Enterprise Java Developer

Programación Orientada a Objetos

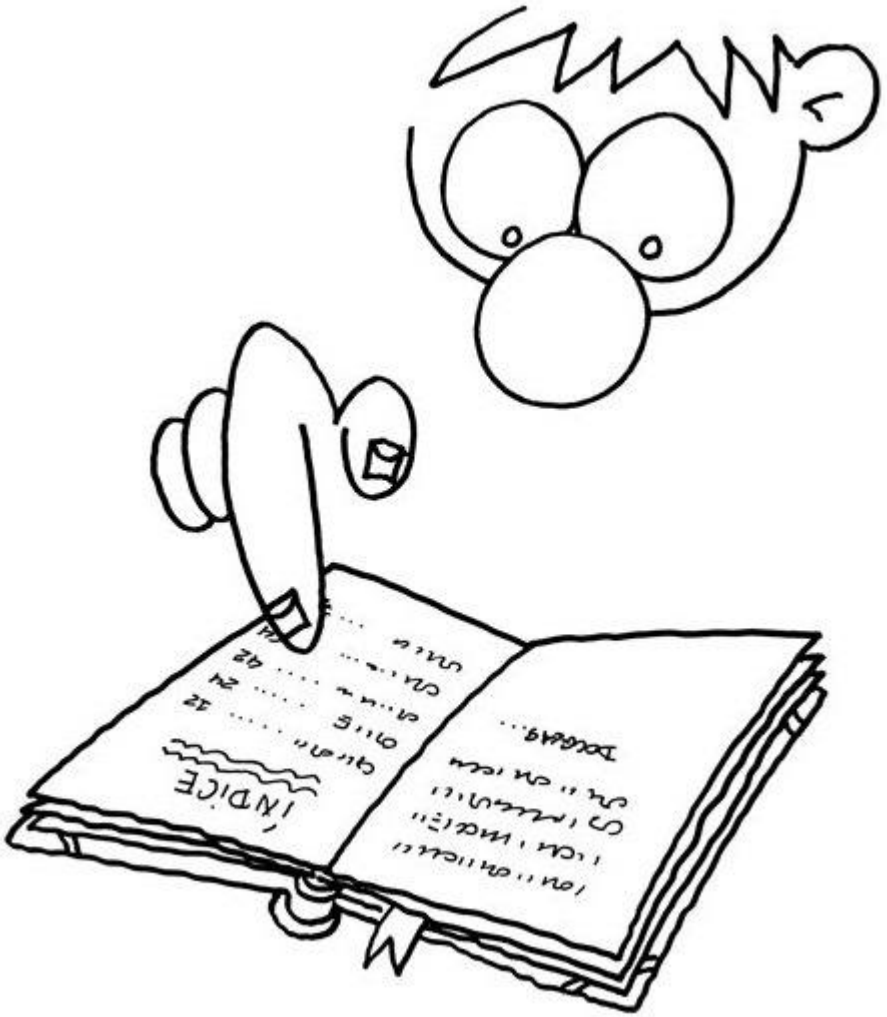
Interfaces y Polimorfismo

Eric Gustavo Coronel Castillo
gcoronelc@gmail.com
gcoronelc.blogspot.com

Ricardo Walter Marcelo Villalobos
ricardomarcelo@hotmail.com

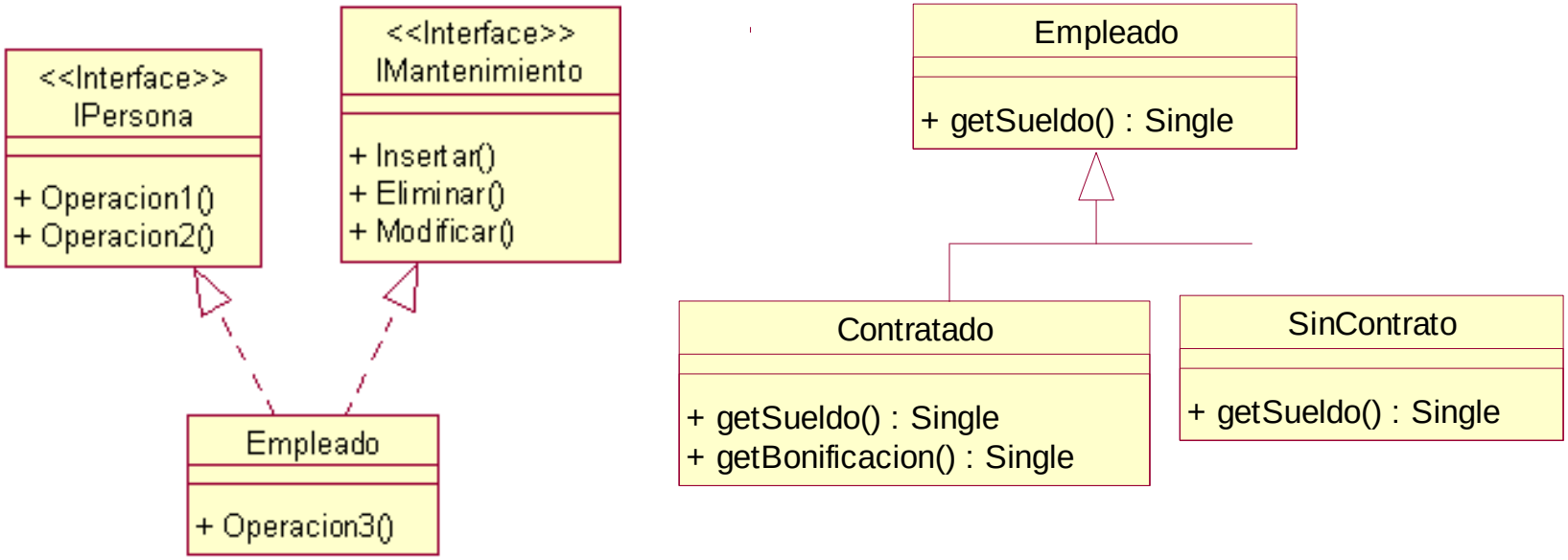
ÍNDICE

- Objetivo
- Interface
- Diferencia entre Clase Concreta, Abstracta e Interface
- Polimorfismo
- Operador instanceof
- Casting
- Ligadura Estática y Dinámica
- Paquetes (Packages)
- Control de Acceso a los Miembros de una Clase



OBJETIVOS

- Aplicar interfaces en el diseño de componentes software.
- Aplicar el polimorfismo en el diseño de componentes software

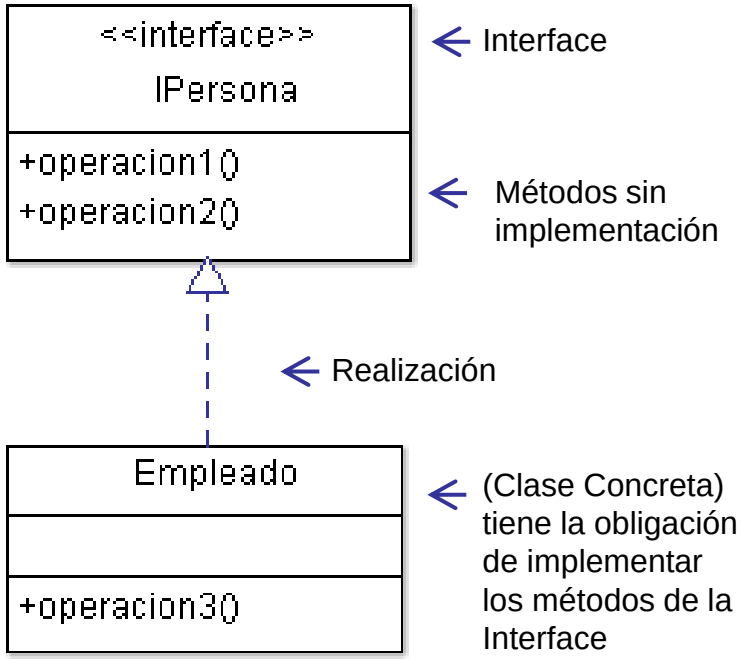


INTERFACE

- Solo contienen operaciones (métodos) sin implementación, es decir solo la firma (signature).
- Las clases son las encargadas de implementar las operaciones (métodos) de una o varias Interfaces (*Herencia múltiple*).
- Se dice que se crean Interface cuando sabemos que queremos y no sabemos como hacerlo y lo hará otro o lo harán de varias formas (*polimorfismo*).

```
public interface IPersona {
   public void operacion1();
   public void operacion2();
}
```

```
public class Empleado implements IPersona {
   public void operacion1() {
      //implementar el método de la interface
   }
   public void operacion2() {
      //implementar el método de la interface
   }
   public void operacion3() {
      //implementación
   }
}
```



