



# TEMA 5

## Herencia

V1.3

Manuel Pereira González



## Agenda

- **Introducción**
- Clases Derivadas
  - Implementación
  - Constructores
- Sobreescritura de Métodos
  - Métodos Heredados vs Sobreescritos
  - Métodos static y final
  - Polimorfismo
- Métodos y clases abstractas
- Restricciones de Acceso
- La Clase Object
- Resumen



# Introducción

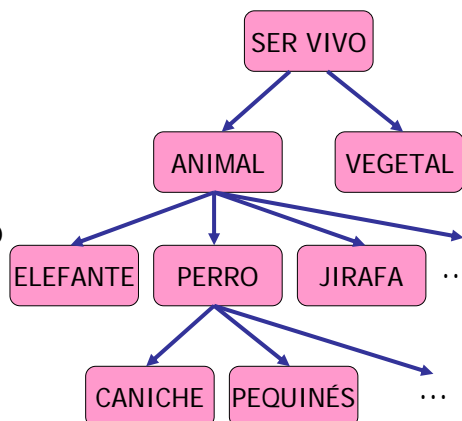


- Herencia
  - Una clase (subclase ó clase hija) **hereda** la estructura de otra (superclase o clase padre), y **concretiza** ciertos aspectos de ésta.
  - La clase hija comprende un **subconjunto** de los objetos de la clase padre.
  - Ej:
    - Clase: **Animal**
      - **Ser vivo que nace, crece, se reproduce y muere**
    - Subclase: **Perro** (subconjunto de animales)
      - **Ser vivo que nace, crece, se reproduce, muere, tiene cuatro patas, dos ojos, ladra, ...**

# Introducción



- Relación de Generalización / Especialización
- Jerarquía definida a través de la **Herencia**:
  - Un CANICHE **es un** PERRO
  - Una PERRO **es un** ANIMAL
  - Un ANIMAL **es un** SER VIVO



# Introducción



- Ventaja fundamental:  
**Reutilización/Centralización de Código**. No es necesario volver a escribir los atributos y métodos de la clase padre en la clase hija.
  - No duplicidad del código
  - Disminuye volumen de código
  - Facilita el mantenimiento
  - Permite el polimorfismo: Tratar con objetos de una clase general sin saber la subclase concreta a la que pertenecen

# Agenda



- Introducción
- **Clases Derivadas**
  - Implementación
  - Constructores
- Sobreescritura de Métodos
  - Métodos Heredados vs Sobreescritos
  - Métodos static y final
  - Polimorfismo
- Métodos y clases abstractas
- Restricciones de Acceso
- La Clase Object
- Resumen



# Clases Derivadas: Implementación



- En java para que una clase herede de otra se utiliza la palabra clave **extends**

```
public class Animal {
    public int diaNacimiento;
    public int mesNacimiento;
    public int anioNacimiento;

    public Animal() {
    }

    public Animal(int diaNacimiento,
        int mesNacimiento, int anioNacimiento) {
        this.diaNacimiento = diaNacimiento;
        this.mesNacimiento = mesNacimiento;
        this.anioNacimiento = anioNacimiento;
    }

    public void morir() {
        System.out.println("Ay, muero");
    }

    public void imprimeNacimiento() {
        System.out.println("Naci el dia " + diaNacimiento
            + " del " + mesNacimiento + " de " + anioNacimiento);
    }
}

public class Perro extends Animal {
    public String raza;

    public Perro(String raza) {
        this.raza = raza;
    }

    public void ladrar() {
        System.out.println("GUAU");
    }
}
```

# Clases Derivadas: Implementación



- Una clase derivada hereda las variables y métodos de la clase padre, además de añadir sus variables y métodos propios

```
public class PruebaAnimales {
    public static void main(String[] args) {
        Animal a1 = new Animal();
        Animal a2 = new Animal(5, 7, 1978);
        Perro p = new Perro("Caniche");

        a1.imprimeNacimiento();
        a2.imprimeNacimiento();
        p.imprimeNacimiento();
        p.ladrar();
        p.morir();
        a1.morir();
        a2.morir();
    }
}
```

```
Command Prompt
C:\Practicas\Programacion>java PruebaAnimales
Naci el dia 0 del 0 de 0
Naci el dia 5 del 7 de 1978
Naci el dia 0 del 0 de 0
GUUU
Ay, muero
Ay, muero
Ay, muero
C:\Practicas\Programacion>
```

