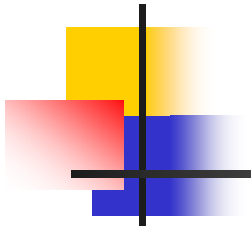
Laboratorio de Programación en Java

Laboratorio de Programación
Curso 2010/2011

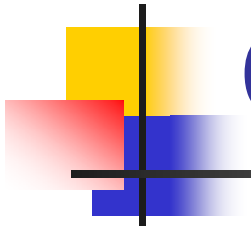
Sergio Ilarri Artigas
silarri@unizar.es



Índice (Bloques)

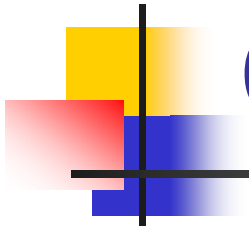
- Introducción
- Sintaxis y Elementos
- Programación OO
- Manejo de Excepciones
- Entorno
- Acceso a Bases de Datos en Java
- Anexos: “Algunos Paquetes y Clases de Uso Común” y “Flujos de E/S”

Introducción



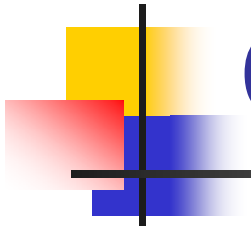
Características (I)

- 1) Orientado a objetos:
  - Objetos: datos + código
  - Objetos + tipos de datos atómicos
  - Todo dentro de una clase
  - Reutilización del software:
    - Librerías de código abierto
- 2) Simple (~ C++, sin punteros)



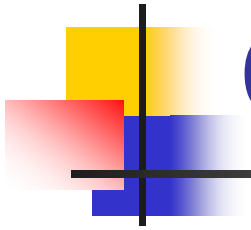
Características (II)

- 3) Portable:
  - a) Compilador de Java: Java bytecodes
  - b) JVM (Java Virtual Machine)
  - Varias implementaciones
  - Interpretado => técnicas de compilación:
    - Compilar a código nativo (=> pérdida de portabilidad)
    - JIT (Just in Time)
    - Recompilación dinámica (partes críticas)



Características (III)

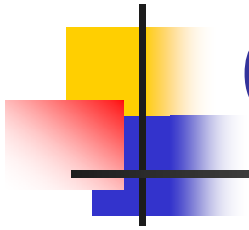
- 4) Robusto:
  - Fuertemente tipado
  - Comprobaciones compilación/ejecución
  - Excepciones
  - No punteros
- 5) Recolección automática de basura:
  - No gestión manual de memoria
  - No hay *memory leaks*... (o casi)
  - *Garbage collector*: objectos inalcanzables



Características (IV)

6) Programación concurrente
7) Distribuido (*RMI*, *CORBA*, *HTTP*, *sockets*, etc.)
8) Seguridad
9) Carga dinámica de clases

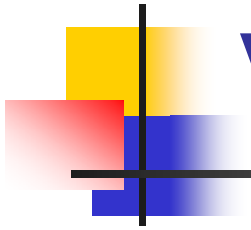
Cuidado con el *ClassNotFoundException*



Características (V)

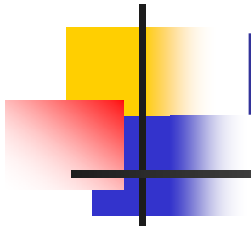
10) Gratis, muchos paquetes, código fuente

11) En sus inicios, se decía que era lento, pero ya no se puede decir lo mismo



Versión Actual de Java

- **2007**: ***Java 1.6*** (Java SE 6)
  - *JavaScript* integrado
  - Soporte para *Ruby*, *Python* y otros lenguajes de script
  - Soporte para servicios web
  - *JDBC* 4.0
  - *JDB* (*Java Database*)
  - Soporte para *NetBeans IDE* 5.5
- **¿Finales de 2010?**: ***Java 1.7***

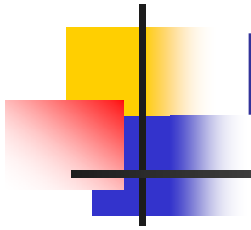


Historia de Java (XIII)

Versión	Año	#clases	#paquetes	Documentación
1.0	1995	212	8	2.65 MB
1.1	1997	504	23	11.5 MB
1.2	1999	1520	59	83.3 MB
1.3	2000	1842	76	120 MB
1.4	2002	2991	135	164 MB
1.5	2004	3278	165	224 MB
1.6	2007	3776	202	259 MB

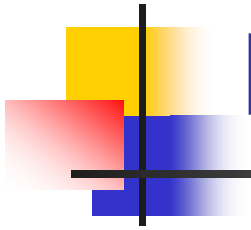
Véase también: http://www.java.com/en/javahistory/timeline.jsp
(The Java History Timeline)

Sintaxis y Elementos



Hello World (I)

```
// Hello.java
public class Hello {
   public static void main(String[] args) {
      System.out.println("Hello, world!");
   }
}
```



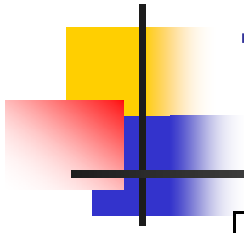
Hello World (II)

- public static void main(String args[])
- Clases:
  - Bloque básico en programación OO:
    - Código = métodos = comportamiento
    - Datos = variables = estado
  - Instanciar = crear un objeto:
    - Instancia <-> variable
    - Clase <-> tipo de dato

Tipos de Datos Primitivos (I)

Keyword	Descripción	Tamaño/formato
(integers)		
byte	Entero de 1 byte de tamaño	8-bit complemento a 2
short	Entero corto	16-bit complemento a 2
int	Entero	32-bit complemento a 2
long	Entero largo	64-bit complemento a 2
(números reales)		
float	Punto flotante simple precis.	32-bit IEEE 754
double	Punto flotante doble precis.	64-bit IEEE 754
(otros tipos)		
char	Un carácter	16-bit Unicode
boolean	Un valor booleano	true, false

¡Fijado!



