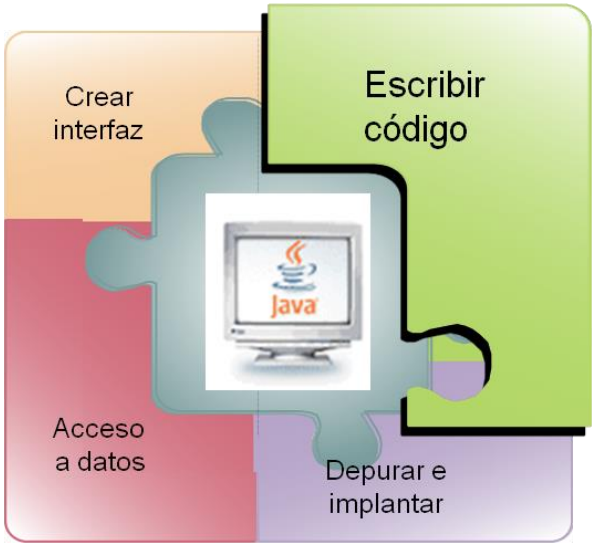
Programación orientada a objetos II

- **Unidad 8**
- **Apuntes referenciados:**
  - ✓ A5.- JorgeSanchez. Reutilización de clases. Herencia
  - ✓ A1.- Jbobi. Capítulo 5

Introducción



- Propiedades de la POO
- Concepto de Herencia
- Mecanismos de Herencia
- this y super
- Mecanismos en Constructores
- Dynamic binding
- Herencia forzada. Clases abstractas
- Revisión de los modificadores
- La clase Object

Propiedades de la POO

- **Encapsulamiento**
  - Una clase se compone tanto de variables (atributos o propiedades) como de funciones y procedimientos (métodos)
  - No se pueden definir variables (ni métodos) fuera de una clase (es decir no hay variables *globales)*
- **Ocultación**
  - Hay una zona privada al definir la clases que sólo es utilizada por esa clases y por alguna clase relacionada
  - Hay una zona pública que puede ser utilizada por cualquier parte del código
- **Polimorfismo**
  - Un método de una clase puede tener varias definiciones distintas (sobrecarga)
  - Una variable puede referirse a objetos de diferentes clases (up casting)
- **Herencia**
  - Una clase puede heredar propiedades de otra

Concepto de Herencia

- **En los lenguajes de programación orientados a objetos el mecanismo básico para la reutilización de código es la herencia**
- **Permite crear nuevas clases que heredan características presentes en clases anteriores**
  - la clase original se denomina clase padre, clase base o superclase
  - la nueva clase se denomina clase hija, derivada o subclase
- **En JAVA, sólo se puede tener herencia de una clase (herencia simple)**

Concepto de Herencia

- **En el lenguaje Java hay dos aspectos donde la herencia es particularmente relevante:**
  - La herencia se emplea exhaustivamente en el propio lenguaje a lo largo del conjunto de librerías que posee
  - El lenguaje da soporte a la definición de nuevas clases heredadas de las características ya definidas

Concepto de Herencia

- **Se emplea la palabra *extends* en la clase hija seguida del nombre de la clase padre**

```
class B extends A { ... }
```

Herencia. Ejemplo 1

```
class Persona {
        public String nombre;
        public String apellidos;
        public int añoDeNacimiento;

        public void imprime() {
                System.out.println("Datos personales: " + nombre + " " +
                apellidos + " (" + añoDeNacimiento + ")");
        }
}

class Alumno extends Persona {
        private String grupo;
        private Horario horario;

        public void ponGrupo(String grupo, Horario horario) {
                this.grupo = grupo;
                this.horario = horario;
        }
}
```