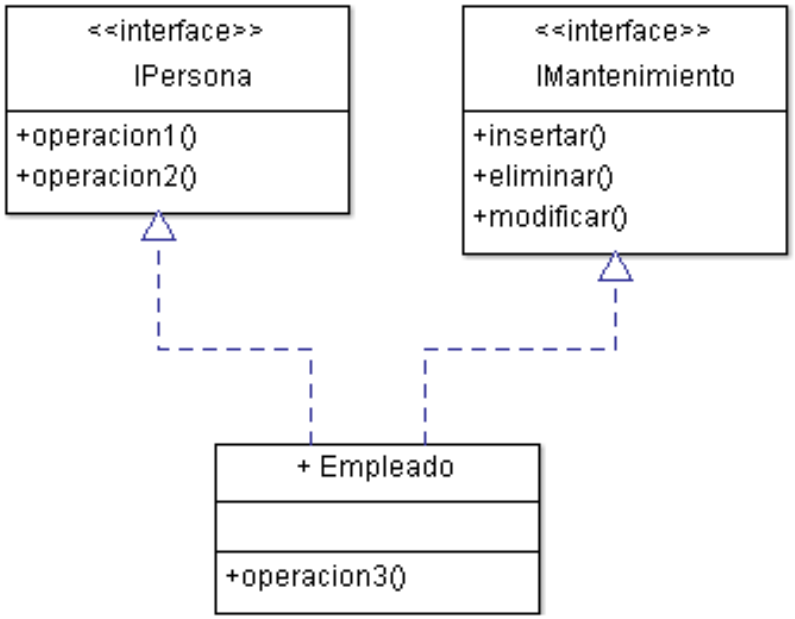
INTERFACE

- Ejemplo de Herencia múltiple de Interface.

```
public interface IPersona {
   public void operacion1();
   public void operacion2();
}
```

```
public interface IMantenimiento {
   public void insertar();
   public void eliminar();
   public void modificar();
}
```

```
public class Empleado
implements Ipersona, IMantenimiento {

    //implementar los métodos de la interface
    // . . .
    // . . .
    // . . .

}
```



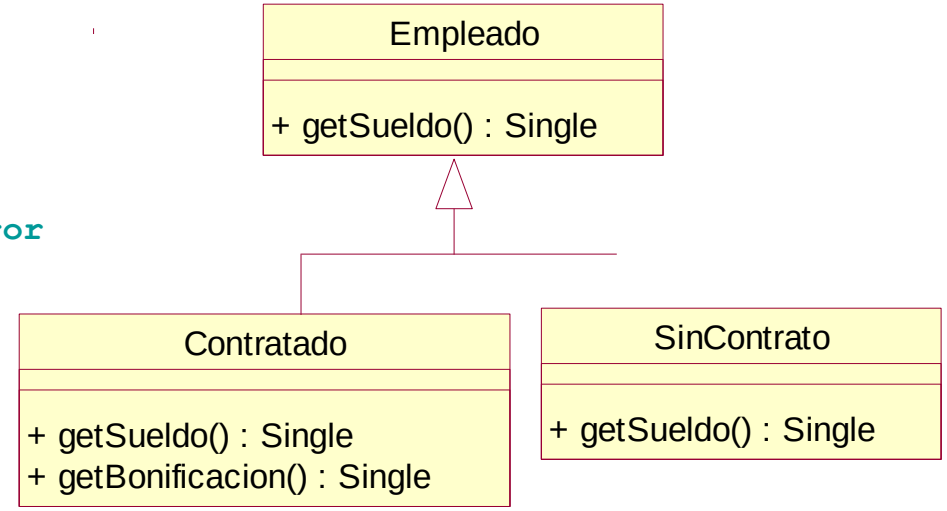
CLASE CONCRETA, ABSTRACTA E INTERFACE

Tipo	Clase Concreta	Clase Abstracta	Interface
Herencia	extends (simple)	extends (simple)	implements (múltiple)
Instanciable	Si	No	No
Implementa	Métodos	Algunos métodos	Nada
Datos	Se permite	Se permite	No se permite

