UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

Programacion Orientada a Objetos

Sesión 6: Clases abstractas. Creación y ampliación de clases abstractas

Recordando…

¿Tienen alguna consulta o duda sobre la clase previa?

Universidad Tecnológica del Perú

Agenda

- Introducción a Clases Abstractas
- Creación y Ampliación de Clases Abstractas

Logro de la sesión

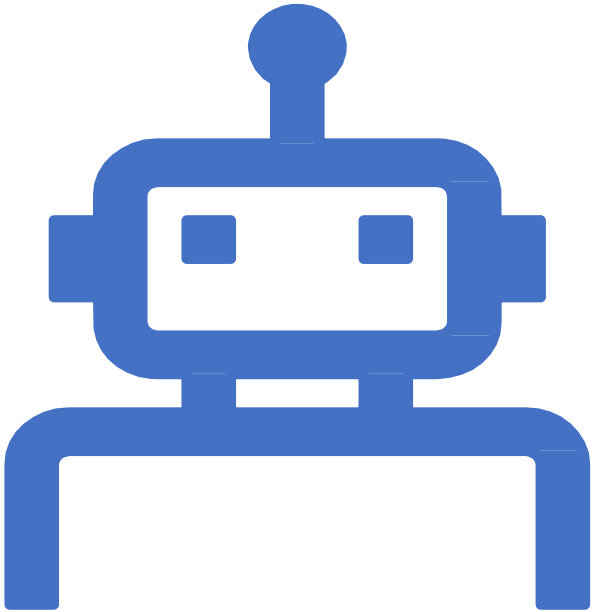
Al finalizar la sesión el estudiante entiende la importancia de las clases abstractas y su propósito en la creación de estructuras de código flexibles y reutilizables.

Utilidad

Las clases abstractas en Java son una herramienta fundamental para la creación de jerarquías de clases, la definición de interfaces comunes y la promoción de la reutilización y el mantenimiento del código. Permiten una organización clara y modular del código, lo que resulta en un diseño más robusto y mantenible de las aplicaciones.

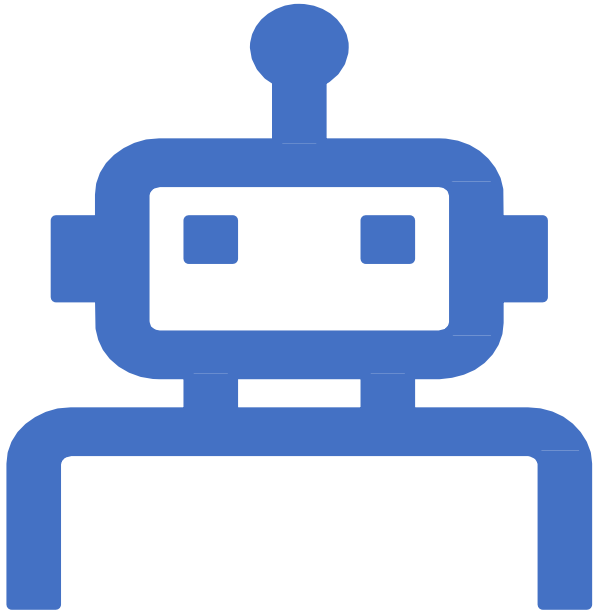
Clases Abstractas

- Una clase abstracta en Java es una clase que no puede ser instanciada, es decir, no puedes crear un objeto de una clase abstracta directamente.
- Estas clases se definen con la palabra clave **abstract** y pueden tener **métodos abstractos**, que son métodos sin cuerpo (sin implementación).
- La idea detrás de una clase abstracta es ***proporcionar una base para otras clases***, permitiendo que otras clases hereden de ella y utilicen sus características, pero obligándolas a proporcionar implementaciones para los métodos abstractos.



