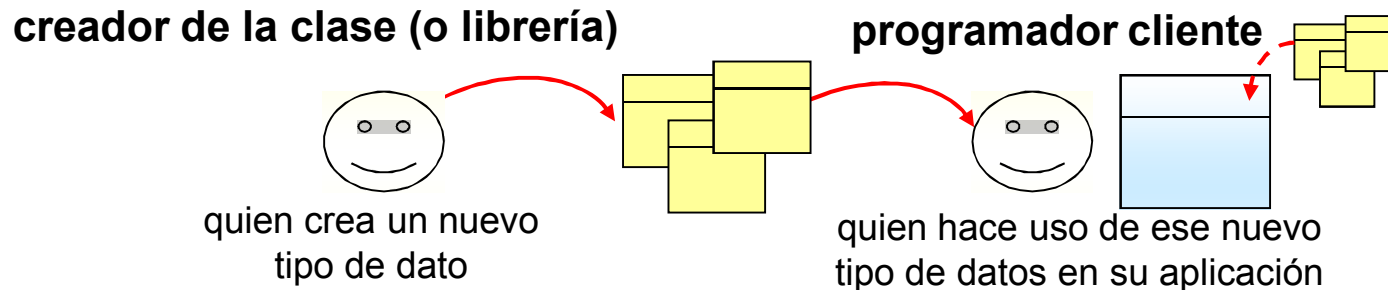


Especificadores de Acceso

- Librería de componentes: **package**
 - Creación de librerías: paquete.
 - Nombres únicos de clases
 - Archivos **JAR** (Java ARchive)
- Especificadores de acceso
 - `public`
 - `protected`
 - `private`
 - acceso por defecto (`package`)
- Especificadores y constructores

Control de acceso

¿Qué podría pasar si se modifica una librería de componentes que está siendo usada por otros programadores?



El código podría romperse !!

El creador de la clase debe sentirse libre para mejorar el código y el programador cliente no debería escribir nuevamente su código, si la librería cambia.

¿Cómo se asegura esto?

- (1) **Por convención**: no quitar métodos existentes en la versión anterior.
- (2) **Usando especificadores de acceso**, para indicarle al programador cliente, qué esta disponible y que no lo está.

Antes de entrar en especificadores de acceso, falta responder una pregunta útil en este contexto: ¿cómo se crea una librería de componentes en java?

Librería de componentes

Paquetes java

- En Java, una librería de componentes (clases e interfaces) es un grupo de archivos **.class**, también llamada **paquete**.
- Para agrupar componentes en una librería o paquete, debemos anteponer la palabra clave **package + nombre de paquete** al comienzo del archivo fuente de cada una de las componentes.

```
package graficos;  
public class Rectangulo {  
    . . .  
}
```

Establece que la clase Rectangulo pertenece al paquete graficos

- Las clases e interfaces que se definen sin usar la sentencia package, se ubican en un paquete sin nombre, llamado default package.

```
public class HolaMundo {  
    . . .  
}
```

Establece que la clase HolaMundo pertenece al paquete por defecto.

Librería de componentes

Paquetes java

El nombre del paquete, es parte del nombre de la clase. El nombre completo de la clase o nombre canónico lleva adelante del nombre de la clase, el nombre del paquete.

<code>graficos.Rectangulo</code>	→	Nombre completo de la clase Rectangulo
<code>java.util.Vector</code>	→	Nombre completo de la clase Vector

Para usar la clase `Rectangulo` desde clases que no pertenecen al paquete `graficos`, se debe usar la palabra clave `import` o especificar el nombre completo de la clase:

```
package ar.edu.unlp.ayed;
import graficos.Rectangulo;
// import graficos.*;

class Figuras {
    Rectangulo r = new Rectangulo();
}
```

```
package ar.edu.unlp.ayed;

class Figuras {
    graficos.Rectangulo r;
    r = new graficos.Rectangulo();
}
```

La sentencia `import` permite usar el nombre corto de la clase, en todo el código fuente. Si no se usa el `import` se debe especificar el nombre completo de la clase.

Librería de componentes

Paquetes java

¿qué sucede si se crean 2 clases con el mismo nombre?

Supongamos que 2 programadores escriben una clase de nombre Vector, en el paquete default
=> aquí se plantea un conflicto de nombres .

