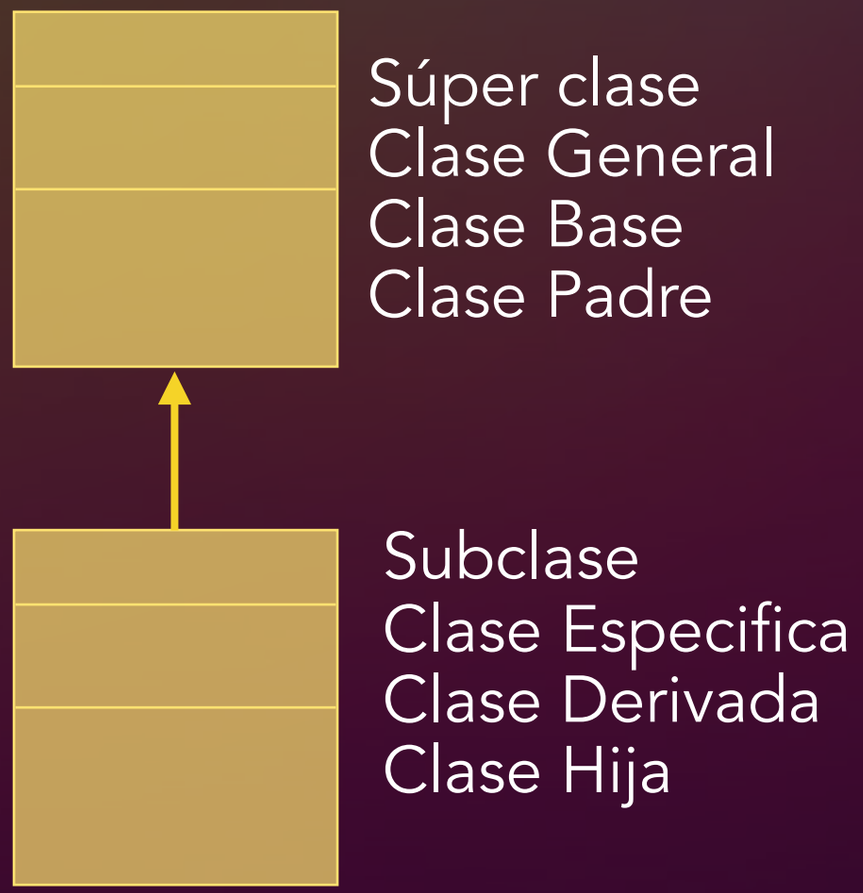
4. Herencia y Polimorfismo

Herencia

- La herencia es el proceso que implica la creación de clases a partir de clases ya existentes, permitiendo, además, agregar más funcionalidades.
- Relación jerárquica



Herencia ¿para qué?

- Promover la reutilización de código
- Optimizar la modificación o integración de código
- Extender la descripción y definición de clases
- Favorecer la aplicación del polimorfismo
- Permitir la especialización del comportamiento en el modelado y la programación de los sistemas.

Herencia simple

- Para heredar en Java se utiliza la palabra reservada **extends** al momento de definir la clase, su sintaxis es la siguiente:
- *[modificadores] class ClaseHija* ***extends*** *ClasePadre*

Animal.java

```java
public class Animal{
    public String nombre;
    public Animal(String nombre){
        this.nombre = nombre;
    }
    public void comer(){
        System.out.println("El "+nombre+" está comiendo");
    }
}
```

Linea 10, Columna 10 Java

Perro.java

```java
public class Perro extends Animal {
    public Perro(){
        super("perro");
    }
    public void ladrar(){
        System.out.println("El "+nombre+" está ladrando");
    }
}
```

Linea 10, Columna 10 Java

Perrera.java

```java
public class Perrera{
    public static void main(String[] args) {
        Perro fido = new Perro();
        fido.ladrar();
        fido.comer();
    }
}
```

Linea 10, Columna 10

```
mac-xkn:super axcana$ java Perrera
El perro está ladrando
El perro está comiendo
mac-xkn:super axcana$
```

Super

- ***super*** es la palabra reservada que permite acceder a los atributos y métodos de la clase padre o superclase.
- Con ***super*** podemos invocar ya sea un método constructor o miembros de clase que venga de la clase desde la que se hereda.

