Programación

Unidad 9: Clases y métodos abstractos e Interfaces

Unidad 9

- Clases abstractas
- Métodos astractos
- Interfaces
- Interfaces vs clases abstractas

Clases abstractas

Unidad 9: Clases y métodos abstractos e Interfaces

Clases abstractas

A menudo existen clases que sirven para **definir un tipo genérico** pero que **no tiene sentido** instanciar (crear objetos de ella). Por ejemplo, puede tener sentido instanciar un Círculo pero a lo mejor no instanciar una Figura, porque… ¿qué figura es? ¿cuál es su área? ¿y su perímetro?

Estas clases pueden estar siendo usadas simplemente **para agrupar bajo un mismo tipo a otras clases**, o para contener **código reutilizable**, o para forzar un API a sus subclases…

La clases se definen como abstractas mediante la keyword: **abstract**.

Declaración de una clase abstracta:

- `modificador_acceso abstract class nom_clase {}`

Ejemplo:

- `public abstract class MiClase {};`

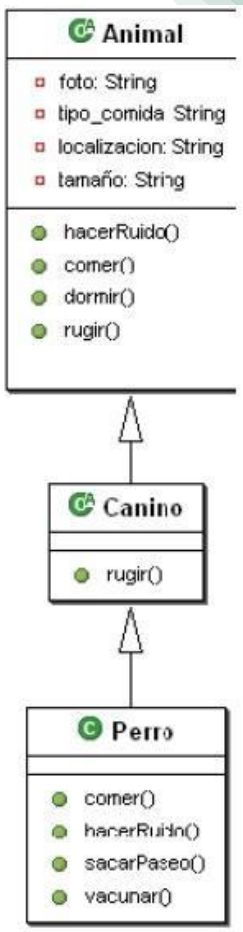
Unidad 9: Clases y métodos abstractos e Interfaces

Clases abstractas

Ejemplo

```
public abstract class Animal
{
  ........
}
public abstract class Canino extends Animal
{
  .......
}
public class Perro extends Canino
{
  .......
}
public class Test
{
  public static void main(String[] args)
  {
    Canino c;
    c = new Canino();
    c.rugir();
  }
}
```

No compila. Canino es una clase abstracta



Métodos abstractos

Unidad 9: Clases y métodos abstractos e Interfaces

Métodos abstractos

Además de clases abstractas, también podemos tener **métodos abstractos**. Una clase abstracta implicaba que **tenía que ser heredada**. **No podía ser instanciada**.

Un método abstracto implica que **tiene que ser sobreescrito**. No está implementado.

Una **clase** con **uno o varios métodos abstractos** tiene que ser declarada **abstracta**.

No obstante, una clase abstracta **no tiene porqué tener métodos abstractos**.

Unidad 9: Clases y métodos abstractos e Interfaces

Métodos abstractos

Los métodos se definen como abstractos mediante la keyword: **abstract**.

Declaración de un método abstracto:

```
modif_acceso abstract tipo_retorno nombre([tipo param,…]);
```

Ejemplo:

```
public abstract void miMetodo();
```

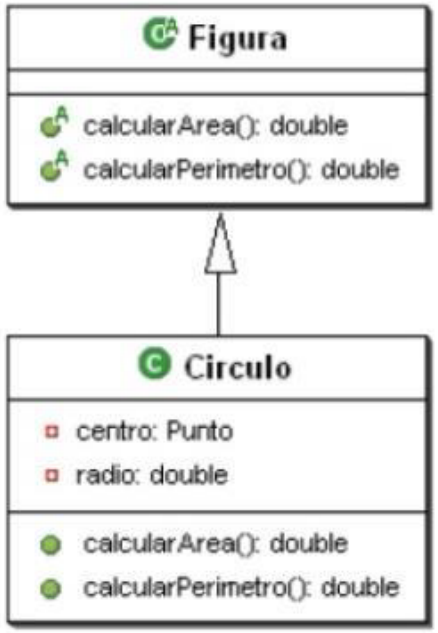
El objetivo de un método abstracto es **forzar una interfaz** (API) pero no una implementación.

Unidad 9: Clases y métodos abstractos e Interfaces

Métodos abstractos

Ejemplo

```
public abstract class Figura
{
  public abstract double calcularArea();
  public abstract double calcularPerimetro();
}

public class Circulo extends Figura
{
  private Punto centro = null;
  private double radio = 0.0;

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

  public double  calcularPerimetro()
  {
    return 2*Math.PI*radio;
  }
}
```



Interfaces

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces

Los interfaces en Java nos solucionan en parte la **no existencia de la herencia múltiple**; habilitando así las posibilidades del polimorfismo en la herencia múltiple sin los problemas que esta conlleva.