Lo que se debe hacer es crear nombres únicos para cada clase para evitar colisión de nombre.

```
package util;  
public class Vector {  
    ...  
}
```

```
package ayed.estructuras;  
public class Vector {  
    . . .  
}
```

¿Qué sucede si se importan dos librerías que incluyen el mismo nombre de clase?

```
import ayed.estructuras.*;  
import util.*;  
...  
Vector vec1 = new Vector();  
ayed.estructuras.Vector vec2 = new ayed.estructuras.Vector();
```

} Ambas contienen la clase Vector

Colisión! ¿A qué clase hace referencia?: el compilador no puede determinarlo, fuerza a escribir el nombre completo de la clase

ok!

Librería de componentes

Paquetes java

- Las clases e interfaces que son parte de distribución estándar de JAVA, están agrupadas en paquetes de acuerdo a su funcionalidad. Algunos paquetes son:

<code>java.lang</code>	clases básicas para crear aplicaciones
<code>java.util</code>	librería de utilitarios (tablas de hashing, vectores, etc.)
<code>java.applet</code>	manejo de Applets
<code>java.io</code>	manejo de entrada/salida
<code>java.awt / javax.swing</code>	manejo de GUI (Graphic User Interface)

- Los únicos paquetes que se importan automáticamente (no requieren usar la sentencia `import`) son el paquete `java.lang` y el paquete actual (paquete en el que estamos trabajando).

```
package ayed.estructuras;  
public class Vector {  
    . . .  
}
```

Se pueden usar, sin importar
todas las clases del paquete
ayed.estructuras



Nota: se recomienda usar como primera parte del nombre del paquete, el nombre invertido del dominio de Internet, de manera de tener menos probabilidad de encontrar un mismo nombre de clase en un mismo nombre de paquete. Usar minúscula para nombres de paquetes e inicial mayúscula para nombres de clases.

Ejemplos `ar.edu.unlp.graficos`
`com.sun.image`

Paquetes JAVA


Nombres únicos

Un paquete, normalmente esta formado por varios archivos `.class`. Para mantenerlos ordenadas, Java hace uso de la estructura jerárquica de directorios del Sistema Operativo (SO) y ubica todos los `.class` de un mismo paquete en un directorio. De esta manera, se resuelve:

- el nombre único del paquete
- la búsqueda de los `.class` (que de otra forma estarían diseminados en el disco)

```
package ar.edu.unlp.utiles;
```

```
public class Vector {  
    . . .  
}
```



`\ar\edu\unlp\utiles\Vector.class`

Cuando el intérprete Java, ejecuta un programa y necesita localizar dinámicamente un archivo `.class` (cuando se crea un objeto o se accede a un miembro `static`), procede de la siguiente manera:

- Busca en los **directorios estándares** (donde está instalado el **JRE** y en el actual)
- Recupera la variable de entorno **CLASSPATH**, que contiene la lista de directorios usados como raíces para buscar los archivos `.class`. Comenzando en la raíz, el intérprete toma el nombre del paquete (de las sentencias `import`) y reemplaza cada “.” por una barra “\” o “/” (según el SO) para generar un camino donde encontrar las clases, a partir de las entradas del CLASSPATH.

Paquetes JAVA

Nombres únicos


Consideremos el dominio unlp.edu.ar, invirtiendo obtenemos un nombre de dominio único y global: `ar.edu.unlp`. Si nosotros queremos crear un paquete `utiles` con las clases `Vector` y `List`, tendríamos:

```
package ar.edu.unlp.utiles;  
public class Vector {  
    . . .  
}
```

```
package ar.edu.unlp.utiles;  
public class List {  
    . . .  
}
```

Supongamos que a ambos archivos los almacenamos en un directorio del disco (`c:\tallerjava\`).

`C:\tallerjava\ar\edu\unlp\utiles\Vector.class`
`C:\tallerjava\ar\edu\unlp\utiles>List.class`



¿a partir de donde comienza el intérprete a buscar el directorio `\ar\..`? A partir de alguna de las entradas indicadas en la variable de entorno **CLASSPATH**:

CLASSPATH = `.;c:\tallerjava;c:\java\librerias`



Esta variable puede contener muchas entradas separadas por ";"

Paquetes JAVA

Organización de archivos – Formato JAR

Es posible agrupar archivos `.class` de uno o más paquetes, en un único archivo con extensión `jar` (Java ARchive). El formato JAR usa el formato zip. Los archivos JAR son multiplataforma. Es posible incluir además de archivos `.class`, archivos de imágenes y audio, etc.