Animal.java

```java
public class Animal{
  public String nombre;
  public Animal(String nombre){
    this.nombre = nombre;
  }
  public void comer(){
    System.out.println("El "+this.nombre+" está comiendo");
  }
}
```

Linea 10, Columna 10 Java

Perro.java

```java
public class Perro extends Animal {
  public Perro(){
    super("perro");
  }
  public void ladrar(){
    System.out.println("El "+super.nombre+" está ladrando");
  }
  public void ladrarYComer(){
    this.ladrar();
    super.comer();
  }
}
```

Linea 10, Columna 10 Java

Perrera.java

```java
public class Perrera{
    public static void main(String[] args){
        Perro fido = new Perro();
        fido.ladrarYComer();
    }
}
```

Linea 10, Columna 10

```
mac-xkn:super axcana$ java Perrera
El perro está ladrando
El perro está comiendo
mac-xkn:super axcana$
```

Y si hacemos que se mueva…

- ¿Funciona correctamente?
- ¿es la mejor solución?
- ¿qué ocurre si heredamos animal a un ave?
- ¿qué ocurre si heredamos animal a una víbora?
- ¿que ocurre si heredamos animal a una foca?

Animal.java

```java
public class Animal{
    public String nombre;
    public Animal(String nombre){
        this.nombre = nombre;
    }
    public void comer(){
        System.out.println("El "+nombre+" está comiendo");
    }
    public void moverse(){
        System.out.println("El "+nombre+" está corriendo");
    }
}
```

Linea 10, Columna 10 Java

Perrera.java

```java
public class Perrera{
    public static void main(String[] args) {
        Perro fido = new Perro();
        fido.ladrar();
        fido.comer();
        fido.moverse();
    }
}
```

Linea 10, Columna 10 Java

Abstract class

Herencia

Animal.java

```java
public abstract class Animal{
    public String nombre;
    public Animal(String nombre){
        this.nombre = nombre;
    }
    public void comer(){
        System.out.println("El "+nombre+" está comiendo");
    }
    public abstract void moverse();
}
```

Linea 10, Columna 10 Java

Ave.java

```java
public class Ave extends Animal{

    public Ave(){
        super("Ave");
    }
    public void moverse(){
        System.out.println("El "+nombre+" está volando");
    }

}
```

Linea 10, Columna 10 Java

Veterinaria.java

```java
public class Veterinaria{
    public static void main(String[] args) {
        Perro fido = new Perro();
        fido.moverse();
        fido.comer();
        Ave piolin = new Ave();
        piolin.comer();
        piolin.moverse();
    }
}
```

herencia — -bash — 44×12

```
mac-xkn:herencia axcana$ java Veterinaria
El perro está corriendo
El perro está masticando
El Ave está comiendo
El Ave está volando
mac-xkn:herencia axcana$
```

Linea 10, Columna 10 Java

Clases abstractas

- Una clase abstracta es una clase de la que no se pueden crear objetos, debido a que define la existencia de métodos, pero no su implementación.
  - Pueden contener métodos abstractos y métodos concretos.
  - Pueden contener atributos.
  - Pueden heredar de otras clases.
- Para declarar una clase abstracta en Java solo es necesario anteponer la palabra reservada abstract antes de palabra reservada class.

Sobrecarga vs Sobrescritura

Override

- La sobrescritura es un concepto que tiene sentido en la herencia y se refiere al hecho de volver a definir un método heredado.

Animal.java

```java
public abstract class Animal{
    public String nombre;
    public Animal(String nombre){
        this.nombre = nombre;
    }
    public void comer(){
        System.out.println("El "+nombre+" está comiendo");
    }
    public abstract void moverse();
}
```

Linea 10, Columna 10 Java

Overload

- La sobrecarga sólo tiene sentido en la clase misma, se pueden definir varios métodos con el mismo nombre, pero con diferentes tipos, orden y número de parámetros.