Tipos de Datos Primitivos (II)

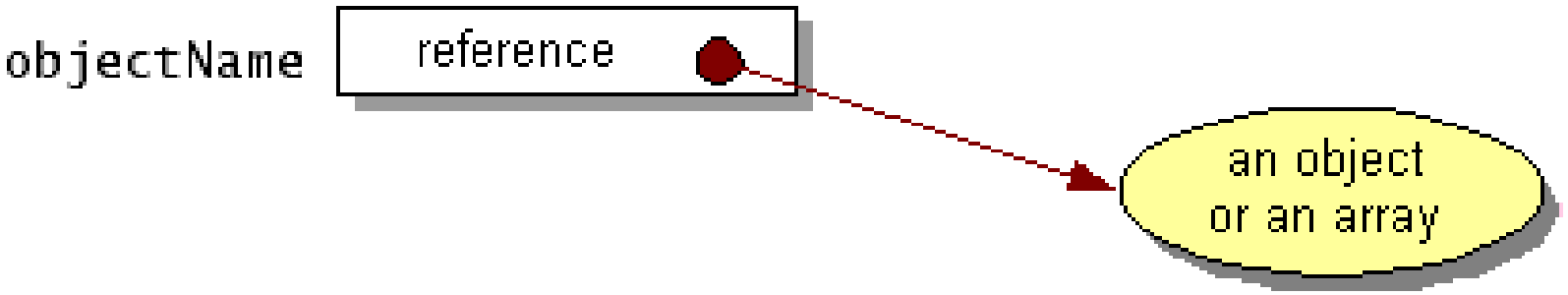
Literal	**Tipo de datos**
178	int
8864L	long
37.266	double
37.266D	double
87.363F	float
26.77e3	double
‘c’	char
true	boolean
false	boolean

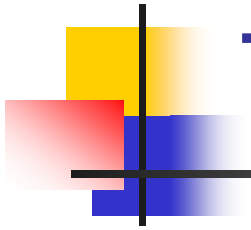
Tipos de Datos Primitivos (III)

```
public class exampleVariables {
  public static void main(String args[]) {
    byte lByte = Byte.MAX_VALUE;
    char aChar = 'S';
    final int aFinalVar = 0;
    final int blankfinal;
    boolean b = true;
    System.out.println("Largest byte:" + lByte);
    System.out.println("Character:" + aChar);
    System.out.println("Boolean value:" + b);
    blankfinal = 0;
}
```

Tipos Referencia (I)

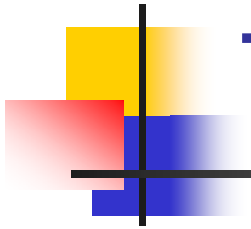
- Los que no son primitivos:
  - Vectores, clases e interfaces.
- Es una dirección al valor.
- Valor especial *null*





Tipos Referencia (II)

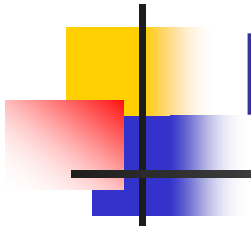
- Operaciones sobre una referencia:
  - Asignación: String a = “Hola a todos”;
  - Comparación (¡de las referencias!):
    - ==
    - !=
- Operaciones sobre el objeto apuntado:
  - Acceso a atributos y métodos: operador “.”
  - (nuevo tipo) refObj
  - miObjeto instanceof miClase



Tipos Referencia (III)

- Manejo de tipos primitivos como objetos:
  - Tipos *wrapper*.
    - Ejemplo: Boolean b = new Boolean(true);
  - *Autoboxing/unboxing* (Java 1.5)

- Paso de parámetros: por valor.
  - Se envía una copia de la referencia.
  - ¡Pero se puede acceder al objeto apuntado!



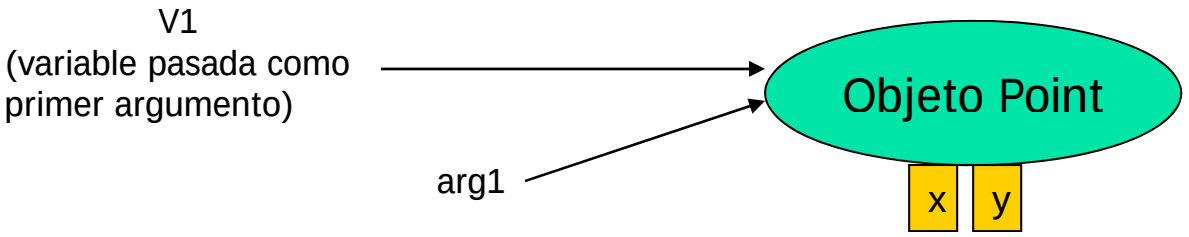
Paso por Valor (I)

