

Intro POO (VI)

- El método **toString()**
- Métodos **get...()** y **set...()**
- Operadores ***abstract*** y ***final*** para clases

Recordando

- Herencia
 - Concepto
 - Características generales
 - Relación con el encapsulamiento
 - Operador super
 - Uso en Java
- Package
 - Concepto
 - Características
 - Uso en Java

Temas a revisar

- Recordando clase anterior...
- El método **toString()**
- Métodos **get...()** y **set...()**
- Operadores ***abstract*** y ***final*** para clases

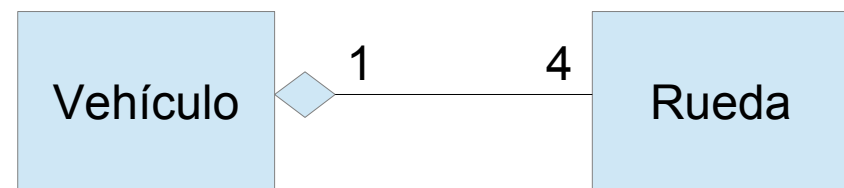
Ocultamiento de información

- **Public:** Cualquier clase puede acceder a las propiedades y métodos públicos.
- **Protected:** Sólo las clases heredadas y aquellas situadas en el mismo paquete pueden acceder a las propiedades y métodos protegidos.
- **Private:** Las variables y métodos privados sólo pueden ser accedidos desde dentro de la clase.

Herencia - composición

- ☠ Para definir una nueva clase
 - Sea una clase Rueda y una clase Vehículo
 - * Se sabe que un vehículo posee 4 instancias de tipo Rueda.
 - **class Vehículo {
 Rueda r1,r2, r3, r4; ... }
}**
- ☠ A esto se le llama una **relación de composición**, un Vehículo está compuesto por 4 ruedas.

**Representación de la Composición
en UML**



Herencia - especialización

☠ Se tiene una clase y se desea crear algo de características similares pero con algunas extensiones o modificaciones que la hagan más particular.

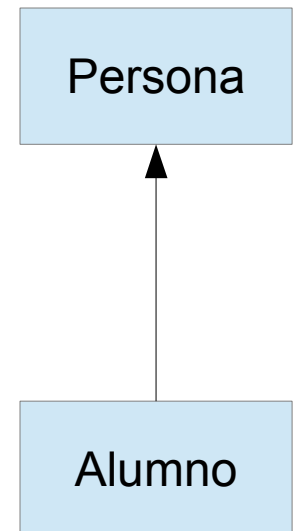
→ Se tiene la clase Persona

- Se sabe que todo Alumno es una Persona

→ **class Alumno *extends* Persona { ...}**

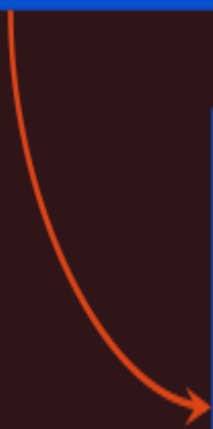
☠ A esto se le llama una **relación de especialización**

Representación de la Especialización en UML



Herencia – operador super

```
class cPadre {  
    boolean unaVariable;  
    void unMetodo()  
    { unaVariable = true; } }
```



```
class cHija extends cPadre {  
    boolean unaVariable;  
    void unMetodo() {  
        unaVariable = false;  
        super.unMetodo( );  
        System.out.println(unaVariable);  
        System.out.println(super.unaVariable);  
    } }
```

¿Qué sucede en este código?
¿Qué se podría observar por pantalla?

Los Package - implementación

The image shows a Java IDE with two code editors. The left editor shows the file structure of a package named 'mios' containing a class 'utiles.class' and a file 'utiles.java'. The right editor shows the code for 'pruebaPaquete.java' which imports the 'mios' package and defines a 'pruebaPaquete' class with a 'main' method. The bottom editor shows the code for 'utiles.java' which defines the 'utiles' class with methods 'largo', 'vacia', and 'esPar'.