Perro.java

```java
public class Perro extends Animal {
    public Perro(){
        super("perro");
    }
    public void ladrar(){
        System.out.println("El "+nombre+" está ladrando");
    }
    public void moverse(){
        System.out.println("El "+nombre+" está corriendo");
    }
    public void comer(){
        System.out.println("El "+nombre+" está masticando");
    }
    public void ladrar(String magnitud){
        System.out.println("El "+nombre+" ladra "+magnitud);
    }
}
```

Linea 10, Columna 10 Java

IS-A vs HAS-A

- La relación IS-A (es un) se basa en la herencia y permite afirmar que un objeto es de una clase en particular.
- La relación HAS-A (tiene un) es un concepto de Abstracción y encapsulamiento como composición. Se basa en el uso más que en la herencia.

IS-A

- Es posible afirmar que un objeto es de una clase.
- Caballo hereda de Animal,
  → Caballo ***es un*** Animal.
- Purasangre hereda de Caballo,
  → Purasangre ***es un*** Caballo.

Animal.java

```java
public class Animal {

}
```

Linea 10, Columna 10 Java

Caballo.java

```java
public class Caballo extends Animal{

}
```

Linea 10, Columna 10 Java

Purasangre.java

```java
public class Purasangre extends Caballo{

    public static void main(String args[]){
    }
}
```

Linea 10, Columna 10 Java

HAS-A

- El código en la clase **tiene** como atributo **una** referencia de otra clase.
- → Un Caballo ***tiene una*** SillaDeMontar, debido a que cada instancia de Caballo tendrá una referencia hacia una SillaMontar.

Animal.java

```
public class Animal {

}
```

Linea 10, Columna 10 Java

Caballor.java

```
public class Caballo extends Animal{
   private SillaDeMontar miSilla;
}
```

Linea 10, Columna 10 Java

SillaDeMontar.java

```
class SillaDeMontar {

}
```

Linea 10, Columna 10 Java

Interfaz :: Interface

- Es una colección de **métodos abstractos** y **propiedades constantes**.
- Especifica qué se debe hacer pero no su implementación.
- Serán las clases quienes describan la lógica del comportamiento de los métodos mediante la implementación de las interfaces.

Interface

- Para crear una interfaz en Java, se utiliza la palabra reservada ***interface*** en lugar de ***class***.
- Los métodos que se declaran una interfaz son siempre **públicos** y **abstractos**.
- Una interfaz puede contener atributos, pero estos son siempre **públicos**, **estáticos** y **finales**.
- Las interfaces pueden ser implementadas por cualquier clase. La clase que implementa una interfaz está obligada a implementar los métodos que la interfaz declaró.

LaInterfaz.java

```java
public interface LaInterfaz {

        //declaración del método;
        tipoRetorno metodo([Parametros]);

}
```

Linea 10, Columna 10 Java

LaClase.java

```java
public class LaClase implements LaInterfaz{

    public tipoRetorno metodo([Parametros]){
        //implementación del método;
    }

}
```

Linea 10, Columna 10 Java

Interface

A.java

```java
interface A{
    public void alfa();
}
```

Linea 10, Columna 10 Java

C.java

```java
class C implements A{
    public void alfa(){
        System.out.println("α");
    }
    public void omega(){
        System.out.println("ω");
    }
}
```

Linea 10, Columna 10 Java

...

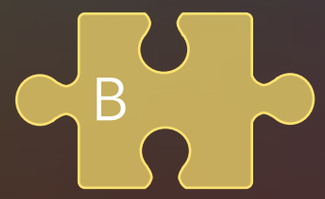
```java
C objC = new C();
objC.alfa(); ✓
objC.omega(); ✓

A objA = new C();
objA.alfa(); ✓
objA.omega(); ✗
```

Linea 10, Columna 10 Java

Interface vs abstract class

abstract* class

abstract* class



*declara y define métodos

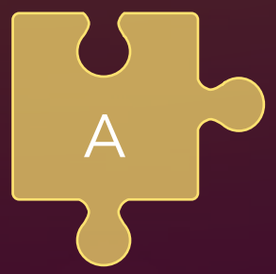
class C extends A ✓

Herencia Simple

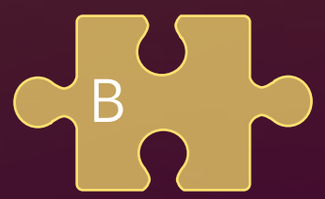
class C extends A, B ✗

Herencia Múltiple

interface*



interface*



*sólo declara métodos

class C implements A ✓

class C implements A, B ✓

Implementación múltiple

- Una clase puede implementar múltiples interfaces y dentro del cuerpo de la clase.
- Deben implementarse todos los métodos de las interfaces que se implementen.