El J2SE o JDK tiene una herramienta que permite crear archivos JAR, desde la línea de comando, es el utilitario `jar`.

Por ejemplo: ubicados en el directorio donde están los archivos `.class`:

```
c:\tallerjava\ar\edu\unlp\utiles\jar cf utiles.jar *.class
```

- En este caso, en el **CLASSPATH**, se especifica el nombre del archivo `jar`:

```
CLASSPATH = .; c:\utiles.jar;c:\java\librerias
```

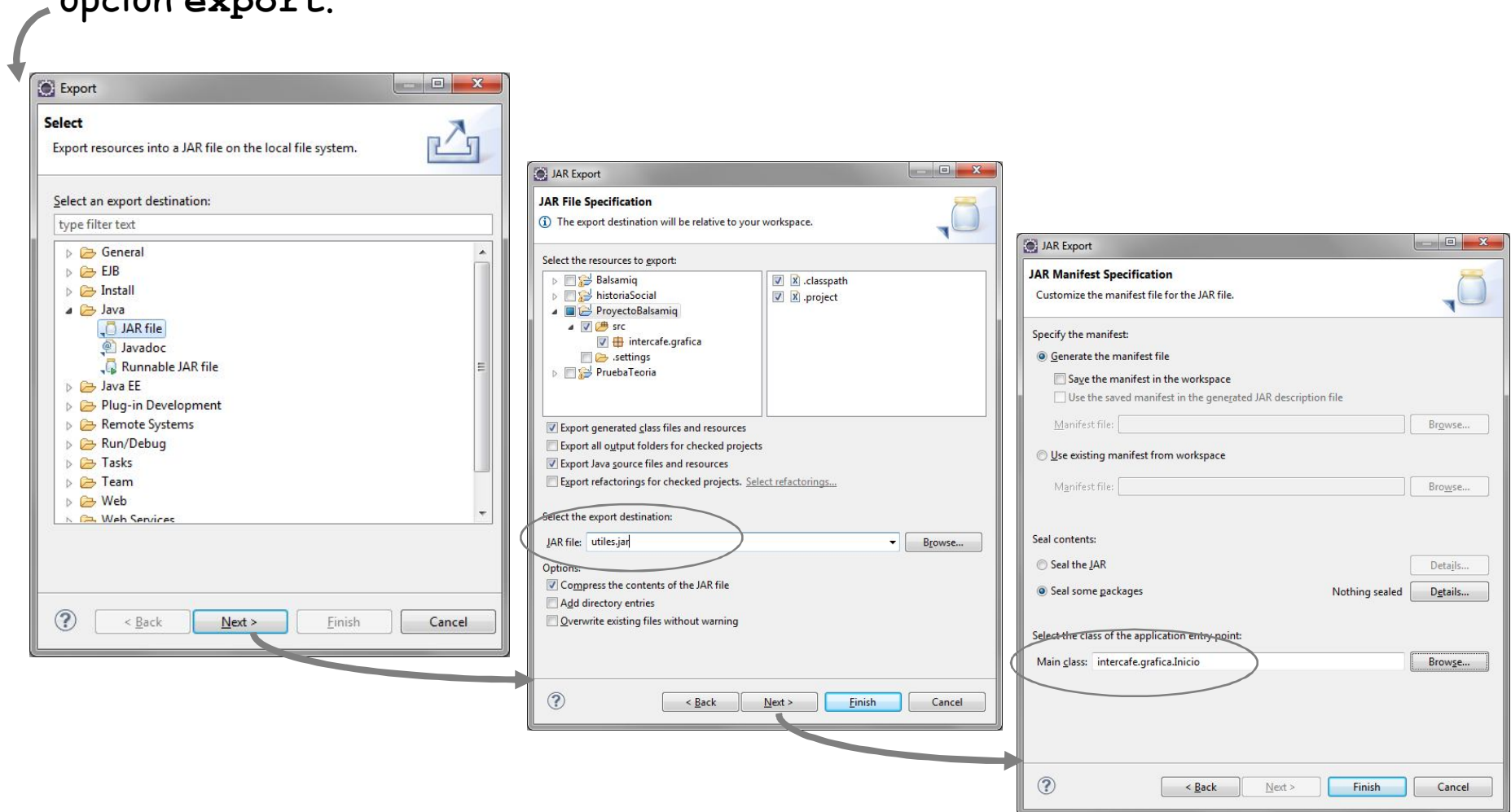
 Los archivos jar pueden ubicarse en cualquier lugar del disco

- El intérprete Java se encarga de buscar, descomprimir, cargar e interpretar estos archivos.