File Path: ~/Desktop/ejPackage/usoPaquetes/mios/utiles.java

```
1 // *****
2 * Ejemplo de la creación de un paquete personalizado
3 * clase con utilitarios que serán llamados después desde
4 * otros programas mediante un import.
5 * Este package debe ser guardado en un directorio con el
6 * mismo nombre del package.
7 */
8
9 // se define el nombre del paquete
10 package mios;
11
12 public class utiles {
13
14     public int largo(String s) {
15         return s.length();
16     }
17
18     public boolean vacia(String s) {
19         return (s.length() == 0);
20     }
21
22     public boolean esPar(int n) {
23         return (n%2 == 0);
24     }
25 }
26
27
```

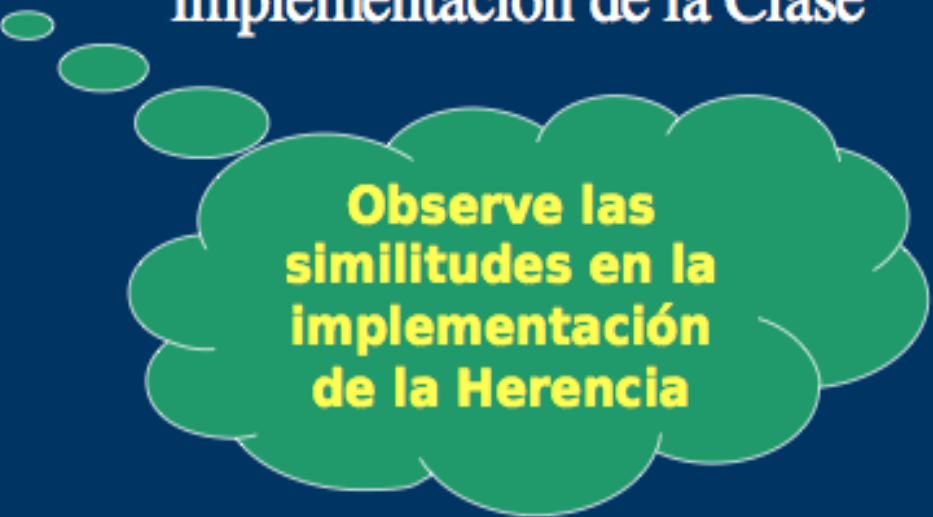
File Path: ~/Desktop/ejPackage/usoPaquetes/pruebaPaquete.java

```
1 // hacer un import de nuestro propio paquete
2 import mios.*;
3
4 public class pruebaPaquete {
5
6     /**
7      * @param args
8      */
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         String st = new String();
12         // se crea un objeto de la clase utiles
13         // de nuestro paquete mios
14         utiles u = new utiles();
15         int x = 3;
16         st = "hola";
17         // se invocan a operaciones de la clase
18         // utiles del paquete creado.
19         System.out.println(u.largo(st));
20         System.out.println(u.vacia(st));
21         System.out.println(u.esPar(x));
22     }
23
24 }
```


Ejs. Herencia en LDP - POO

- C++:
class C_Hija : public C_Padre
{ Implementación de la Clase }
- JAVA:
class C_Hija extends C_Padre
{ Implementación de la Clase }
- Delphi:
type
C_Hija = class(C_Padre)
 implementación de la Clase
end;

- ADA:
type C_Hija is new C_Padre
 with
 implementación de la Clase
- SmallTalk:
class C_Hija superclass
C_Padre
implementación de la Clase



**Observe las
similitudes en la
implementación
de la Herencia**

Herencia – Clases, sub clases, paquetes, encapsulamiento

private: es el nivel más restrictivo, donde una subclase no puede acceder a los elementos (variables y métodos) privados de la superclase.

protected: variables y métodos de una clase son visibles en las subclases.

public: variables y métodos declarados de esta forma pueden ser usados en cualquier clase.

package: variables y métodos pueden ser usados en la clase que fueron declaradas y en las clases que pertenecen al mismo paquete.

	Clase	Subclases	Paquete	Todas
private	X			
protected	X	X	X	
public	X	X	X	X
package	X		X	

Método toString

- Está disponible para todos los objetos en Java, pues se hereda de la clase Object.
- Se puede optar por usar el método ***toString***, predeterminado o pueden definirlo según sus necesidades.
 - La implementación predeterminada devuelve una cadena de texto, que incluye: nombre de la clase y un número hexadecimal que representa el ***código hash*** del objeto.

¿Qué es un código hash?

Método toString

- Se debe definir un método público
- Devuelve un tipo String
- Se agrega un mensaje y los atributos que se requieran

```
public String toString( ) {  
    return "Objeto de la clase con atributo: " + unAtributo;  
}
```

