

SCJP – Capítulo 3

Declaración y Control de Acceso

Agenda

- Modificadores de acceso
- Reglas de declaración
- Implementación de interfaces

Modificadores de Acceso

- Sólo puede haber una clase pública por archivo de código fuente
- El nombre del archivo debe ser igual al nombre de la clase pública
- Si la clase es parte de un paquete, la sentencia `package` debe ser la primera en el archivo de código fuente

Modificadores de Acceso

- La sentencia de `import` debe estar entre la sentencia `package` y la definición de la primera clase del archivo fuente
 - Si no hay sentencia `package`, la sentencia `import` debe ser la primera
 - Si no hay sentencia `package` o `import`, la definición de la clase debe ser la primera (sin considerar los comentarios, estos pueden estar en cualquier parte)

Modificadores de Acceso

- Las sentencias `import` y `package` se aplican a todas las clases contenidas en el código fuente.
- Los modificadores de accesos son cuatro: `public`, `protected`, `private` y el cuarto modificador de acceso es llamado “visibilidad de paquete” y **es cuando se omite alguno de los tres anteriores.**
- **No** modificadores de accesos: `abstract`, `final` y `strictfp`.

Acceso por Omisión (default)

```
package cl.uv.inf;  
class Profesor { }
```

```
package cl.uv.inf.administrativo;  
import cl.uv.inf.Profesor;  
class ProfesorHora extends Profesor { }
```

Uso de final

```
package cl.uv.inf;  
public final class Profesor { }
```

```
package cl.uv.inf.administrativo;  
import cl.uv.inf.Profesor;  
class ProfesorPlanta extends Profesor { }
```

Uso de objetos final

```
class Padre {  
    String nombre = "mi nombre";  
}
```

```
class Informe {  
    void cambiarNombre(final Padre padre){  
        padre.nombre = "Otro Nombre"; //OK  
    }  
}
```


Clases Abstractas

```
abstract class Auto {  
    private String patente;  
    private double velocidad;  
    public abstract void manejar();  
    public final void detenerse() {  
        velocidad = 0.0;  
    }  
}
```

```
class RentaCar {  
    private Auto auto = new Auto();  
}
```

Combinaciones Equibocadas

- Clases

abstract **final** class Clase { }

- Metodos

abstract **synchronized** void metodo { }

abstract **strictfp** void metodo { }

abstract **native** void metodo { }

abstract **static** void metodo { }

abstract **final** void metodo { }

Accediendo a Métodos Públicos

```
class Persona {  
    public String unMetodo(){  
        return "contenido de unMeotodo()";  
    }  
}
```

```
class Profesor extends Persona{  
    public void usandoUnMetodo(){  
        System.out.println("Profesor : " + this.unMetodo());  
    }  
}
```

```
Persona p = new Persona();  
System.out.println("Persona : " + p.unMetodo());  
}
```

Accediendo a Métodos Privados

```
class Persona {  
    private String unMetodo(){  
        return "contenido de unMeotodo()";  
    }  
}
```

```
class Profesor extends Persona{  
    public void usandoUnMetodo(){  
        System.out.println("Profesor : " + this.unMetodo());  
    }  
}
```

```
Persona p = new Persona();  
System.out.println("Persona : " + p.unMetodo());  
}
```

Variables Locales

- Las variables locales (o de método) **no tienen modificador de acceso**

```
class Algo {  
    void hacerCosas(){  
        private int x = 7;  
        this.hacerMasCosas(x)  
    }  
}
```





















- El único modificador que se puede usar es **final**

Variables Locales

- Ocultación o ensombreamiento (shadowing)

```
class TestServer {  
    int contador = 0;  
    void contar(){  
        int contador = 1; // oculta el atributo  
        contador = 5; // acceso a la variable local  
        this.contador = 7 ; // acceso al atributo  
    }  
}
```

Tabla de Accesos

Visibilidad	public	protected	<i>default</i>	private
De la misma clase				
De cualquier clase en el mismo paquete				
Desde cualquier clase (no subclass) fuera del paquete				
Desde una subclass en el mismo paquete				
Desde una subclass fuera del paquete				

Variables y Métodos de Clase

```
class Perro {  
    static int cuentaPerros = 0;  
    Perro(){  
        cuentaPerros++;  
    }  
    public static void main(String[] args){  
        new Perro();  
        new Perro();  
        new Perro();  
        System.out.println("nro de Perros :" + cuentaPerros);  
    }  
}
```


Implementación de Interface

- Todos los métodos son implícitamente **public** y **abstract** (no es necesario escribirlo).
- Las interfaces **no** pueden tener métodos estáticos (**static**).
- Todas las variables definidas son implícitamente **public**, **static** y **final** (en otras palabras son constantes y **no** variables de clase).

Implementación de Interface

- Como los métodos son abstractos, estos no pueden ser marcados como: **final**, **native**, **strictfp** o **synchronized**.
- Un interface puede extender (**extends**) una o más interfaces.
- Una interface **no** puede extender nada que no sea otra interface.

Implementación de Interface

- Una interface no puede implementar (**implements**) otra interface o clase.
- Las interfaces deben ser declaradas con la palabra clave interface.
- La siguiente declaración de interface es correcta:

```
public abstract interface Manejar { }
```

La palabra clave abstract es opcional

Declarando una interfaz

```
public interface Manejar {  
    public static final int MAX = 120;  
    abstract public int getVelocidad();  
}
```

Implementando Multiples interfaces

```
public interface Gatear { }  
public interface Caminar { }  
public interface Correr  
    extends Gatear, Caminar { }  
public interface Nadar { }  
  
class Persona implements Correr, Nadar { }
```

Interfaces y Clases Abstractas

```
interface Correr {  
    void desplazarse(int mts);  
}  
interface Nadar {  
    void desplazarse(int millas);  
    void sumergirse();  
}  
abstract class AbsPersona implements Nadar, Correr{  
    public void desplazarse(int x) { ... }  
}  
class Persona extends AbsPersona {  
    public void sumergirse(){ ...}  
}
```

Problemas con los Contratos

```
interface SuperHeroe {  
    void volar();  
}  
interface Pajaro{  
    String volar();  
}  
class Condorito implements SuperHeroe, Pajaro {  
    public void volar();  
    public String volar(); // Error de Compilacion  
}
```

Problemas: extends e implements

- *class Foo {} // OK*
- *class Bar implements Foo {} // No Puedes implementar una clase*
- *interface Baz {} // OK*
- *interface Fi {} // OK*
- *interface Fee implements Baz {} // Una Interface no puede implementar otra interface*
- *interface Zee implements Foo {} // Una Interface no puede implemetar una clase*
- *interface Zoo extends Foo {} // Una interface no puede extender una clase*
- *interface Boo extends Fi {} // OK. Una interface puede extender a otra*
- *class Toon extends Foo, Button {} // Una clase no puede extender multiples clases*
- *class Zoom Implements Fi, Fee {} // OK. Una clase puede implementar multiples interfaces*
- *interface Vroom extends Fi, Fee {} // OK. Una interface puede extender multiples interfaces*