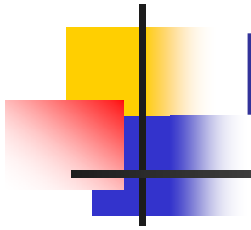
```
public void badSwap(int var1, int var2)
{
   int temp = var1;
   var1 = var2;
   var2 = temp;
}
```

Paso por Valor (II)

```
public void tricky(Point arg1, Point arg2)
{
    arg1.x = 100;
    arg1.y = 100;

    Point temp = arg1;
    arg1 = arg2;
    arg2 = temp;
}
```





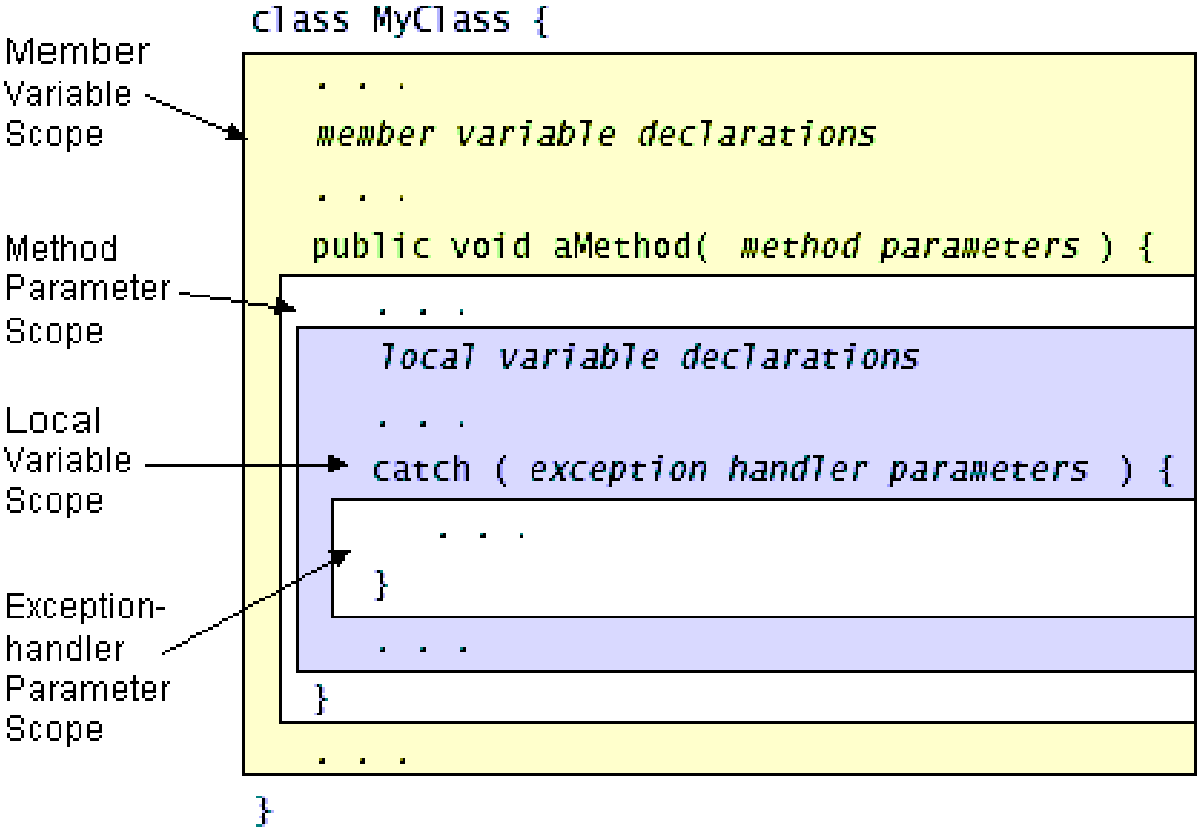
Paso por Valor (III)

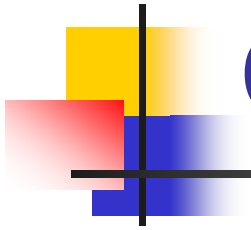
```
public static void main(String [] args)
{
  Point pnt1 = new Point(0,0);
  Point pnt2 = new Point(0,0);
  System.out.println("X: " + pnt1.x + " Y: “ +pnt1.y);
  System.out.println("X: " + pnt2.x + " Y: " +pnt2.y);
  System.out.println(" ");

  tricky(pnt1,pnt2);

  System.out.println("X: " + pnt1.x + " Y:" + pnt1.y);
  System.out.println("X: " + pnt2.x + " Y:“ + pnt2.y);
}
```

Ámbito de las Variables





Operadores Aritméticos (I)

Operador	Uso	Descripción
+	op1 + op2	Suma op1 y op2
-	op1 - op2	Resta op2 de op1
*	op1 * op2	Multiplica op1 y op2
/	op1 / op2	Divide op1 entre op2
%	op1 % op2	Calcula el resto de dividir op1 entre op2

Operadores Aritméticos (II)

```
System.out.println(x++);          System.out.println(++x);
int i = 1 + ++x;                  int i = 1 + x++;
```

Operador	Uso	Descripción
++	op++	Incrementa op en 1; se evalúa al valor de op antes de incrementarse
	++op	Incrementa op en 1; se evalúa al valor de op después de incrementarse
--	op--	Disminuye op en 1; se evalúa al valor de op antes de disminuirse
	--op	Disminuye op en 1; se evalúa al valor de op después de disminuirse

Operadores Relacionales y Condicionales (I)