Ver ejemplo en Campus Virtual TestArreglo.java

Método toString

```
7 public class TestArreglo {
8     public static void main(String [] args) throws Exception {
9         InputStreamReader isr = new InputStreamReader(System.in);
10        BufferedReader br = new BufferedReader(isr);
11
12        System.out.println("\nTrabajo basico con Arreglos!!!\n");
13        System.out.println("Cuantos datos quieres mostrar? : ");
14
15        int largo = Integer.parseInt(br.readLine());
16
17        arreglo A = new arreglo(largo);
18        System.out.println("Largo total arreglo: "+A.vector.length);
19        A.mostrar(largo);
20
21        System.out.println(A);
22
23        // destruccion explicita del objeto
24        A = null;
25        if (A==null) System.out.println("Objeto destruido\n");
26    }
27 }
28
29 class arreglo {
30     int [] vector = new int[50];
31
32     public arreglo(int n) {
33         for (int i=0;i<n;i++)
34             this.vector[i] = i;
35     }
36
37     public String toString() {
38         String msj = "";
39         for (int i=0; i < vector.length ; i++ ) msj+= String.valueOf(vector[i]) + "-";
40         return msj;
41     }
```

Métodos setter y getter

- Métodos setter

- Permite asignar un valor un atributo, pero de forma explícita
- No retorna ningún valor (siempre es void)
- Nos permite dar acceso público a ciertos atributos que permitimos se puedan modificar desde fuera de la clase.

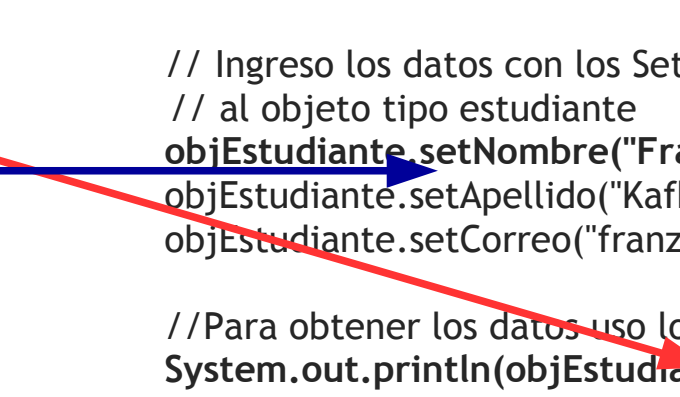
- Métodos getter

- Permite recuperar o acceder el valor asignado a un atributo.
- Retorna un valor del tipo asociado a lo que queremos retornar.

Métodos setter y getter

```
class Estudiante {  
    private String nombre;  
    private String apellido;  
    private String correo;  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getApellido() {  
        return apellido;  
    }  
  
    public void setApellido(String apellido) {  
        this.apellido = apellido;  
    }  
  
    public String getCorreo() {  
        return correo;  
    }  
  
    public void setCorreo(String correo) {  
        this.correo = correo;  
    }  
}
```

```
public class TestEstudiante {  
  
    public static void main(String args[]) {  
  
        Estudiante objEstudiante = new Estudiante();  
  
        // Ingreso los datos con los Setter  
        // al objeto tipo estudiante  
        objEstudiante.setNombre("Franz");  
        objEstudiante.setApellido("Kafka");  
        objEstudiante.setCorreo("franz.kafka@ufro.cl");  
  
        // Para obtener los datos uso los Getter  
        System.out.println(objEstudiante.getNombre());  
        System.out.println(objEstudiante.getApellido());  
        System.out.println(objEstudiante.getCorreo());  
    }  
}
```



Elementos Abstract y Final

- El modificador **abstract** declara que la clase es una clase abstracta.
- Una clase abstracta puede contener métodos abstractos (métodos sin implementación).
- Una clase abstracta está diseñada para ser una superclase y no puede instanciarse.
- *¿Para qué sirve sino puedo crear objetos?*

```
abstract class claseAbstracta {  
    // atributos  
    // métodos . . . }
```


Elementos Abstract y Final

- Al utilizar el modificador **final** se declara que una clase es final y esto implica que no puede tener subclases.
- Existen al menos dos razones para hacer esto:
 - **Seguridad**: evitar que la clase sea reemplazada y permita la ejecución de código malicioso.
 - **Diseño**: por motivos de DOO, si la clase no dará lugar a nuevas clases.
- En algunos casos puede resultar muy drástico el tener una clase como Final
 - Es posible tener sólo algunos métodos de tipo Final.

```
final class claseFinal { ... }
```

```
class claseFinal {  
    final void metodoFinal( ) { ... }  
}
```

Caso personasUFRO

- A partir del caso iniciado la clase anterior, se le pide:
 - Agregar el concepto de clase abstracta a la clase Persona
 - Agregar el concepto de método final en al menos un método de la clase Docente y Alumno
 - Incorpore en la clase Docente y Alumno, el concepto de los métodos setter y getter
 - Además incorpore el concepto del método toString en la clase Docente y Alumno

Resumiendo

- Herencia y otros LDP
- Uso del método toString
- Métodos setter y getter
- Clases abstractas y finales