Los interfaces son un tipo de clase especial que no implementa ninguno de sus métodos. **Todos son abstractos**. Por tanto no se pueden instanciar.

La declaración de un interface Java se realiza mediante la keyword: **interface** seguido de su nombre.

- `modif_acceso interface nom_interfaz [extends interface1[,interface2…]]`

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces

Ejemplo:

```
public interface InterfaceGrupo extends Interface1, Interface2, Interface3 {
    // declaración de constantes
    double E = 2.718282;
    // métodos
    void doSomething (int i, double x);
    int doSomethingElse(String s);
}
```

Un interface **puede heredar de otros interfaces** (de tantos como queramos, a diferencia de las clases).

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces

De los interfaces también se hereda, aunque se suele decir implementa. Y se realiza mediante la keyword: **implements**.

Declaración de la herencia:

```
modif_acceso class nom_clase implements nom_interface[,nom_int….] {}
```

Ejemplo:

```
public class MiClase implements MiInterface {};
```

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces

Una **clase** puede **implementar múltiples interfaces**.

Una **clase** puede heredar de otra clase y a la vez **implementar múltiples interfaces**.

Un **interface** puede también definir **constantes**.

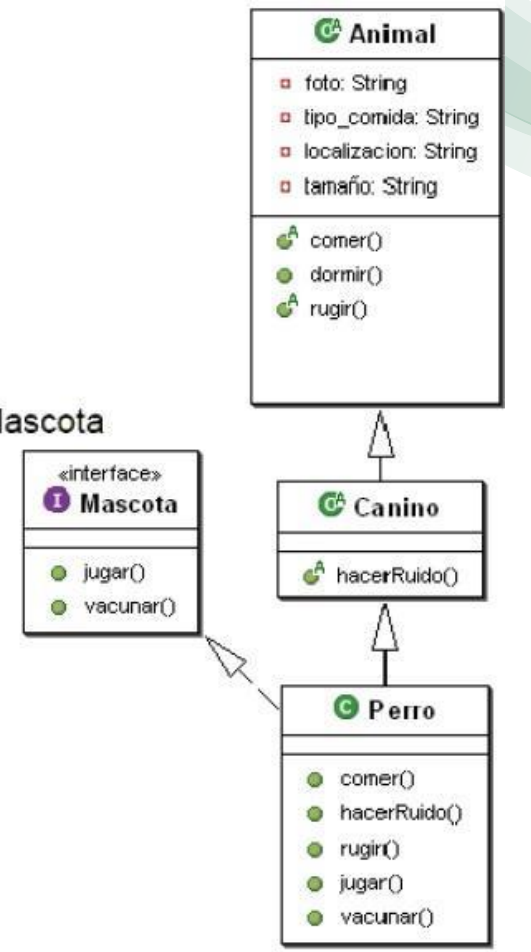
Si una **clase que implementa un interface**, **no implementa todos los métodos** de este, deberá ser **definida como abstracta**.

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces

Ejemplo

```
public interface Mascota
{
  public abstract void jugar();
  public abstract void vacunar();
}

public class Perro extends Canino implements Mascota
{
  public void comer() { … }

  public void hacerRuido() { … }

  public void rugir() { … }

  public void jugar() { … }

  public void vacunar() { … }
}
```



Interfaces vs. Clases abstractas

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces vs. Clases abstractas

Un **interface** no puede **implementar ningún método**.

Una **clase** puede **implementar n interfaces** pero solo puede **heredar de una sola clase**.

Un **interface no forma parte de la jerarquía de clases**. Clases dispares pueden implementar el mismo interface.

El **objetivo** de un **método abstracto** es **forzar una interfaz** (API) pero no una implementación.

Unidad 9: Clases y métodos abstractos e Interfaces

Interfaces vs. Clases abstractas

Haremos una **clase que no herede de nadie** cuando la clase no pase la prueba de **Es-Un**.

Haremos una **subclase** cuando necesitemos hacer una **especialización** de la superclase mediante sobreescritura o añadiendo nuevos métodos.

Haremos una **clase abstracta** cuando queramos definir un **grupo genérico de clases** y además tengamos algunos **métodos implementados que reutilizar**. También **cuando no queramos que nadie instancie** dicha clase.

Haremos un **interface** cuando queramos definir un **grupo genérico de clases y no tengamos métodos implementados que reutilizar**. O cuando nos veamos forzados por la **falta de herencia múltiple** en Java.