Paquetes JAVA

Organización de archivos – Formato JAR

El archivo JAR también puede construirse desde un proyecto Eclipse, con la opción **export**.

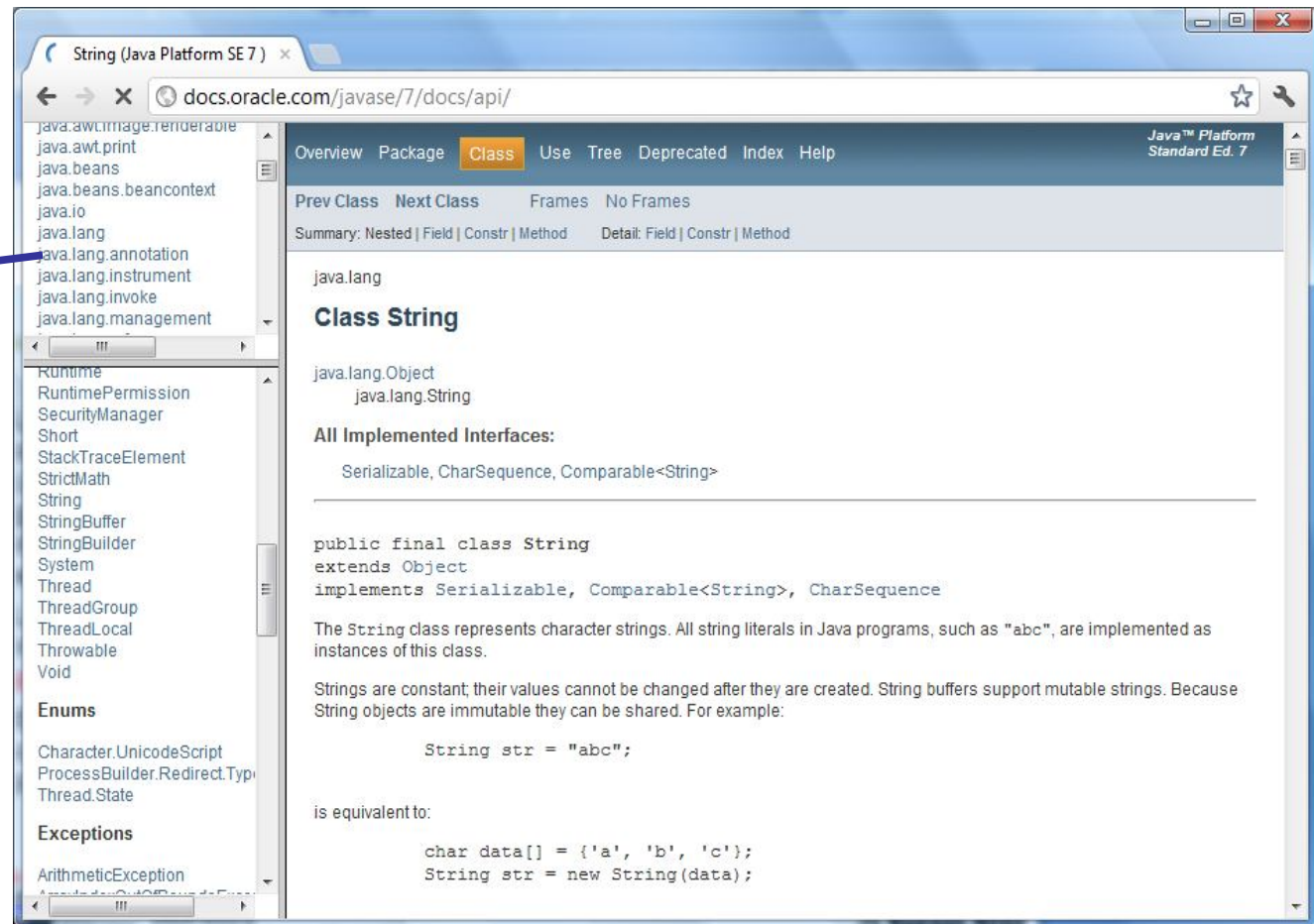


Paquetes en java

La API (Application Programming Interface)

La API JAVA es una colección de clases y otras componentes de software compiladas (archivos .class) que proveen una amplia gama de funcionalidades como componentes de GUIs, I/O, manipulación de colecciones, etc.