Clases Abstractas

- La estructura es prácticamente igual, ya que poseen nombre, atributos y métodos, pero para que una clase sea abstracta la condición es que al menos uno de sus métodos sea abstracto (se le agrega la palabra reservada **abstract** y no se especifica el cuerpo del método).....su uso depende de la aplicación del concepto de Herencia y adicionaremos a la estructura básica de clase la palabra reservada **abstract.**



Clases Abstractas

https://www.youtube.com/watch?v=EFL2U4MsNZw

Características principales:

- No puedes instanciar una clase abstracta.
- Una clase abstracta puede tener métodos abstractos y no abstractos.
- Si una clase tiene al menos un método abstracto, entonces la clase debe ser declarada abstracta.

- Las subclases que heredan de una clase abstracta deben proporcionar implementaciones para todos los métodos abstractos de la superclase, a menos que la subclase también sea abstracta.

¿Cuando Utilizarlas?

Al trabajar clases y métodos abstractos, no solo mantenemos nuestra aplicación más organizada y fácil de entender, sino que también al no poder instanciar una clase abstracta nos aseguramos de que las propiedades específicas de esta, solo estén disponibles para sus clases hijas.

Con las Clases Abstractas lo que hacemos es definir un proceso general que luego será implementado por las clases concretas que hereden dichas funcionalidades.....¿Que?..... es decir, si tengo una clase que hereda de otra Abstracta, estoy obligado a poner en el código, todos los métodos abstractos de la clase padre, pero esta vez serán métodos concretos y su funcionalidad o cuerpo será definido dependiendo de para que la necesite, de esa manera si tengo otra clase que también hereda del mismo padre, implementaré el mismo método, pero con un comportamiento distinto.

Creación y Ampliación de Clases Abstractas

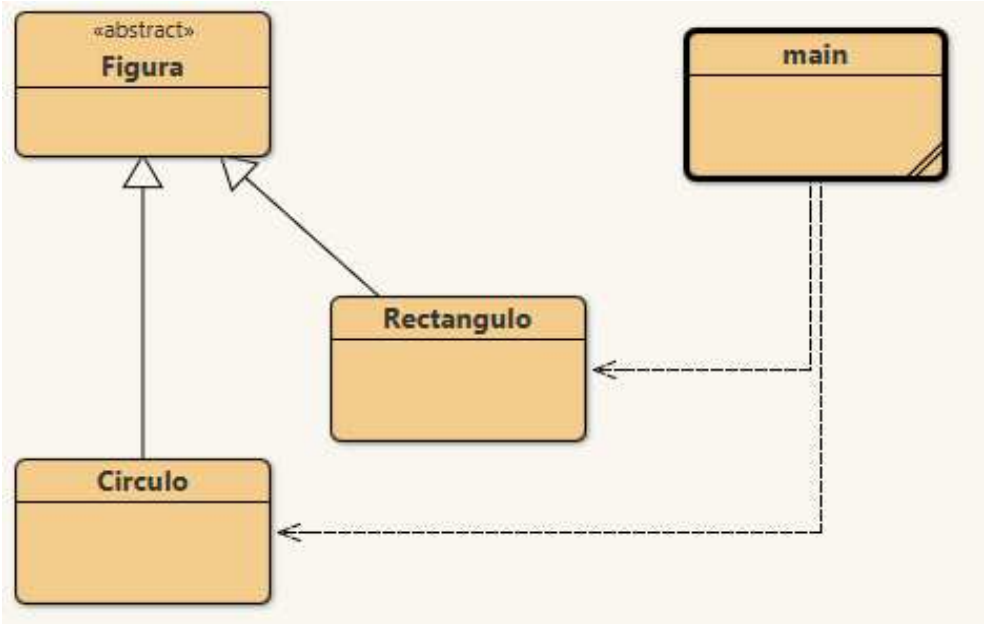
Creación: Para crear una clase abstracta, utiliza la palabra clave **abstract**:

```
abstract class NombreClaseAbstracta {
    // campos y métodos
}
```

Ampliación: Cuando una clase normal (no abstracta) extiende o hereda de una clase abstracta, debe implementar todos los **métodos abstractos** de la clase abstracta:

```
class ClaseConcreta extends
NombreClaseAbstracta {
    // implementación de todos los métodos
abstractos
}
```

Ejercicio 1:



En el código se define una **clase abstracta Figura** con métodos abstractos **area() y perimetro()**, así como las clases concretas **Circulo y Rectang**ulo que extienden de la clase Figura e implementan los métodos abstractos.