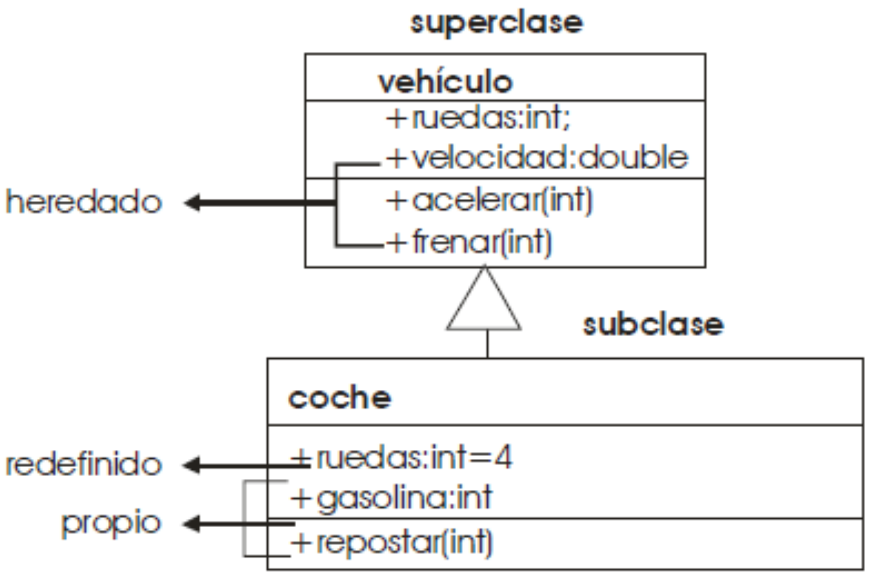
Herencia. Ejemplo 1

- **En el ejemplo, la clase *Alumno* hereda los atributos *nombre, apellidos y añoDeNacimiento* de la clase *Persona*, así como el método *imprime***
  - sobre un objeto de la clase *Alumno* es posible llamar al método *imprime*

```
//Creación de un nuevo alumno
        Alumno alumno1 = new Alumno();
//...
//Llamada al método imprime heredado de la clase Persona
        alumno1.imprime();
```

Herencia. Ejemplo 2

```
class coche extends vehiculo {
...
} //La clase coche parte de la definición de vehículo
```



Herencia. Ejemplo 2

```
class vehiculo {
   public int velocidad;
   public int ruedas;
   public void parar() {
         velocidad = 0;
   }
   public void acelerar(int kmh) {
         velocidad += kmh;
   }

class coche extends vehiculo{
   public int ruedas=4;
   public int gasolina;
   public void repostar(int litros) {
         gasolina+=litros;
   }
}
.....................................................................
public class app {
   public static void main(String[] args) {
         coche coche1=new coche();
       coche1.acelerar(80);//Método heredado
       coche1.repostar(12);
```

Mecanismos de Herencia

- **La subclase tiene (o hereda) <u>todos los atributos y métodos</u> de la superclase, aunque no todos los miembros tienen porque ser accesibles:**
  - Serán accesibles todos los métodos y propiedades *protected, public y "de paquete"*
  - *no serán accesibles* los métodos y propiedades *private*

Tipo de acceso	Palabra reservada	Ejemplo	Acceso desde una subclase del mismo paquete	Acceso desde una subclase de otro paquete
Privado	**private**	private int PPrivada;	No	No
Sin especificar		int PSinEspecificar;	Sí	No
Protegido	**protected**	protected int PProtegida;	Sí	Sí
Publico	**public**	public int PPublica;	Sí	Sí