untitled

```
public class NombreClase implements Interfaz1, Interfaz2, ..., InterfazN {

    // Implementar método(s) de la Interfaz1
    // Implementar método(s) de la Interfaz2
    // ...
    // Implementar método(s) de la InterfazN


}
```

Linea 10, Columna 10 Java

Herencia múltiple entre interfaces

- Las interfaces pueden heredar de otras interfaces.
- Una interfaz puede heredar de una o más interfaces aplicando así herencia múltiple.
- La interfaz que hereda de otras interfaces posee todos los métodos definidos en ellas.

untitled

```
public interface NombreInterfaz extends Interfaz1, Interfaz2, ..., InterfazN {


     tipo nombreMetodosInterfaz1([Parametros]);
     tipo nombreMetodosInterfaz2([Parametros]);
     …
     tipo nombreMetodosInterfazN([Parametros]);


}
```

Linea 10, Columna 10 Java

Atributos en interfaces

- Todos los datos miembros que definidos en una interfaz son públicos, estáticos y finales.
- Las interfaces pueden utilizarse para crear grupos de constantes que pueden ser llamados sin crear una instancia.

DiaSemana.java

```
public interface DiaSemana{
    int UNO=1, DOS=2, TRES=3;
    int CUATRO=4, CINCO=5;
    int SEIS=6, SIETE=7;
    String NOMBRES[] = {"Lunes",
    "Martes","Miércoles","Jueves",
    "Viernes","Sabado","Domingo"};
}
```

Linea 10, Columna 10 Java

MiSemana.java

```
public class MiSemana{
  public static void main(String[] args) {
    System.out.print("El día "+DiaSemana.DOS+" es: ");
    System.out.println(DiaSemana.NOMBRES[DiaSemana.DOS]);
  }
}
```

Linea 10, Columna 10 Java

interfaceSemana — -bash — 54×11

```
mac-xkn:interfaceSemana axcana$ javac DiaSemana.java
mac-xkn:interfaceSemana axcana$ javac MiSemana.java
mac-xkn:interfaceSemana axcana$ java MiSemana
El día 2 es: Miércoles
mac-xkn:interfaceSemana axcana$
```

default en interfaces

- Con los métodos default se proporciona una implementación por defecto de un método
- Para agregar una nueva “funcionalidad” no es necesario que las clases hijas de la interfaz definan dicho método.

Animal.java

```java
public interface Animal {
    default void moverse(){
        System.out.println(“Animal moviéndose");
    }
}
```

Linea 10, Columna 10 Java

Acuario.java

```java
public class Acuario {

  public static void main(String args[]){
     Delfin flipper = new Delfin();
     flipper.moverse();
  }

}
```

Delfin.java Java

Delfin.java

```java
public class Delfin implements Animal {

}
```

Linea 10, Columna 10 Java

Defender Methods

- Extensión Virtual de Métodos (Defender Methods)
- Sobrescirtura entre Interfaces (Overridding)

Animal.java

```
public interface Animal {
    default void moverse(){
        System.out.println("Animal moviéndose");
    }
}
```

Linea 10, Columna 10 Java

Mamifero.java

```
public interface Mamifero {
    default void moverse(){
        System.out.println("Mamífero moviéndose");
    }
}
```

Linea 10, Columna 10 Java

Acuario.java

```
public class Acuario {

  public static void main(String args[]){
     Delfin flipper = new Delfin();
     flipper.moverse();
  }

}
```

Linea 10, Columna 10 Java

Delfin.java

```
public class Delfin implements Animal,Mamifero {
    public void moverse(){
        System.out.println("Delfín nada");
    }
}
```

Linea 10, Columna 10 Java

Métodos default con la misma firma

- Cuando una clase implementa dos interfaces que tienen definido un método default con la misma firma (nombre método y diferentes tipos, orden y número de argumentos) es necesario redefinir (sobrescribir) el método e indicar a que interfaz llama.