POLIMORFISMO

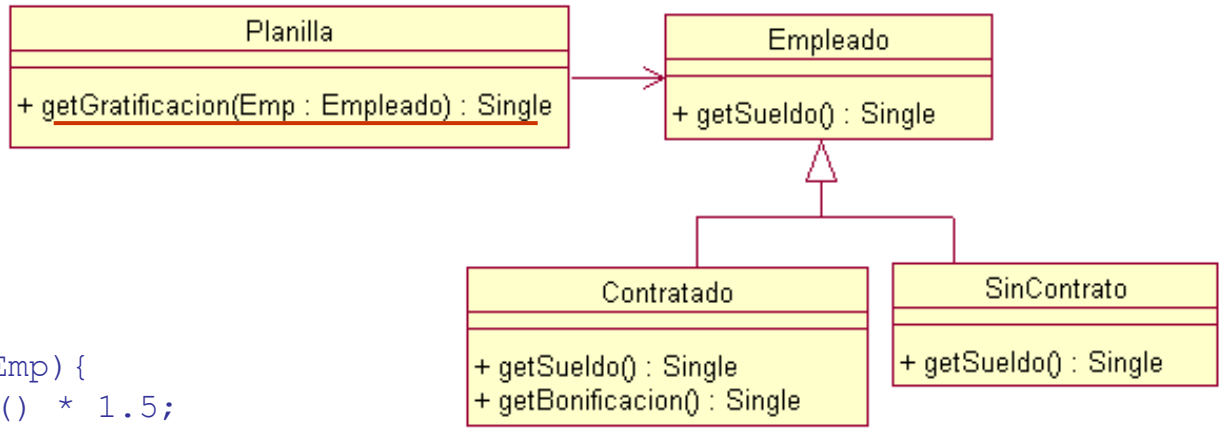
- Se dice que existe polimorfismo cuando un método de una clase es implementado de varias formas en otras clases.
- Algunos ejemplos de Polimorfismos de herencia son: *sobre-escritura*, *implementación* de métodos abstractos (clase abstracta e interface).
- Es posible apuntar a un objeto con una variable de tipo de *clase padre* (supercalse), esta sólo podrá acceder a los miembros (campos y métodos) que le pertenece.

```
// Variable de tipo Empleado y apunta a un
// objeto de tipo Contratado.
Empleado objEmp = new Contratado();

// Invocando sus métodos
double S = objEmp.getSueldo();         //OK
double B = objEmp.getBonificacion(); //Error
```



POLIMORFISMO

- El método getGratificacion puede recibir objetos de Empleado o subtipos a este.
- Cuando invoque el método getSueldo se ejecutará la versión correspondiente al objeto referenciado.

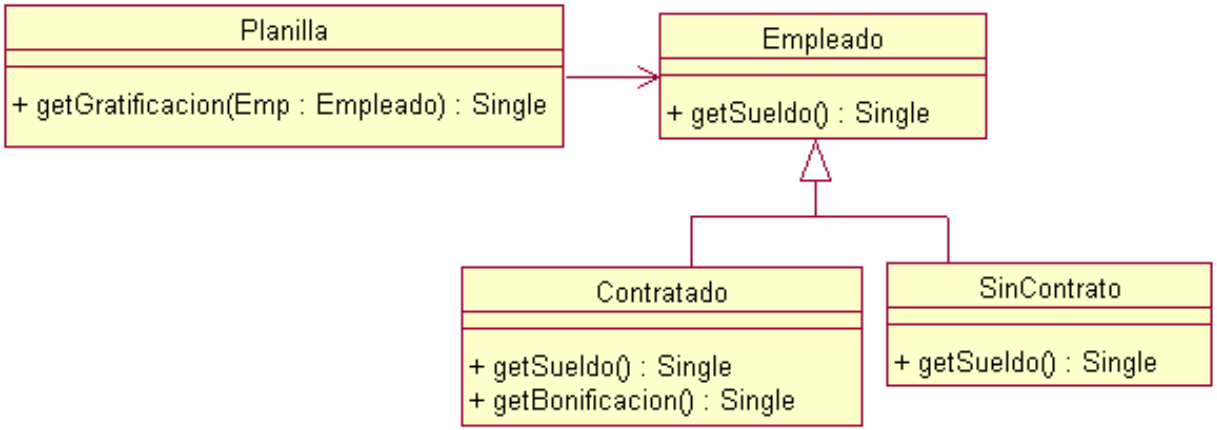



```
public class Planilla {
   public static double
   getGratificacion(Empleado Emp){
         return Emp.getSueldo() * 1.5;
   }
}
```

```
//Usando la clase Planilla
double G1 = Planilla.getGratificacion(new Contratado());
double G2 = Planilla.getGratificacion(new SinContratado());
```