# Clases Derivadas: Constructores



- Por defecto desde un constructor de una clase hija se llama al constructor sin argumentos de la clase padre

```
public class Animal {  
    public Animal() {  
        System.out.println(  
            "Soy un animal y estoy naciendo");  
    }  
}  
  
public class Perro extends Animal {  
    public String raza;  
    public Perro(String raza) {  
        System.out.println(  
            "Soy un perro y estoy naciendo");  
        this.raza = raza;  
    }  
}
```

```
public class PruebaAnimales {  
    public static void main(String[] args) {  
        System.out.println("Paso 1");  
        Perro p = new Perro("Caniche");  
        System.out.println("Paso 2");  
        Animal a1 = new Animal();  
        System.out.println("Paso 3");  
    }  
}
```

```
C:\ Command Prompt  
C:\Practicas\Programacion>java PruebaAnimales  
Paso 1  
Soy un animal y estoy naciendo  
Soy un perro y estoy naciendo  
Paso 2  
Soy un animal y estoy naciendo  
Paso 3  
C:\Practicas\Programacion>_
```

# Clases Derivadas: Constructores



- Por defecto desde un constructor de una clase hija se llama al constructor sin argumentos de la clase padre
- Si se desea llamar a otro constructor de la clase padre se utiliza la palabra clave **super**
- Para mantener el encapsulamiento, una clase derivada debe inicializar sus variables específicas en el constructor, y dejar al constructor del padre inicializar las suyas.

# Clases Derivadas: Constructores



```
public class Animal {
    public int diaNacimiento;
    public int mesNacimiento;
    public int anioNacimiento;

    public Animal() {
        System.out.println(
            "Soy un animal y nazco no se cuando");
    }

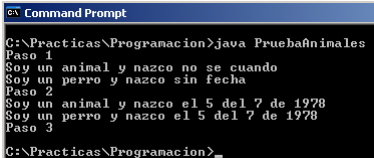
    public Animal(int diaNacimiento,
        int mesNacimiento, int anioNacimiento) {
        System.out.println("Soy un animal y nazco el " +
            diaNacimiento + " del " + mesNacimiento
            + " de " + anioNacimiento);
        this.diaNacimiento = diaNacimiento;
        this.mesNacimiento = mesNacimiento;
        this.anioNacimiento = anioNacimiento;
    }
}

public class PruebaAnimales {
    public static void main(String[] args) {
        System.out.println("Paso 1");
        Perro p = new Perro("Caniche");
        System.out.println("Paso 2");
        Perro p2 = new Perro(5, 7, 1978, "Fosterrier");
        System.out.println("Paso 3");
    }
}

public class Perro extends Animal {
    public String raza;

    public Perro(String raza) {
        System.out.println(
            "Soy un perro y nazco sin fecha");
        this.raza = raza;
    }

    public Perro(int diaNacimiento, int mesNacimiento,
        int anioNacimiento, String raza) {
        super(diaNacimiento, mesNacimiento, anioNacimiento);
        System.out.println("Soy un perro y nazco el " +
            diaNacimiento + " del " + mesNacimiento
            + " de " + anioNacimiento);
        this.raza = raza;
    }
}
```



## Agenda



- Introducción
- Clases Derivadas
  - Implementación
  - Constructores
- **Sobreescritura de Métodos**
  - Métodos Heredados vs Sobrescritos
  - Métodos static y final
  - Polimorfismo
- Métodos y clases abstractas
- Restricciones de Acceso
- La Clase Object
- Resumen



## Sobreescritura de métodos: Métodos heredados vs Sobreescritos



- Una clase hija puede sobreescribir un método de la clase padre para modificar su implementación

```
public class Animal {
    public Animal() {
    }

    public void nace() {
        System.out.println("Un animal nace");
    }

    public void crece() {
        System.out.println("Un animal crece");
    }
}

public class Perro extends Animal {
    public Perro() {
    }

    public void nace() {
        System.out.println("Un perro nace");
    }
}
```

```
public class PruebaAnimales {
    public static void main(String[] args) {
        Perro p = new Perro();
        p.nace();
        p.crece();
    }
}
```

```
Command Prompt
C:\Practicas\Programacion>java PruebaAnimales
Un perro nace
Un animal crece
C:\Practicas\Programacion>_
```

