Y

Universidad Tecnológica del Perú

Especificaciones para Interfaces

- Lo definimos utilizando la palabra clave de la interface.
- Puede contener solo variables estáticas.
- No puede contener un constructor porque las interfaces no pueden ser instanciadas.
- Las interfaces pueden extender otras interfaces.
- Una clase puede implementar cualquier número de interfaces.

Interfaces

```
interface Animal {
  public void eat();
  public void makeSound();
}

class Cat implements Animal {
  public void makeSound() {
    System.out.println("Meow");
  }
  public void eat() {
    System.out.println("omnomnom");
  }
}
```

Un ejemplo de una interfaz simple:

Propiedades de las Interfaces

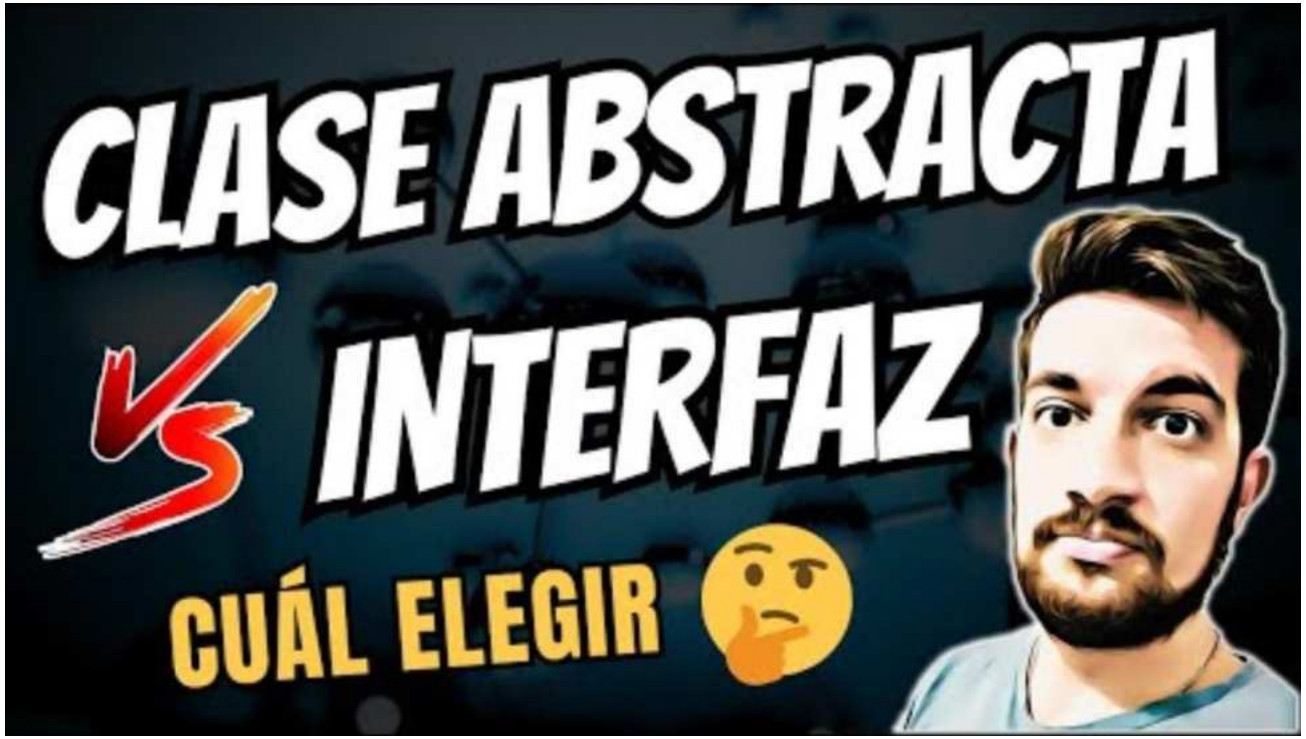
- Una interfaz es implícitamente abstracta. No es necesario utilizar la palabra clave abstracta al declarar una interfaz.
- Cada método en una interfaz también es implícitamente abstracto, por lo que la palabra clave abstracta no es necesaria.
- Los métodos en una interfaz son implícitamente públicos.
- Una clase puede heredar de una sola superclase, ¡pero puede implementar múltiples interfaces.

Clase abstracta vs Interface

Cuando una clase hereda de otra clase (abstracta o no), estás definiendo qué es, pasar de una idea abstracta a una concreción. Además, estás definiendo una relación entre clases pero cuando implementas una interfaz, estás definiendo cómo se comporta. Es algo como definir y cumplir un contrato.

Quizá lo mejor, ya a modo conclusión, es ver las clases Abstractas e Interfaces frente a frente en una tabla comparativa. Esto te ayudará a entenderlas mejor aunque mi recomendación es siempre probar con algunos ejemplos y creando tu propio código.