Operador	Uso	Se evalúa a *true* si
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 mayor o igual que op2
<	op1 < op2	op1 menor que op2
<=	op1 <= op2	op1 menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
!=	op1 != op2	op1 y op2 no son iguales

Operadores Relacionales y Condicionales (II)

Operador	Uso	Se evalúa a *true* si
&&	op1 && op2	op1 y op2 son *true* (cortocircuitado)
||	op1 || op2	op1 o op2 es *true* (cortocircuitado)
!	! op	op es *false*
&	op1 & op2	op1 y op2 son *true*
|	op1 | op2	op1 o op2 es *true*
^	op1 ^ op2	op1 y op2 son diferentes (*xor*)

(0 <= index) && (index < NUM_ENTRIES)

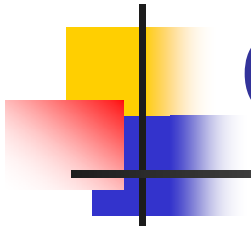
(a != null) && (a.getTime() > 0)

Evita el *NullPointerException*

Operadores de Desplazamiento

Operador	Uso	Significado
>>	op1 >> op2	desplazamiento de op2 bits de op1 a la derecha
<<	op1 << op2	desplazamiento de op2 bits de op1 a la izquierda
>>>	op1 >>> op2	desplazamiento sin signo de op2 bits de op1 a la derecha

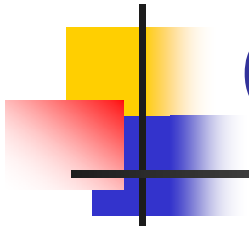
Los bits de la izquierda se rellenan con 0's



Operadores Lógicos

OPERACIONES BIT A BIT

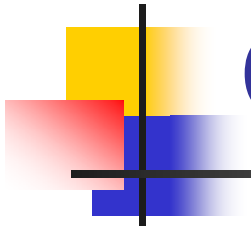
Operador	Uso	Significado
&	op1 & op2	*and* bit a bit
|	op1 | op2	*or* bit a bit
^	op1 ^ op2	*xor* bit a bit
~	~op2	complemento bit a bit



Operadores de Asignación

Operador	Uso	Equivalencia
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

Otros Operadores



- op1 ? op2 : op3
- float[] arrayOfFloats = new float[10];
  arrayOfFloats[6];
- a.hello();
- (args), () → Al declarar o llamar a un método
- (tipo) variable
- Integer anInteger = new Integer(10);
- unObjeto instanceof unaClase

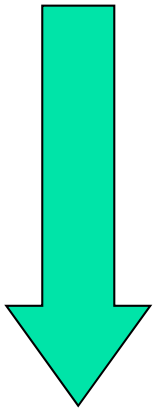
Prioridad de los Operadores

x + y / 100

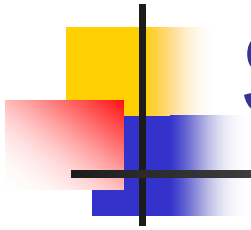
x + (y / 100)

[] . (*params*) *expr*++ *expr*--
++*expr* --*expr* +*expr* -*expr* ~ !
new (*type*)*expr*
* / %
+ -
<< >> >>>
< > <= >= instanceof
== !=
&
^
|
&&
||
? :
= += -= *= /= %= &= ^= |= <<= >>= >>>=

+ prioridad

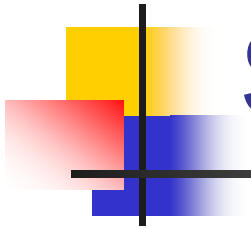


- prioridad



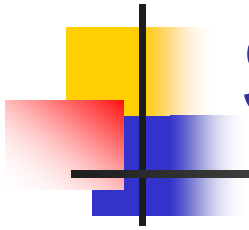
Sentencias (I)

- Unidad completa de ejecución.
- 1) Sentencias de expresión (“**;**”):
  - aValue **=** 8933.234;
  - aValue**++**;
  - System.out.println**(**aValue**)**;
  - Integer integerObject = **new** Integer(4);



Sentencias (II)

- 2) Sentencias de control de flujo:
  - *while*, *do-while* , *for*
  - *if-then-else*, *switch-case*
  - *try-catch-finally*, *throw*
  - *break*, *continue*, *label:*, *return*



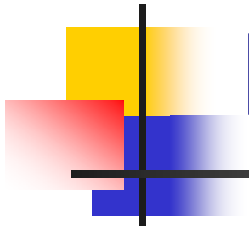
Sentencias (III)

- 3) Sentencias declarativas:
  - double aValue = 8933.234;
- Bloques:

{

... // sentencias

}



Bucles (I)

- `while (expresión) sentencia/bloque`
- `for (inicialización; termin; incr/decr) sentencia/bloque`
- `do sentencia/bloque while (expresión)`

Bucles (II): Ejemplos

- ```
  while (c != 'g') {
     copyToMe.append(c);
     c = copyFromMe.charAt(++i);
  }
  ```

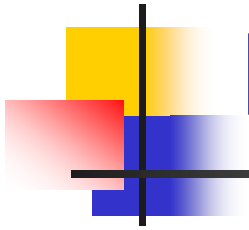
- ```
  for (int i = 1; i <= 100; i++)
     System.out.println(i);
  ```

```
char c = copyFromMe.charAt(i);
do {
   copyToMe.append(c);
   c = copyFromMe.charAt(++i);
} while (c != 'g');
```

Bucles (III): Ejemplos (II)

- ```
  for ( ; ; ) {}
  ```

- ```
  for (int i=0; i < arrayInts.length; i++) {
     System.out.print(arrayOfInts[i] + " ");
  }
  ```