## Sobreescritura de Métodos: Métodos static y final



- No se pueden sobreescribir los métodos de clase (**static**).
- Si se declara un método de tipo **final**, no puede ser sobreescrito por clases derivadas

```
public class Animal {
    public Animal() {
    }

    public final void nace() {
        System.out.println("Un animal nace");
    }

    public void crece() {
        System.out.println("Un animal crece");
    }
}
```

```
public class Perro extends Animal {
    public Perro() {
    }

    // INCORRECTO!!!!!!
    // No se puede sobreescribir un método final
    public void nace() {
        System.out.println("Un perro nace");
    }
}
```

# Sobreescritura de Métodos: Polimorfismo



- Polimorfismo
  - Tratar objeto de una clase más general independientemente de que sea de una clase concreta
  - Ej: Si tengo un perro, un elefante y una jirafa, puedo tratarlos a todos como animales
  - Ej: Polígono -> Método para calcular el perímetro
    - Círculo:  $2 * PI * R$
    - Rectángulo:  $Base * Altura$
    - Triángulo:  $Base * Altura / 2$
  - Lista de polígonos, cada uno sabe calcular su área pero se tratan de igual manera sin saber de qué tipo de polígono concreto se trata

# Sobreescritura de Métodos: Polimorfismo



```
public class Figura {
    public float calcularArea() {
        return 0;
    }
}

public class Triangulo extends Figura {
    public float base, altura;

    public Triangulo(float base, float altura) {
        this.base = base;
        this.altura = altura;
    }

    public float calcularArea() {
        return base * altura / 2f;
    }
}

public class Circulo extends Figura {
    public float radio;
    public static final float PI = 3.14f;

    public Circulo(float radio) {
        this.radio = radio;
    }

    public float calcularArea() {
        return PI * radio * radio;
    }
}

public class PruebaFiguras {

    public static void main(String[] args) {
        Figura[] figuras = new Figura[2];
        figuras[0] = new Triangulo(3, 2);
        figuras[1] = new Circulo(2f);
        imprimirAreas(figuras);
    }

    public static void imprimirAreas(Figura[] figs) {
        for(int i=0; i<figs.length; i++) {
            System.out.println("El área es: "
                + figs[i].calcularArea());
        }
    }
}
```

Command Prompt

```
C:\Practicas\Programacion>java PruebaFiguras
El area es: 3.0
El area es: 12.56
C:\Practicas\Programacion>_
```



# Agenda



- Introducción
- Clases Derivadas
  - Implementación
  - Constructores
- Sobreescritura de Métodos
  - Métodos Heredados vs Sobreescritos
  - Métodos static y final
  - Polimorfismo
- **Métodos y clases abstractas**
- Restricciones de Acceso
- La Clase Object
- Resumen



## Métodos y Clases Abstractas



- Un método abstracto (**abstract**) de una superclase **no tiene implementación**, debe ser implementado por las clases derivadas
- Para poder definir un método abstracto, es necesario que la clase sea declarada abstracta
- Una clase abstracta puede tener métodos abstractos y métodos que no lo son
- No se puede instanciar una clase abstracta (no se pueden crear objetos de esa clase)

# Métodos y Clases Abstractas



```
public abstract class Figura {
    public String nombre;

    public Figura(String nombre) {
        this.nombre = nombre;
    }

    public void imprimirNombre() {
        System.out.println("El nombre es: " + nombre);
    }

    public abstract float calcularArea();
}

public class Triangulo extends Figura {
    public float base, altura;

    public Triangulo(String nombre, float base,
        float altura) {
        super(nombre);
        this.base = base;
        this.altura = altura;
    }

    public float calcularArea() {
        return base * altura / 2f;
    }
}

public class Circulo extends Figura {
    public float radio;
    public static final float PI = 3.14f;

    public Circulo(String nombre, float radio) {
        super(nombre);
        this.radio = radio;
    }

    public float calcularArea() {
        return PI * radio * radio;
    }
}

public class PruebaFiguras {
    public static void main(String[] args) {
        Figura[] figuras = new Figura[3];
        figuras[0] = new Triangulo("Triangulo", 3, 2);
        figuras[1] = new Circulo("Circulo", 2f);

        // NO COMPILA!!!!!!
        // No se puede instanciar una clase abstract
        figuras[2] = new Figura("Figura");
    }
}
```

## Agenda



- Introducción
- Clases Derivadas
  - Implementación
  - Constructores
- Sobreescritura de Métodos
  - Métodos Heredados vs Sobreescritos
  - Métodos static y final
  - Polimorfismo
- Métodos y clases abstractas
- **Restricciones de Acceso**
- La Clase Object
- Resumen



# Restricciones de Acceso



- Cuatro posibles visibilidades de atributos y métodos: **public**, **private**, **protected** y **package** (por defecto, no se pone **nada**).