La API está agrupada en librerías de clases e interfaces Java relacionadas, llamadas paquetes. Las componentes de los paquetes pueden utilizarse para crear aplicación nuevas.



Especificadores de acceso

- Permiten al autor de una librería de componentes establecer qué está disponible para el usuario (programador cliente) y qué no. Esto se logra usando alguno de los siguientes especificadores de acceso:

public	protected	package (no tiene palabra clave)	private
---------------	------------------	---	----------------

más libre (+)

más restrictivo (-)

- El control de acceso permite ocultar la implementación. Le permiten a un programador de una librería, limitar el acceso a la misma, para posteriormente poder hacer cambios que no afecten al código del usuario de dicha librería.
- En Java, los especificadores de acceso se ubican adelante de la definición de cada método y atributo de la clase.

```
package tp03.ejercicio7;
import tp03.ejercicio6.ListaEnlazadaGenerica;
import tp03.ejercicio6.ListaGenerica;
public class PilaGenerica<T> {
    private ListaGenerica<T> datos;
    public Pila(){
        datos = new ListaGenericaEnlazada<T>();
    }
    . . .
}
```

**El especificador,
solamente controla
el acceso a dicha
definición**

Especificadores de acceso en miembros

¿Qué pasa si a un miembro de una clase no se le define especificador de acceso?

- Tiene acceso por defecto, no tiene palabra clave y comúnmente se lo llama acceso **package** o **friendly**. Implica que tienen acceso a dicho miembro, sólo las clases ubicadas en el mismo paquete que él.
- El acceso **package** le da sentido a agrupamiento de clases en paquetes.

```
package ar.edu.unlp.ayed;
public class Cola {
    Lista elementos;
    Cola() {
        elementos = new Lista();
    }
    Object pop() {
        return elementos.getFirst();
    }
    void push(Object o) {
        elementos.addLast(o);
    }
    . . .
}
```

```
package ar.edu.unlp.ayed;
```

```
public class Estructuras {
```

```
    public static void main(String[] args) {
```

```
        Cola cola1 = new Cola();
```

```
        cola1.push(1);
```

```
        cola1.elementos=new Lista();
```

```
    }
}
```



El acceso es válido
porque pertenecen
al mismo paquete

¿Qué pasa si elimino las líneas **package ar.edu.unlp.ayed;** en ambas definiciones de las clases?

Especificadores de acceso en miembros

public

- El atributo o método declarado public está disponible para TODOS. Cualquier clase con cualquier parentesco, que pertenezca a cualquier paquete tiene acceso.
- Esto es útil para los programadores que hacen uso de la librería o paquete.

```
package ar.edu.unlp.ayed;
public class Cola {
    public Lista elementos;
    public Cola() {
        elementos = new Lista();
    }
    Object pop() {
        return elementos.getFirst();
    }
    public void push(Object obj) {
        elementos.addLast(obj);
    }
    . . .
}
```

```
package taller.ayed;
import ar.edu.unlp.ayed.*;
```

```
public class Estructuras {
```

```
    public static void main(String[] args) {
```

```
        Cola cola1 = new Cola();
```

✓ La clase es pública y el constructor es público, es posible crear objetos Cola

```
        cola1.elementos=new Lista();
        cola1.push(1);
```

✓ También es posible acceder a sus miembros.

```
    }
}
```

¿Qué pasa si se agrega la línea
cola1.pop()?

Especificadores de acceso en miembros

private


- El atributo o método declarado `private` solamente está accesible para la clase que lo contiene. Los miembros `private` están disponible para su uso adentro de los métodos de dicha clase. Cualquier método que se considere “*helper*” para la clase, se define `private`.



```
package ar.edu.unlp.ayed;
public class Cola {
    private Lista elementos;
    private Cola() {
        elementos = new ListaEnlazada();
    }
    public static Cola getCola() {
        // Se podría hacer algun tipo de control
        return new Cola();
    }

    public Object pop() {...}
    public void push(Object o) {...}
    . . .
}
```

```
package ar.edu.unlp.ayed;

public class Estructuras {
    public static void main(String[] args) {

        Cola c1 = new Cola(); 

        Cola c2 = Cola.getCola(); 
        c2.elementos = new ListaEnlazada();
    } 
}
```