OPERADOR instanceof

- Este operador permite verificar si el objeto es instancia de un tipo especifico.




```
public class Planilla {
   public static double getGratificacion(Empleado Emp){
      if (Emp instanceof Contratado)
            return Emp.getSueldo() * 1.5;
      if (Emp instanceof SinContratado)
            return Emp.getSueldo() * 1.2;
   }
}
```

```
//Usando la clase Planilla
double G1 = Planilla.getGratificacion(new Contratado());
double G2 = Planilla.getGratificacion(new SinContratado());
```

CASTING

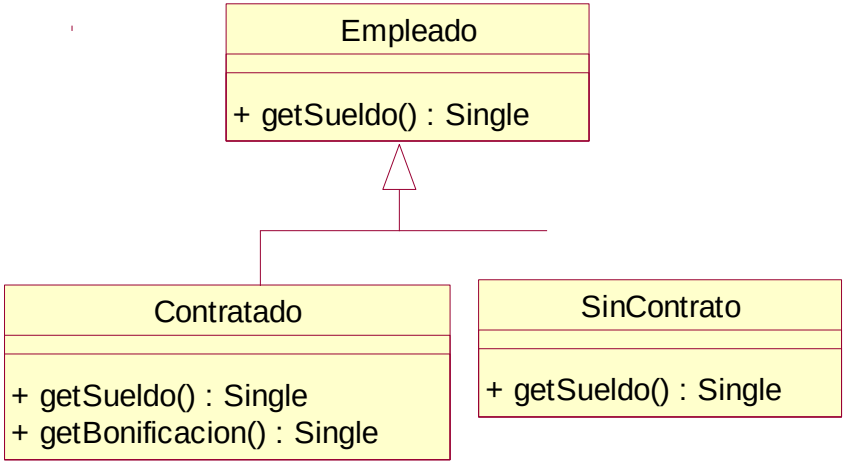
- Para restablecer la funcionalidad completa de un objeto, que es de un tipo y hace referencia a otro tipo, debe realizar una conversión (Cast).
- **UpCasting**: Conversión a clases superiores de la jerarquía de clases (Herencia), es automático (conversión implícita), basta realizar la asignación.
- **DownCasting**: Conversión hacia abajo, es decir hacia las subclases de la jerarquía (Herencia), es recomendable realizar Cast (conversión explicita), si no es compatible genera un error (Excepción).

```
//UpCasting (Conversión implícita)
Contratado a = new Contratado();
Empleado b = a;

//DownCasting (Conversión explicita)
Empleado a = new Contrtado();
Contratado b = (Contratado)a;

//Error de compilación
SinContrato a = new SinContrato();
Contratado b = (Contratado)a;
```

LIGADURA ESTÁTICA Y DINÁMICA

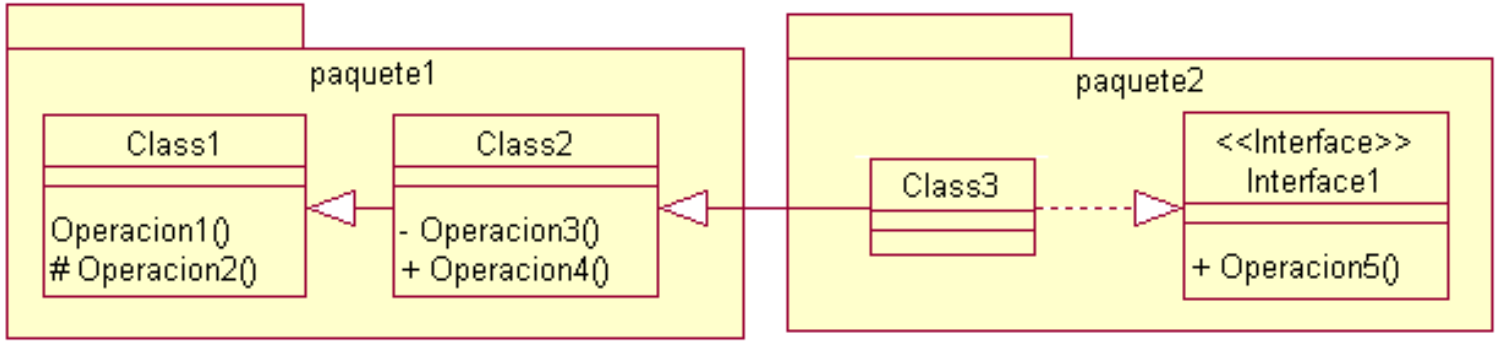
- Esta relacionado en el momento que los nombres de variables se ligan (enlazan) con sus tipos de datos.
- **Ligadura (enlace) estática o temprana**: Consiste en fijar el tipo de dato de las variables en tiempo de compilación.
- **Ligadura (enlace) dinámica o tardía:** Consiste en fijar el tipo de dato de las variables en tiempo de ejecución, es decir una variable de un tipo puede almacenar objetos de otros tipos de la jerarquía de clases.

```
//LIGADURA ESTATUCA
Empleado a = new Empleado();

//LIGADURA DINAMICA
Empleado a;
a = new Empleado();
//Obtener el sueldo del Empleado
double s = a.getSueldo();
a = new Contratado();
//Obtener el sueldo del emp. Contratado
double s = a.getSueldo();
a = new SinContrato();
//Obtener el sueldo del emp. SinContrato
double s = a.getSueldo();
```