VISIBILIDAD	public	protected	nada	private
Propia clase	SI	SI	SI	SI
Mismo paquete	SI	SI	SI	NO
Otro paquete	SI	NO	NO	NO
Subclase en paquete	SI	SI	SI	NO
Subclase en otro paquete	SI	SI	NO	NO

# Restricciones de Acceso



```
public class HijaVisibilidades extends Visibilidades {  
  
    public void imprimeAtributoPublico() {  
        System.out.println(atributoPublico);  
    }  
  
    public void imprimeAtributoPackage() {  
        System.out.println(atributoPackage);  
    }  
  
    public void intentaUsarMetodoPrivado() {  
        // NO COMPILA!!!!!!  
        // No se puede acceder a un método privado de  
        // la clase padre  
        metodoPrivado();  
    }  
  
    // Sobreescribe el método protegido de la clase padre  
    protected short metodoProtegido() {  
        return -2;  
    }  
}  
  
public class Visibilidades {  
  
    public String atributoPublico;  
    int atributoPackage;  
  
    // Un método privado no tiene sentido si  
    // no se utiliza desde otro método de la misma clase  
    private float metodoPrivado() {  
        return -1f;  
    }  
  
    protected short metodoProtegido() {  
        return 1;  
    }  
}
```

# Agenda



- Introducción
- Clases Derivadas
  - Implementación
  - Constructores
- Sobreescritura de Métodos
  - Métodos Heredados vs Sobreescritos
  - Métodos static y final
  - Polimorfismo
- Métodos y clases abstractas
- Restricciones de Acceso
- **La Clase Object**
- Resumen



## La Clase Object



- En Java, todo objeto implícitamente hereda de la clase Object
- La clase Object tiene ciertos métodos, que las subclases pueden sobreescibir
  - `public String toString()`
  - `public int hashCode()`
  - `public boolean equals(Object obj)`
  - `protected Object clone()`
- Todo objeto, por tanto, tiene una implementación por defecto de estos métodos

# Agenda



- Introducción
- Clases Derivadas
  - Implementación
  - Constructores
- Sobreescritura de Métodos
  - Métodos Heredados vs Sobreescritos
  - Métodos static y final
  - Polimorfismo
- Métodos y clases abstractas
- Restricciones de Acceso
- La Clase Object
- **Resumen**



# Resumen



- |   |  |
|---|--|
| <ul style="list-style-type: none"><li>▪ Introducción<ul style="list-style-type: none"><li>▪ Superclases y Subclases</li><li>▪ Reutilización / Centralización de código</li></ul></li><li>▪ Clases Derivadas<ul style="list-style-type: none"><li>▪ Implementación (extends)</li><li>▪ Constructores</li></ul></li><li>▪ Sobreescritura de Métodos<ul style="list-style-type: none"><li>▪ Métodos Heredados vs Sobreescritos</li><li>▪ Métodos static y final<ul style="list-style-type: none"><li>▪ No se pueden sobreescribir</li></ul></li><li>▪ Polimorfismo</li></ul></li></ul> | <ul style="list-style-type: none"><li>▪ Métodos y clases abstractas<ul style="list-style-type: none"><li>▪ Método abstract: Sin implementación por def</li><li>▪ Sólo métodos abstract en clases abstract</li><li>▪ Clases abstract no instanciables</li></ul></li><li>▪ Restricciones de Acceso<ul style="list-style-type: none"><li>▪ public, protected, nada y private</li></ul></li><li>▪ La clase Object<ul style="list-style-type: none"><li>▪ Método toString()</li></ul></li></ul> |
|---|--|

## Resumen: Para más información



- [http://pisuerga.inf.ubu.es/Isi/Invest/Java/Tuto/II\\_6.htm](http://pisuerga.inf.ubu.es/Isi/Invest/Java/Tuto/II_6.htm)
- [http://eees.ii.uam.es/alfonso/web\\_poo\\_04/teoria/material/subclases\\_e\\_interfaces.pdf](http://eees.ii.uam.es/alfonso/web_poo_04/teoria/material/subclases_e_interfaces.pdf)
- <http://www.ii.uam.es/%7Ecastells/docencia/poo/4-jerarquias.pdf>