Clase abstracta o Interface



https://www.youtube.com/watch?v=Bj2ta9xv4cM

UTP Universidad Tecnológica del Perú

Clase abstracta vs Interface

Clase Abstracta	Interfaz
La palabra clave abstract se usa para crear una clase abstracta y se puede usar con métodos.	La palabra clave de interface se usa para crear una interfaz, pero no se puede usar con métodos.
Una clase puede extender solo una clase abstracta.	Una clase puede implementar más de una interfaz.
Las variables no son definitivas por defecto. Puede contener variables no finales.	Las variables son finales por defecto en una interfaz.
Una clase abstracta puede proporcionar la implementación de una interfaz.	Una interfaz no puede proporcionar la implementación de una clase abstracta.
Puede tener métodos con implementaciones.	Proporciona una abstracción absoluta y no puede tener implementaciones de métodos.
Puede tener modificadores de acceso públicos, privados, estáticos y protegidos.	Los métodos son implícitamente públicos y abstractos en la interfaz de Java.
No admite herencias múltiples.	Es compatible con herencias múltiples.
Es ideal para la reutilización del código y la perspectiva de la evolución.	Es ideal para la declaración de tipo.

Ejercicio :

Escenario:

En una ferretería se venden diferentes tipos de herramientas. Todas las herramientas tienen un nombre, un precio y una marca. Sin embargo, algunas herramientas son eléctricas y otras son manuales. Para este ejercicio, consideraremos dos tipos de herramientas: herramientas eléctricas y herramientas manuales.

Diseño

Clase abstracta Herramienta:

Esta clase representará cualquier herramienta vendida en la ferretería.
Tendrá atributos comunes como nombre, precio y marca.
Tendrá métodos abstractos que las clases derivadas deben implementar, como esElectrica().

Clase HerramientaElectrica:

Esta clase heredará de Herramienta.
Tendrá un atributo adicional para indicar la potencia en vatios.
Implementará el método esElectrica().

Clase HerramientaManual:

Esta clase también heredará de Herramienta.
Las herramientas manuales no son eléctricas, por lo que el método esElectrica() simplemente devolverá false.

```java
// Clase abstracta Herramienta
public abstract class Herramienta {
    protected String nombre;
    protected double precio;
    protected String marca;

    public Herramienta(String nombre, double precio, String marca)
{
        this.nombre = nombre;
        this.precio = precio;
        this.marca = marca;
    }

    // Método abstracto que las clases derivadas deben implementar
    public abstract boolean esElectrica();
}
```

```java
// Clase HerramientaElectrica que hereda de Herramienta
public class HerramientaElectrica extends Herramienta {
    private int potencia; // Potencia en vatios

    public HerramientaElectrica(String nombre, double precio, String marca,
int potencia) {
        super(nombre, precio, marca);
        this.potencia = potencia;
    }

    @Override
    public boolean esElectrica() {
        return true;
    }
}
```

```
// Clase HerramientaManual que hereda de
Herramienta
public class HerramientaManual extends Herramienta
{

    public HerramientaManual(String nombre, double
precio, String marca) {
        super(nombre, precio, marca);
    }

    @Override
    public boolean esElectrica() {
        return false; // Las herramientas manuales no
son eléctricas
    }
}
```

```
public class FerreteriaMain {
    public static void main(String[] args) {
        // Crear una herramienta eléctrica
        HerramientaElectrica taladro = new HerramientaElectrica("Taladro",
50.00, "Bosch", 500);

        // Crear una herramienta manual
        HerramientaManual martillo = new HerramientaManual("Martillo", 10.00,
"Stanley");

        // Mostrar información
        mostrarInfoHerramienta(taladro);
        mostrarInfoHerramienta(martillo);
    }

    public static void mostrarInfoHerramienta(Herramienta herramienta) {
        System.out.println("Nombre de la herramienta: " + herramienta.nombre);
        System.out.println("Precio: $" + herramienta.precio);
        System.out.println("Marca: " + herramienta.marca);
        if (herramienta.esElectrica()) {
            System.out.println("Esta herramienta es eléctrica.");
        } else {
            System.out.println("Esta herramienta es manual.");
        }
        System.out.println("-----------------------------");
    }
}
```

Practica

- Implementar el ejercicio anterior y desarrollar una clase adicional (a su criterio) que herede de herramienta.
- No olvidar sobrescribir el método de acuerdo a la clase implementada.

Conclusiones

- Las clases abstractas en programación orientada a objetos sirven como una base común para otras clases, permitiendo definir comportamientos (métodos) que las clases derivadas deben implementar. Estas clases no pueden ser instanciadas directamente, lo que significa que su principal función es ser extendidas por otras clases.
- Mientras que las clases abstractas se centran en proporcionar una base común y permiten la herencia simple, las interfaces definen contratos que las clases pueden implementar, permitiendo la herencia múltiple. Esto hace que las interfaces sean ideales para definir comportamientos que pueden ser compartidos por clases que no están necesariamente relacionadas en términos de jerarquía de herencia.

Cierre

¿Qué hemos aprendido hoy?

Bibliografía

- MORENO PÉREZ, J. "Programación orientada a objetos". RA-MA Editorial. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=31933
- Vélez Serrano, José. "Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java". Dykinson. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=36368

Universidad Tecnológica del Perú