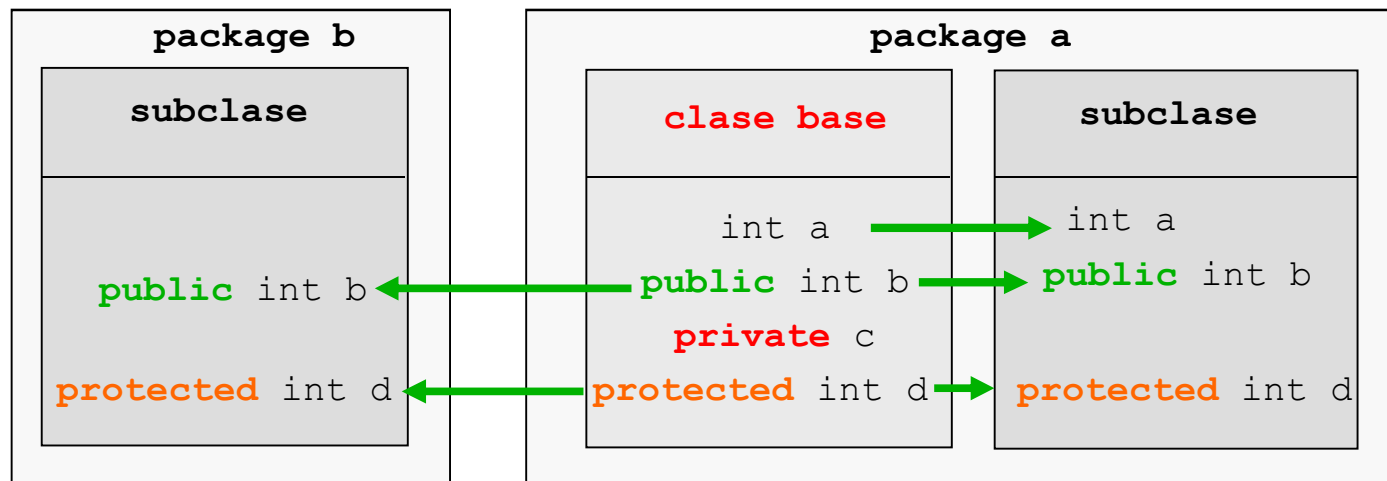
La herencia se implementa a través de la invocación de constructores de las superclases hasta alcanzar la clase `Object`. En este ejemplo el constructor de la clase `ColaPrioridades`, intenta invocar al constructor de la clase `Cola`, el cual es inaccesible debido a que está definido como `privado`.

Especificadores de acceso en miembros

protected

La palabra `protected`, está relacionada con la herencia:

- Si se crea una subclase en un paquete diferente que el de la superclase, la subclase tiene acceso sólo a los miembros definidos `public` de la superclase.
- Si la subclase pertenece al mismo paquete que la superclase, entonces la subclase tiene acceso a todos los miembros declarados `public` y `package`.



¿Puedo definir que un miembro sea accedido por todas las clases hijas?

Si !! esto es `protected`. Un miembro `protected` puede ser accedido por las subclases definidas en cualquier paquete.

Además `protected` provee acceso `package`, es decir las clases del mismo paquete también pueden acceder a sus miembros.

Especificadores de acceso en miembros

protected

```
package ar.edu.unlp.ayed;
public class Lista {
    private Nodo inicio;
    private Nodo actual;
    public boolean add(String elto){
        Nodo nodo = new Nodo(elto);
        if (actual == null)
            inicio = nodo;
        else {
            nodo.setNext(actual.getNext());
            actual.setNext(nodo);
        }
        actual = nodo;
        return true;
    }
}
```

```
protected Nodo getActual()
    return actual;
}
```

Si **getActual()** es **protected** es accesible para cualquier subclase de Lista y no es **public**. !!

```
package misListas;
import ar.edu.unlp.ayed.Lista;

public class ListaPosicional extends Lista {

    public String get(int pos) {...}
    public boolean remove(int pos) {...}
    public boolean add(String elto, int pos) {
        Nodo nodo= new Nodo(elto);
        if (this.getActual()==null) {
            .....
        }
    }
}
```

El método **getActual()** existe en la clase **Lista**, entonces también existe en cualquier subclase de **Lista**. Pero, si dicho método tiene acceso **package**, como la clase **ListaPosicional** no está en el mismo paquete que la clase **Lista**, **getActual()** no estaría disponible en **ListaPosicional**.