Bucles (IV): break/continue (I)

- *break* interrumpe el bucle más interno
- *continue* pasa a la siguiente iteración

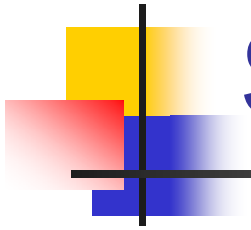
```
for (int i = 1; i <= 100; i++)
{
  if (i % 10 == 0) continue;
  System.out.println(i);
}
```

Bucles (V): break/continue (II)

break y continue con etiquetas

```
buscar:
for (i = 0; i < miArray.length; i++) {
 for (j = 0; j < miArray[i].length; j++) {
    if (miArray[i][j] == loQueBusco) {
       encontrado = true;
       break buscar;
    }
  }
}
```

Búsqueda de un número en un *array* bidimensional



Sentencia Return

- *return*

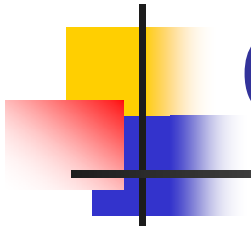
  public void metodo1(...)

  - sale del método actual

- *return* valor

  public <tipo> metodo2(...)

  - sale del método actual y devuelve *valor*



Condicionales: if (I)

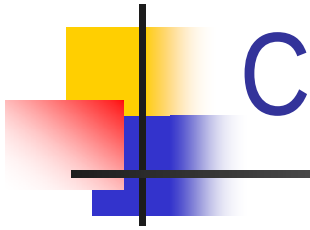
- `if (expresión) sentencia/bloque`
- ```
  if (expresión)
     sentencia1/bloque1
  else
     sentencia2/bloque2
  ```

```
System.out.print("1/x = ");
if (x != 0)
  System.out.println(1/x);
else
{
  System.out.print(“error");
}
```
```

Condicionales: if (II)

- ```
  if (testscore >= 90) { grade = 'A'; }
  else if (testscore >= 80) { grade = 'B'; }
  else if (testscore >= 70) { grade = 'C'; }
  else if (testscore >= 60) { grade = 'D'; }
  else { grade = 'F'; }
  ```

- ```
  if (DEBUG) {
     System.out.println("DEBUG: x = " + x);
  }
  ```



Condicionales: switch

byte, *short*, *char*, *int*, tipos enumerados

```
int mes = 8;
switch (mes) {
   case 1: System.out.println(“Enero"); break;
   case 2: System.out.println(“Febrero"); break;
   case 3: System.out.println(“Marzo"); break;
   case 4: System.out.println(“Abril"); break;
   case 5: System.out.println(“Mayo"); break;
   case 6: System.out.println(“Junio"); break;
   ...
   default: System.out.println(“¡Error!"); break;
}
```

Programación con TADs / Orientada a Objetos

Sergio Ilarri

Implementación de TAD en Java

- Java es un lenguaje orientado a objetos con primitivas suficientes para poder implementar TAD adecuadamente
- Hay varias formas de implementar un TAD en Java, aquí se propone una

TAD Conj. de Caracteres en ADA

nombre del TAD

```
package conjuntos is
type conjcar is private;

procedure vacio(A:out conjcar);
function esVacio(A:in conjcar) return boolean;
procedure poner(c:in character; A:inout conjcar);
function pertenece(c:in character; A:in conjcar) return boolean;
procedure union(A,B:in conjcar; C:out conjcar);
function cardinal(A:in conjcar) return integer;

private
type elementos is array(character) of boolean;
type conjcar is record
elmto:elementos;
card:integer;
end record;
end conjuntos;
```

declaración de operaciones (parte pública)

representación de los valores (parte privada)

TAD Conj. de Caracteres en ADA (II)

```
package body conjuntos is
procedure vacio (A:out conjcar) is
begin
  A.card:=0;
  for c in character loop
    A.elmto(c):=false;
  end loop;
end vacio;
function esVacio (A:in conjcar) return boolean is
begin
  return A.card=0;
end esVacio;
…
end conjuntos;
```

Implementación de las operaciones (parte privada)

TAD Conj. de Caracteres en Java

```
package unizar.labprog;

public interface Conjunto {

public void vacio();
public boolean esVacio();
public void poner(char c);
public boolean pertenece(char c);
public void union(Conjunto B);
public int cardinal();
}
```

nombre del TAD

declaración de operaciones (parte pública)

Diferencias Java / ADA

Además de las obvias de sintaxis…

ADA:
procedure vacio(A:**out** conjcar);

Java:
public void vacio(); // No hace falta pasarle una instancia del TAD como parámetro: el objeto que instancie la clase de Java que implemente esta interfaz estará disponible para todas las operaciones definidas en esa clase (normalmente es implícito, pero pude ser explícito y entonces se le llama this).

TAD Conj. de Caracteres en Java (II)