Animal.java

```java
public interface Animal {
    default void moverse(){
        System.out.println("Animal moviéndose");
    }
}
```

Linea 10, Columna 10 Java

Mamifero.java

```java
public interface Mamifero {
    default void moverse(){
        System.out.println("Mamífero moviéndose");
    }
}
```

Linea 10, Columna 10 Java

Acuario.java

```java
public class Acuario {

  public static void main(String args[]){
     Delfin flipper = new Delfin();
     flipper.moverse();
  }

}
```

Linea 10, Columna 10 Java

Delfin.java

```java
public class Delfin implements Animal,Mamifero {
    public void moverse(){
        Animal.super.moverse();
    }
}
```

Linea 10, Columna 10 Java

Interfaz que contiene método default

- Cuando se implementa una interfaz que contiene un método default considera:
  - Si no se menciona el método default, se debe realizar la implementación de la interfaz.
  - El método default se puede volver a hacer abstracto redefiniendo el método como abstract.
  - Se puede redefinir el método default de forma similar a la que se sobrescriben los métodos convencionales.
  - Cuando una clase implementa dos interfaces que tienen definido un método default con la misma firma debe redefinirse el método e indicar la interfaz que llamará.
  - En una interfaz no se puede implementar un método default que sobrescriba algún método de la clase Object, como toString, equals hasCode.

Polimorfismo

- También llamado Poliformismo se refiere a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos y en consecuencia la habilidad de tener diferentes formas.

- Los objetos que utilizan polimorfismo deben tener una definición que les permita responder al mismo mensaje que se les envía, por lo que cualquier objeto que pueda comportarse como más de un **es un** (*IS-A*) puede ser considerado polimórfico.

- ¿todos los objetos en Java pueden ser considerados polimórficos?

Polimorfismo

- La referencia solo puede ser solo de un tipo y, una vez declarado, el tipo no puede ser cambiado.
- Una referencia es una variable, por lo tanto, a la variable se le puede reasignar otro objeto, a menos que la referencia sea declarada como final.
- El tipo de una referencia determina los métodos que pueden ser invocados del objeto al que referencia, es decir, solo se pueden ejecutar los métodos definidos en el tipo de la referencia.

Paleta.java

Paleta.java

```java
public class Paleta{

    private String palito;
    private String sabor;
    private String envoltura;
    private int precio;
    private String marca;

    public String toString(){
        return "Paleta";
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
```

TixTix.java

```java
public class TixTix extends Paleta{

    public void morder(){
        System.out.println("Mordieron la paleta TixTix");
    }

    public void chupar(){
        System.out.println("Chuparon la paleta TixTix");
    }

    public String toString(){
        return "Paleta TixTix";
    }

}
```

Dulceria.java

```java
public class Dulceria{
    public static void main(String[] args) {

        Paleta tt;

        tt = new TixTix();
        System.out.println(tt);

        tt = new Rocaleta();
        System.out.println(tt);

    }
}
```

Rocaleta.java

```java
public class Rocaleta extends Paleta{

    public void morder(){
        System.out.println("Mordieron la paleta Rocaleta");
    }

    public void chupar(){
        System.out.println("Chuparon la paleta Rocaleta");
    }

    public String toString(){
        return "Paleta Rocaleta";
    }
}
```

Line 0, Column 1 Tab Size: 4 Java

Paleta.java

```java
public abstract class Paleta{

    private String palito;
    private String sabor;
    private String envoltura;
    private int precio;
    private String marca;

    public abstract void morder();

    public String toString(){
        return "Paleta";
    }

    public String getMarca() {
        return marca;
    }
```

TixTix.java

```java
public class TixTix extends Paleta{

    public void morder(){
        System.out.println("Mordieron la paleta TixTix");
    }

    public void chupar(){
        System.out.println("Chuparon la paleta TixTix");
    }

    public String toString(){
        return "Paleta TixTix";
    }

}
```

Dulceria.java

```java
public class Dulceria{
    public static void main(String[] args) {

        Paleta tt;

        tt = new TixTix();
        System.out.println(tt);

        tt = new Rocaleta();
        System.out.println(tt);

    }
}
```

Rocaleta.java

```java
public class Rocaleta extends Paleta{

    public void morder(){
        System.out.println("Mordieron la paleta Rocaleta");
    }

    public void chupar(){
        System.out.println("Chuparon la paleta Rocaleta");
    }

    public String toString(){
        return "Paleta Rocaleta";
    }
}
```

8 characters selected — Tab Size: 4 — Java