Sin embargo, hay un pequeño error: los métodos abstractos deben tener un modificador de acceso explícito, ya sea public, protected, o package-private (sin especificar ningún modificador). Por lo tanto, debes agregar el modificador public a los métodos abstractos en la clase abstracta Figura.

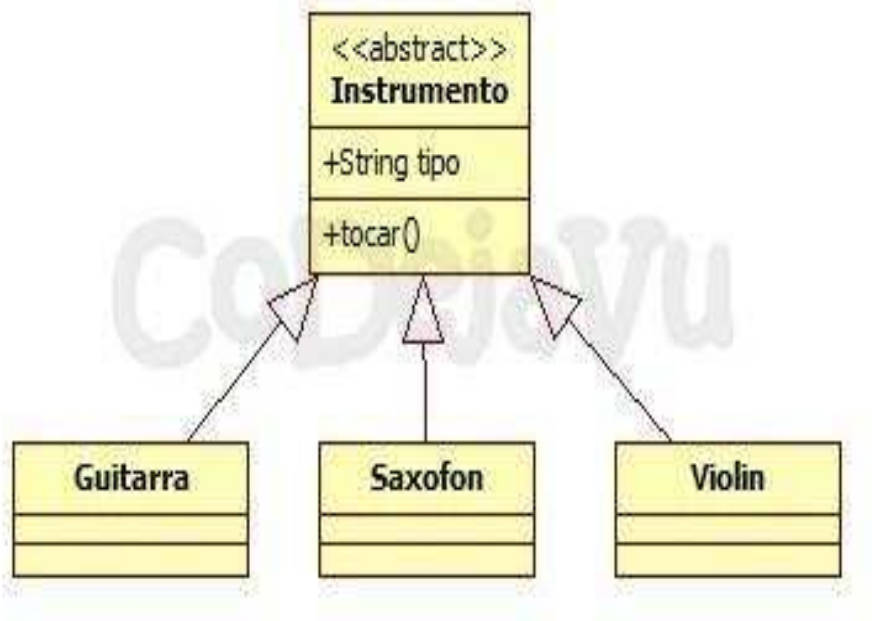
Ejercicio 1: Clase abstracta Figura y clases concretas Circulo y Rectangulo.

```
// Clase abstracta Figura
abstract class Figura
{
public abstract double area();
public abstract double perimetro();
 }
// Clase Circulo que hereda de Figura
public class Circulo extends Figura {
   double radio;

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

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

   @Override
   public double perimetro() {
      return 2 * Math.PI * radio;
   }

}
```

```
// Clase Rectangulo que hereda de Figura
public class Rectangulo extends Figura {
   double largo, ancho;

   Rectangulo(double largo, double ancho) {
      this.largo = largo;
      this.ancho = ancho;
   }

   @Override
   public double area() {
      return largo * ancho;
   }

   @Override
   public double perimetro() {
      return 2 * (largo + ancho);
   }


}
```

```
public class main
{
    public static void main(String[] args) {
      Circulo circulo = new Circulo(5);
      System.out.println("Área del círculo: " + circulo.area());
      System.out.println("Perímetro del círculo: " +
circulo.perimetro());


      Rectangulo rectangulo = new Rectangulo(4, 6);
      System.out.println("Área del rectángulo: " +
rectangulo.area());
      System.out.println("Perímetro del rectángulo: " +
rectangulo.perimetro());
   }
   }
```

Ejercicio 2:

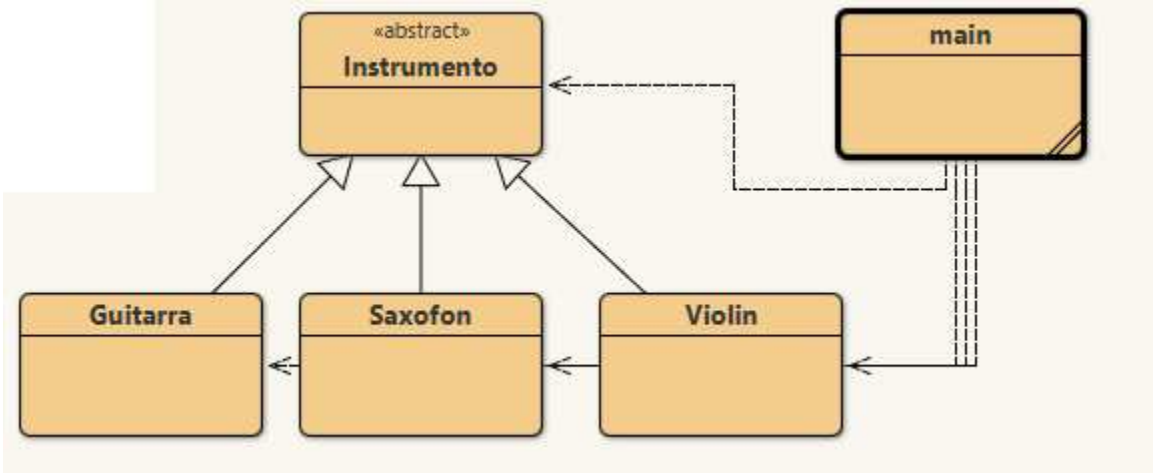


En el diagrama vemos una clase Abstracta Instrumento, la cual posee una propiedad tipo y un método abstracto tocar(), vemos también las clases hijas Guitarra, Saxofon y Violin que para este ejemplo solo utilizaremos (mediante la herencia) las propiedades de la clase Padre.

.

Ejercicio 2:

Todos los instrumentos musicales se pueden tocar, por ello creamos este método abstracto, ya que es un proceso común en todos los instrumentos sin importar el detalle de como se tocan, pues sabemos que una guitarra no se toca de la misma manera que el saxofón, así al heredar de la clase Instrumento, todas sus clases hijas están obligadas a implementar este método y darle la funcionalidad que le corresponda...



Ejercicio 2:

```
//Clase Abstracta Instrumento
abstract class Instrumento{
public String tipo;
 public abstract void tocar();
}

//Clase Concreta Guitarra, hija de Instrumento
class Guitarra extends Instrumento {
public Guitarra(){
  tipo="Guitarra";
 }
 public void tocar() {
  System.out.println("Tocar La Guitarra");
 }
}

//Clase Concreta Violin, hija de Instrumento
class Violin extends Instrumento {
public Violin(){
  tipo="Violin";
 }
 public void tocar() {
  System.out.println("Tocar El violin");
 }
}

//Clase Concreta Saxofon, hija de Instrumento
class Saxofon extends Instrumento {
 public Saxofon(){
  tipo="Saxofon";
 }
 public void tocar() {
  System.out.println("Tocar el Saxofon");
 }
}
public class Principal {
 public static void main(String arg[]){
  /**Objeto miGuitarra de tipo Instrumento */
  Instrumento miGuitarra= new Guitarra();
  System.out.println("Instrumento : "+miGuitarra.tipo);
  miGuitarra.tocar();
  System.out.println();
  /**Objeto miSaxofon de tipo Instrumento */
  Instrumento miSaxofon= new Saxofon();
  System.out.println("Instrumento : "+miSaxofon.tipo);
  miSaxofon.tocar();
  System.out.println();
  /**Objeto miViolin de tipo Instrumento */
  Instrumento miViolin= new Violin();
  System.out.println("Instrumento : "+miViolin.tipo);
  miViolin.tocar();

 }

}
```

Practica Guiada

Implementa los ejercicios anteriormente realizados en el programa java.

Conclusiones

- Las clases abstractas son fundamentales en la programación orientada a objetos en Java.
- Permiten definir plantillas para clases, asegurando una estructura y comportamiento consistentes.
- La correcta implementación y extensión de clases abstractas es esencial para el diseño eficiente de sistemas y aplicaciones.

Cierre

¿Que hemos aprendido hoy?

Universidad Tecnológica del Perú

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