```
package unizar.labprog;
public class ConjuntoImpl implements Conjunto {
private boolean[] _elementos;
private int _card = 0;
private char _numCars = 256;

public ConjuntoImpl(){
  _elementos = new boolean[_numCars];
  _card = 0;
}
…
public void union(Conjunto B) {
  _card = 0;
  for (char c = 0; c < _numCars; c++) {
    _elementos[c] = _elementos[c] || B.pertenece(c) ;
    if (_elementos[c]) { _card++; }
  }
}
…
}
```

representación de los valores (parte privada)

implementación de las operaciones (parte privada)

Diferencias Java / ADA (II)

Java:

public class ConjuntoImpl **implements** Conjunto**;** // ConjuntoImpl es una implementación del TAD Conjunto; podríamos hacer otras

public ConjuntoImpl() … // Constructor de la clase: inicializa los valores de las instancias de la clase (objetos) que creemos. Si no se implementa ninguno, Java crea uno por defecto (normalmente se implementa al menos uno)

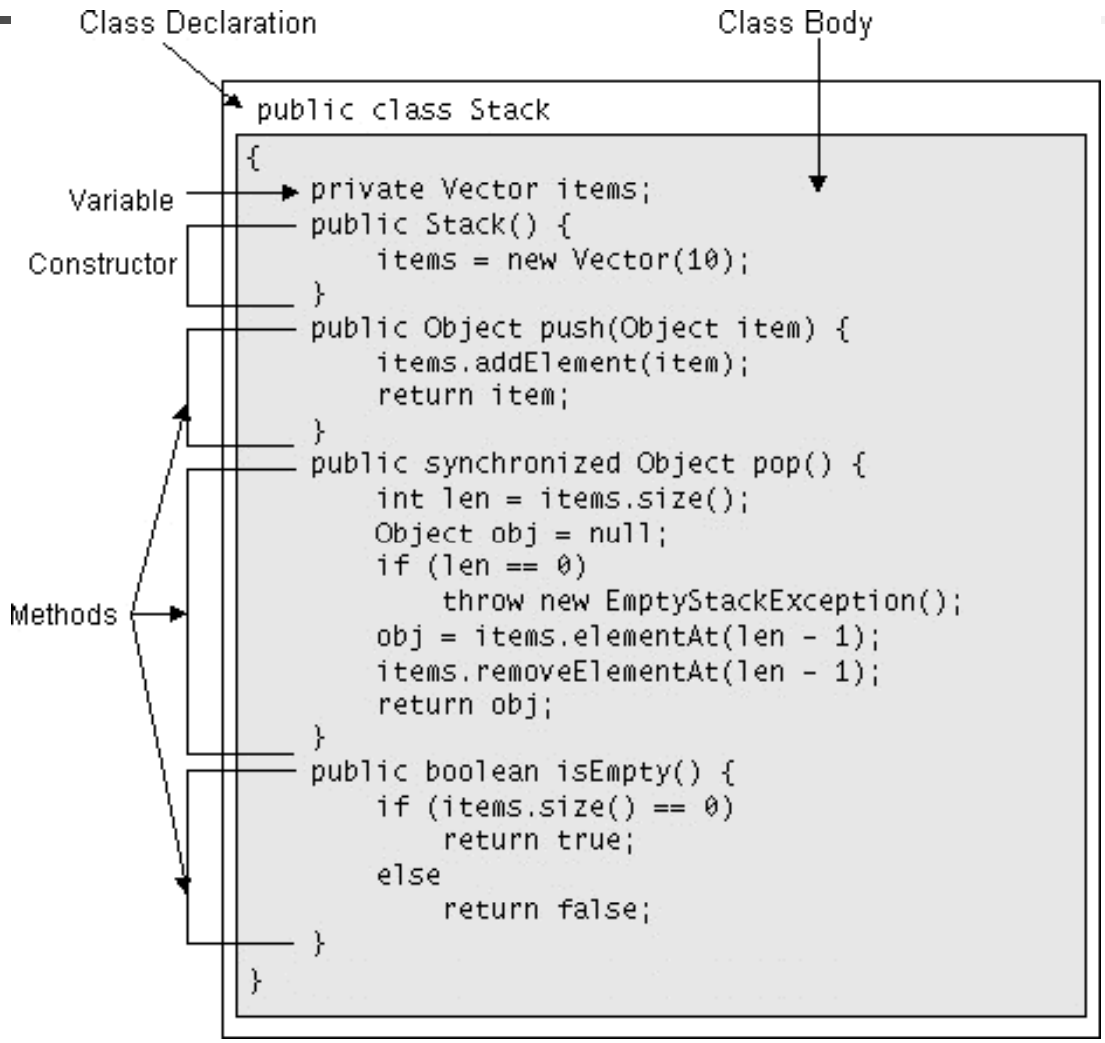
Uso del TAD en Java

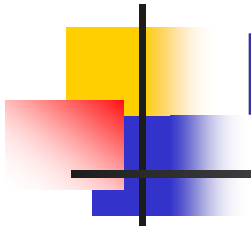
```
package unizar.labprog;

public class TestTAD {
  public static void main(String[] args) {
  // Declaro las variables A y B del TAD Conjunto
  Conjunto A, B;
  // Pero las tengo que instanciar con una implementación
  // concreta
  A = new ConjuntoImpl(); // En esta instanciación se llama al constructor
  B = new ConjuntoImpl(); // En esta instanciación se llama al constructor

  A.poner('a');
  A.poner('b');
  System.out.println("Cardinal de A: " + A.cardinal()); // Escribe Cardinal de A: 2
  B.poner('c');
  System.out.println("Cardinal de B: " + B.cardinal()); // Escribe Cardinal de B: 1
  A.union(B);
  System.out.println("Cardinal de A + B: " + A.cardinal()); // Escribe Cardinal de A+B: 3
  }
}
```

Implementación de Clases





Declaración de Clases

`public`	Class is publicly accessible.
`abstract`	Class cannot be instantiated.
`final`	Class cannot be subclassed.
`class NameOfClass`	***Name of the Class.***
`extends` *`Super`*	Superclass of the class.
`implements` *`Interfaces`*	Interfaces implemented by the class.

`{`
`  `*`ClassBody`*
`}`	

Constructores (I)