Especificadores de acceso para clases

- En Java, los especificadores de acceso en clases, se usan para determinar cuáles son las clases disponibles de una librería.
- Una clase declarada `public`, está disponible para cualquier clase, mediante la cláusula `import`. Se pueden crear instancias de la clase (siempre y cuando exista algún constructor público).

```
package gui;  
public class Control {  
    . . .  
}
```

```
package gui;  
public class Soporte {  
    . . .  
}
```

Cualquier clase que importa el paquete: `import gui.*` ve ambas clases!!

Supongamos que la clase **Soporte** la usan clases del paquete `gui`, pero no se quiere que esté accesible a clases pertenecientes a otros paquetes distintos de `gui`, ¿cómo se define?

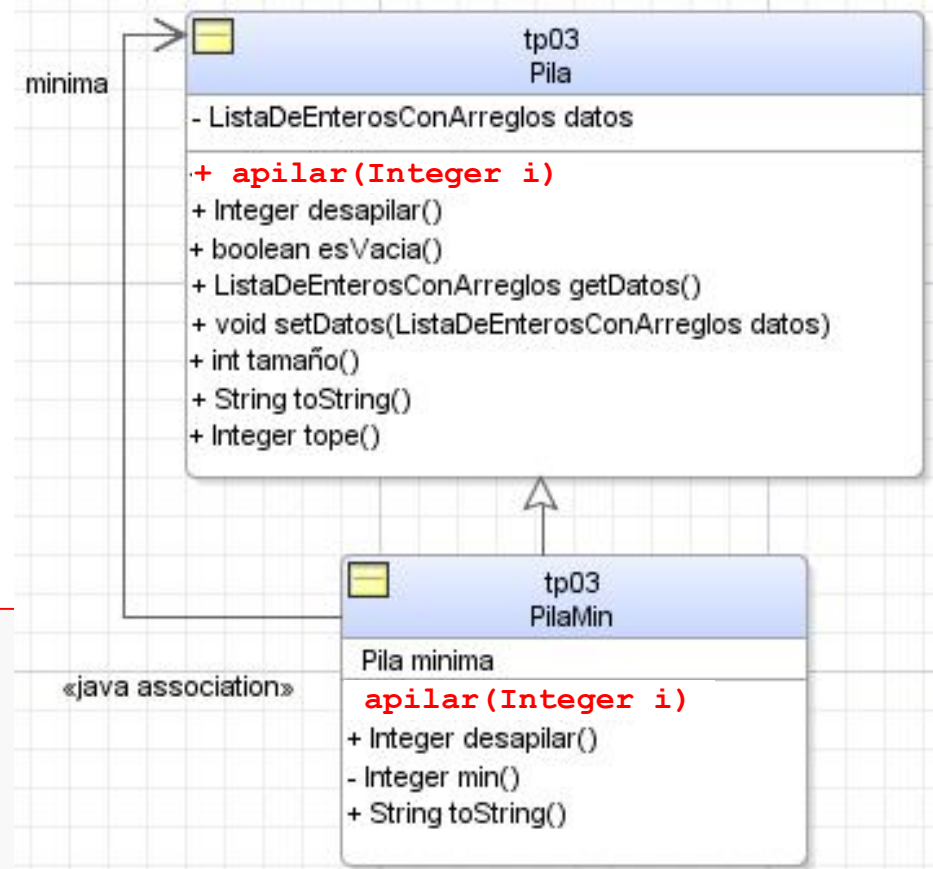
Se la define de acceso `package` y de esta manera, solamente puede usarla las clases del paquete `gui`. Es razonable, que los miembros de una clase de acceso `package` tengan también acceso `package`.

```
package gui;  
class Soporte{  
    . . .  
}
```

La clase `Soporte` puede usarse solamente en el paquete `gui`

Especificadores de acceso y sobrescritura

```
package tp03.accesos;
import tp03.Pila;
import tp03.PilaMin;
public class PilasTest {
    public static void main(String[] args){
        Pila[] pilas = { new Pila(), new PilaMin(), new Pila() };
        for (int i = 0; i < pilas.length; i++) {
            pilas[i].apilar(2*(i+5));
        }
    }
}
```



Los métodos sobrescritos no pueden tener un control acceso más restrictivo que el declarado en la superclase. En las subclases `apilar()`, `#apilar`, `-apilar()` no son válidos.