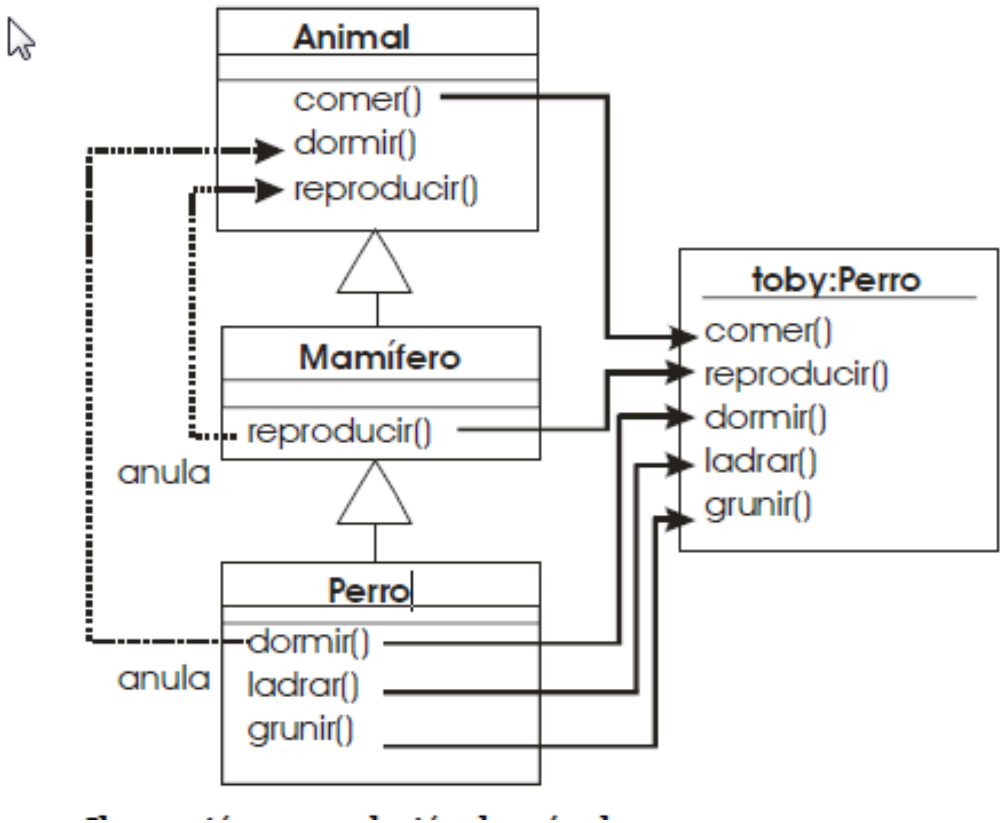
Mecanismos de Herencia

- **En la clase derivada pueden añadirse atributos (que generalmente serán privados) y métodos adicionales**
- **La subclase no hereda los constructores:**
  - cada nueva clase (incluso las derivadas) debe definir sus constructores
  - Si no se implementa ningún constructor se genera uno sin argumentos por defecto

Mecanismos de Herencia.

- **Podemos redefinir o sobrescribir (@overrite) un atributo en la subclase con el mismo nombre que en la superclase:**
  - El atributo de la subclase redefine el atributo de la superclase
  - El atributo de la superclase queda “oculto”
- **Podemos redefinir o sobrescribir(@overrite) un método en la subclase con el mismo nombre y la misma firma que en la superclase:**
  - El método de la subclase anula el método de la superclase
  - El método de la superclase queda “oculto”
  - El atributo de acceso debe ser el mismo o menos restrictivo que el de la superclase
- **Para referenciar la propiedad o método de la subclase escribimos nombre ó this.nombre**
- **Para referirnos a la superclase escribimos super.nombre**

Mecanismos de Herencia. Ejemplo



Mecanismos de Herencia

- Las subclase también pueden sobrecargar métodos para proporcionar una versión (overload)
- Para impedir que se pueda redefinir un atributo o un método se le antepone el modificador *final*
- El modificador *final* aplicado a una clase impide que se puedan definir clases derivadas

this y super

- **La palabra reservada super nos permite llamar a una propiedad o método de la superclase:**

  - this ➡ hace referencia a la clase actual
  - super ➡ hace referencia a la superclase respecto a la clase actual

Nota: Como hemos visto en la diapositiva anterior, super es imprescindible para poder acceder a atributos y métodos redefinidos o anulados por herencia

super. Ejemplo

```
public class vehiculo{
   double velocidad;
   ...
   public void acelerar(double cantidad){
         velocidad+=cantidad;
   }
}

public class coche extends vehiculo{
   double gasolina;
   public void acelerar(double cantidad){
         super.acelerar(cantidad);
         gasolina*=0.9;
   }
```

- la llamada *super.acelerar(cantidad)* llama al método *acelerar de* la clase vehículo

- Se puede llamar a un constructor de una superclase usando la sentencia super( )

```
public class vehiculo{
   double velocidad;
   public vehiculo(double v){
         velocidad=v;
   }
}

public class coche extends vehiculo{
   double gasolina;
   public coche(double v, double g){
         super(v);//Llama al constructor
         gasolina=g
   }
}
```

Mecanismos en Constructores

- **Los constructores tienen la posibilidad de invocar a otro constructor de su propia clase con la sentencia this(..)**
- **Los constructores de las subclases tienen la posibilidad de invocar a los constructores de las superclases con la sentencia super(..)**
- **La llamada super(..) o this(..) en caso de utilizarse, debe ser obligatoriamente la primera sentencia del constructor**

Por defecto Java realiza estas acciones:

- **Si la primera instrucción de un constructor de una subclase es una sentencia que no es ni super ni this**
  - añade de forma invisible e implícita una llamada super( ) al constructor por defecto de la superclase
    - Esto puede dar errores: si en la superclase hemos definido algún constructor y no hemos definido el constructor sin parámetros, el compilador no encontrará ese constructor
    - Si en la superclase no hemos definido ningún constructor, no habrá problemas
- **Si la primera instrucción es super(…)**
  - se llama al constructor seleccionado de la superclase y después continúa con las sentencias del constructor
- **Si la primera instrucción es this(…)**
  - se llama al constructor seleccionado de la clase y después continúa con las sentencias del constructor

Polimorfismo. Upcasting y Downcasting

Una variable puede referirse a objetos de diferentes clases:

- **Upcasting**
  - A una variable de un tipo A podemos no sólo asignarle objetos del tipo A, sino también cualquier objeto subclase de A
  - Ejemplo:

```
class Persona { ... }
class Alumno extends Persona { ... }

Alumno a = new Alumno( ),
Persona p =  a;      // Una variable Persona puede referenciar a un Alumno
                     // Es un Upcasting implícito
```