```
public Stack() {
   items = new Vector(10);
}

public Stack(int initialSize) {
   items = new Vector(initialSize);
}
```

```
Stack s1 = new Stack(20);
Stack s2 = new Stack();
```

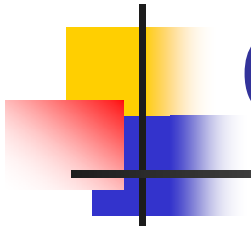
Constructor por defecto (no hace nada)

Constructores (II)

- Se puede llamar al constructor padre:

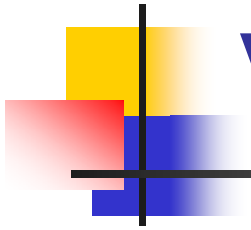
```
class AnimationThread extends Thread {
    public AnimationThread() {
        super("AnimationThread");
        …
    }
    …
}
```

Primera sentencia



Constructores (III)

- Acceso a constructores:
  - *private*
    - ninguna clase más puede invocarlo
    - instanciación a través de *factory methods*
  - *protected*
    - subclases y clases en el mismo paquete
  - *public*
    - cualquiera
  - nada
    - clases en el mismo paquete



Variables miembro

accessLevel	Indicates the access level for this member.
`static`	Declares a class member.
`final`	Indicates that it is constant.
`transient`	This variable is transient.
`volatile`	This variable is volatile.
type name	The type and name of the variable.

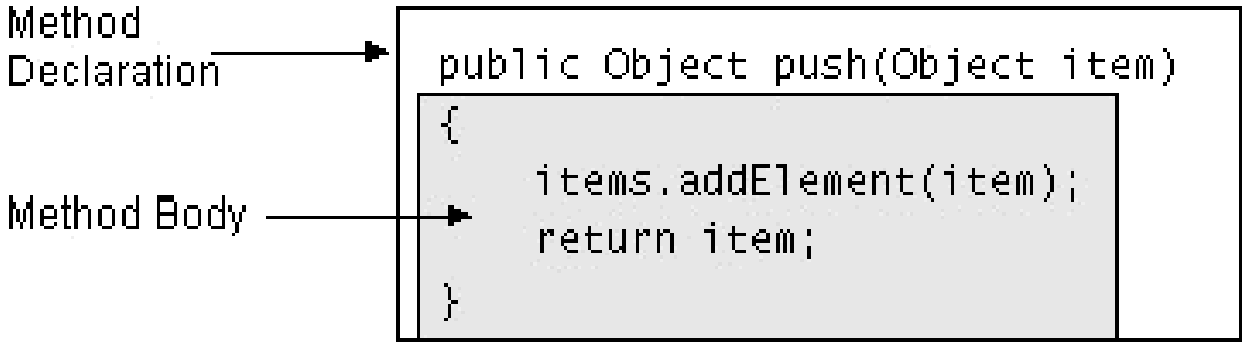
Acceso a variables miembro:

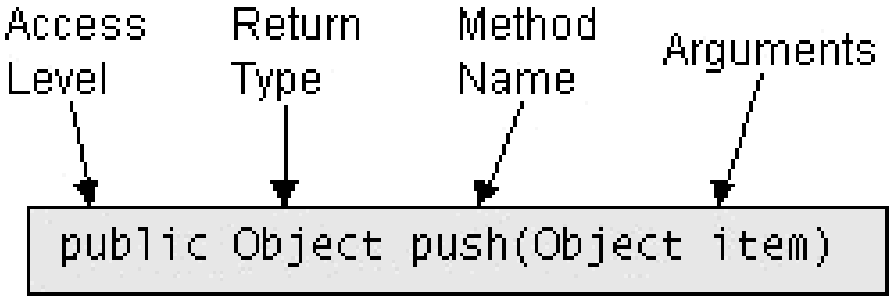
-referenciaObjeto.nombreVariable

-se desaconseja el acceso directo: métodos *get* y *set*

-¿Tiene sentido el valor?
-Independencia tipo/nombre

Métodos: Implementación





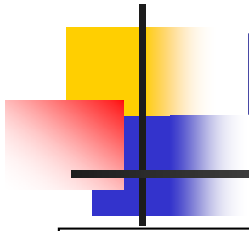
Métodos: Declaración y Acceso

accessLevel	Access level for this method.
`static`	This is a class method.
`abstract`	This method is not implemented.
`final`	Method cannot be overridden.
`native`	Method implemented in another language.
`synchronized`	Method requires a monitor to run.
returnType methodName	The return type and method name.
`(` *paramList* `)`	The list of arguments.
`throws` *exceptions*	The exceptions thrown by this method.

Acceso a métodos:

-referenciaObjeto.nombreMetodo();

-referenciaObjeto.nombreMetodo(argumentos);



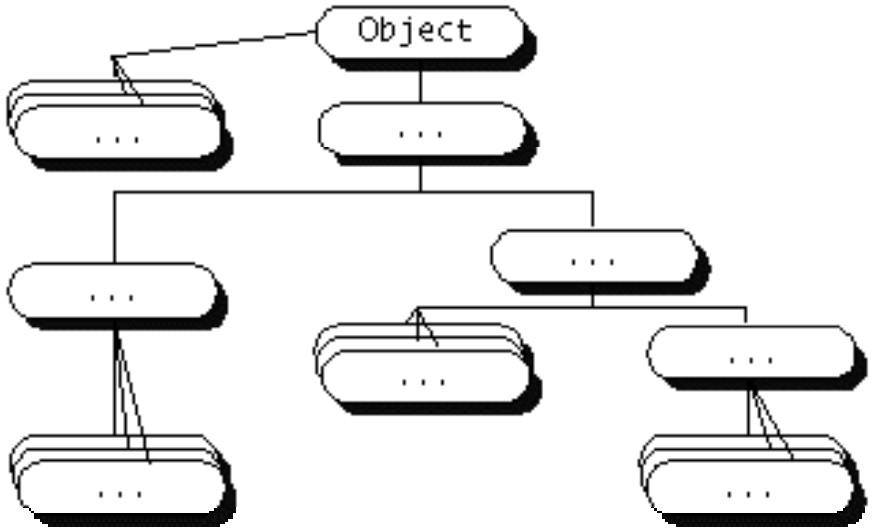
Keywords: *this*