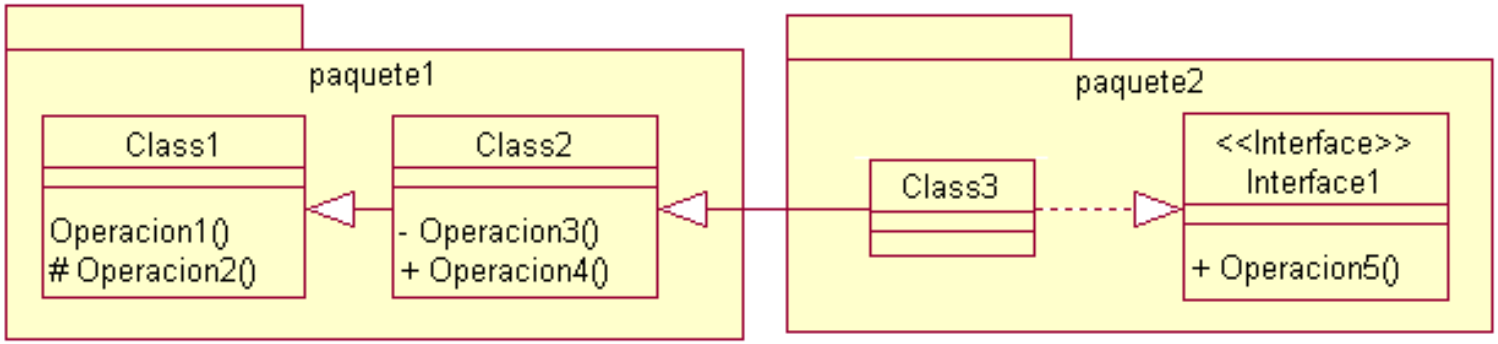
PAQUETES (PACKAGES)

- Organiza y agrupa un conjunto de clases, interfaces, subpaquetes y otros.
- La creación de paquetes evita el conflicto de nombre de clases, además un paquete puede contener clases, campos y métodos que están disponible sólo dentro del paquete.
- Observe la siguiente figura usando notación UML, y responda **¿Qué operaciones (métodos) tendrá la clase Class3?**



PAQUETES (PACKAGES)

Para definir un paquete use *package* y para utilizar clases de otro paquete, indique la ruta del paquete antes del nombre de la clase o use *import*




```
//Definiendo un paquete
package paquete1;

//Clase asociada al paquete
public class Class1() { . . . };
```

```
//Definiendo un paquete
package paquete2;
//Importando todas las clases del paquete
import paquete1.*;
//Clase asociada al paquete
public class Class3()
          extends Class2
           implements Interface1 { . . . };
```

```
//Definiendo un paquete
package paquete1;

//Clase asociada al paquete
public class Class2()
          extends Class1 { . . . };
```

```
//Definiendo un paquete
package paquete2;

//Interface asociada al paquete
public interface Interface1() { . . . };
```

CONTROL DE ACCESO A LOS MIEMBROS DE UNA CLASE

- Se conoce 4 formas de controlar el acceso a los campos (atributos) y métodos (operaciones) de las clases.
- **private** (**-**): Acceso sólo dentro de la clase.
- : Acceso sólo dentro del paquete.
- **protected** (**#**): Acceso en la clase y en subclases (herencia dentro o fuera del paquete).
- **public** (**+**): Acceso desde cualquier parte.

Acceso / Visibilidad	Misma Clase	Mismo paquete	SubClase	Universal
public (+)	Sí	Sí	Sí	Sí
protected (#)	Sí	Sí	Sí	
	Sí	Sí		
private (-)	Sí			

BIBLIOGRAFÍA