  - Esta operación siempre se puede hacer, sin necesidad de indicárselo explícitamente al compilador:

```
Persona p = (Persona) a;      // Upcasting explícito; no es necesario
```

Polimorfismo. Upcasting y Downcasting

- **Downcasting**
  - Se dice del caso en el que una variable de una subclase referencia a un objeto de la superclase
  - Ejemplo:

```
Persona p = new Persona( );
Alumno a = (Alumno) p;   // Una variable Alumno apunta a un objeto Persona
                         // Downcasting  explícito; es necesario
```

  - Esta operación sólo se puede hacer, si el objeto referenciado por "p" es realmente de tipo Alumno

```
Persona p = new Alumno( );
```

    Es decir el Downcasting deshace un Upcasting previo:
  - *De lo* contrario se provoca un error lanzándose una excepción de tipo ClassCastException

Dynamic binding (elección dinámica de método)

- **Ejemplo:**

```
Persona p = new Alumno();
Alumno a = new Alumno();
```

  - Supongamos que la clase Persona dispone de un método imprime( ) y que este método se ha sobreescrito en la clase Alumno
  - Podemos escribir:

```
p.imprime()
a.imprime()
```

  - Ambas invocaciones ejecutarán el código de la clase Alumno

instanceof

- **Permite comprobar si un determinado objeto pertenece a una clase concreta:**

```
objeto instanceof clase
```

  - Comprueba si el objeto pertenece a una determinada clase y devuelve un valor true si es así

- **Ejemplo:**

```
Coche miMercedes=new Coche();
if (miMercedes instanceof Coche)
  System.out.println("ES un coche");
if (miMercedes instanceof Vehículo)
  System.out.println("ES un coche");
if (miMercedes instanceof Camión)
  System.out.println("ES un camión");
```

Herencia forzada. Clases abstractas

Se puede obligar que para usar una clase haya que hacerlo escribiendo una nueva clase que herede de ella

- **Para ello se utiliza el modificador *abstract* delante de la definición de la clase**
- **Ejemplo:**

```
abstract class NombreDeClase { ... }
```

- **Una clase abstracta no se puede instanciar (no se puede crear objetos)**
  - Se debe instanciar la clase derivada

La Clase Object

- **Es una clase de java con la particularidad de ser la "madre de todas las clases"**
  - Todas las clases que escriban los programadores heredan de Object
    - directamente (si no se dice nada)
    - indirectamente (si la clase extiende a otra)
- **Es equivalente:**

```
class Ejemplo { ... }

class Ejemplo extends Object { ... }
```

- **La clase Object define métodos que son compartidos por absolutamente todos los objetos que se creen**

La Clase Object

Algunos métodos de Object son:

- **getClass( )**
  - retorna la clase del objeto en ejecución
- **equals(Object obj)**
  - retorna true si este objeto es igual al pasado como argumento
- **toString( )**
  - retorna una representación textual del objeto
  - Una buena estrategia es sobrecargar toString() para modificar la representación de un objeto

La Clase Object. Ejemplo de toString( )

```
public class Ficha {

    // Atributos
    private String nombre = null;
    private String apellidos = null;
    private int edad = 0;
// Contructores
    public Ficha(String param1, String param2, int param3) {...}
// Métodos
    public String toString() //alternativa 1
    {
        return "Nombre: " + nombre + " Apellidos: " + apellidos + " Edad: " + edad;
    }
    public String mostrar() //alternativa 2
    {
        return "Nombre: " + nombre + " Apellidos: " + apellidos + " Edad: " + edad;
    }

}
```

Revisión de los modificadores de clases, atributos y métodos

RECUERDA….

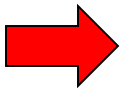
Modificadores de acceso de clases (Unidad 05)

- Java define 4 modificadores fundamentales que califican a clases:

Palabra clave	Definición
public	La clase es accesible desde otros paquetes
(por defecto) de paquete	La clase será visible en todas las clases declaradas en el mismo paquete
abstract	La clase no pueden ser instanciadas. Sirve únicamente para declarar subclases
final	ninguna clase puede heredar de una clase final

Modificadores de atributos y métodos (Unidad 05)

- Java define 4 modificadores fundamentales que califican a métodos y atributos:



Palabra clave	Definición
public	el elemento es accesible desde cualquier sitio
protected	el elemento es accesible dentro del paquete en el que se define y, además, en las subclases
(por defecto) de paquete	el elemento sólo es accesible dentro del paquete en el que se define (clases en el mismo directorio)
private	el elemento sólo es accesible dentro del fichero en el que se define