```
class HSBColor {
   int hue, saturation, brightness;

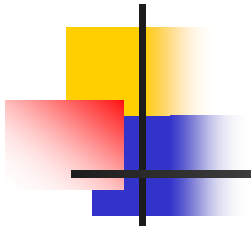
HSBColor (int hue, int saturation, int brightness) {
      this.hue = hue;
      this.saturation = saturation;
      this.brightness = brightness;
   }
```

Evitar ambigüedad
Los argumentos “tienen prioriidad”

Jerarquías de Clases



- *java.lang.Object*
- Palabra clave: *extends*
- Herencia, *overriding*
  - Acceso al padre: *super*

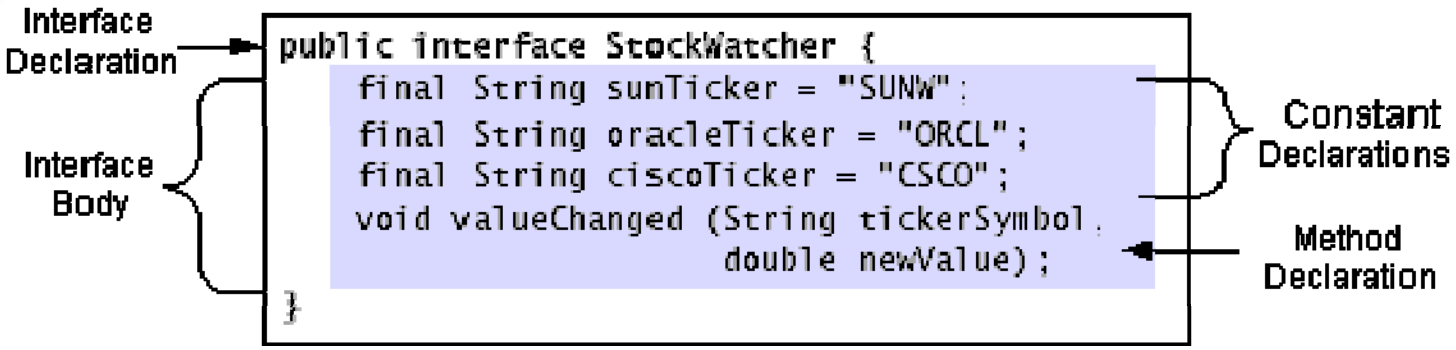


Interfaces (I)

- Protocolo de comportamiento (*contrato*)
- No son parte de la jerarquía de clases
- Un interface puede extender varios
- Puede incluir constantes (*static final*)
- No puede implementar métodos

Diferentes a las clases abstractas

Interfaces (II)



No tiene Sentido que sean privados (contrato)

`public`	Makes this interface public.
`interface InterfaceName`	**This is the name of the interface.**
`Extends SuperInterfaces`	This interface's superinterfaces.

```
{
    InterfaceBody
}
```

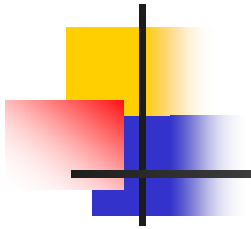
-Métodos implícitamente *public* y *abstract*
-Constantes implícitamente *public*, *static* y *final*
-Interface declarada como *public* o nada (acceso desde el paquete)

Interfaces (III)

```
public class StockApplet extends Applet
  implements StockWatcher {
    ...
}
```

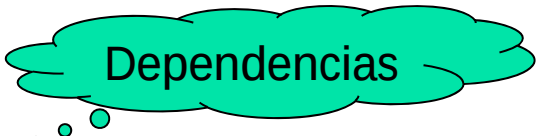
-Herencia múltiple para los interfaces
-Después del *extends*
-Debe implementar todos los métodos o ser abstracta
-Usar constantes de un interface:
    -`StockWatcher.sunTicker`
    -`Implementándolo: sunTicker`

Aunque, por convención, constantes en mayúsculas

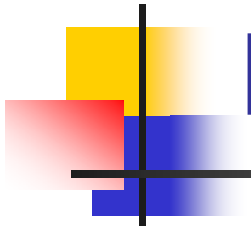


Interfaces (IV)

- Al definir un interface, tienes un nuevo tipo de referencia
- Los interfaces no deben crecer
- Siempre puedes extenderlos




```
public interface StockTracker extends StockWatcher
{
   void currentValue(String tickerSymbol, double
      newValue);
}
```



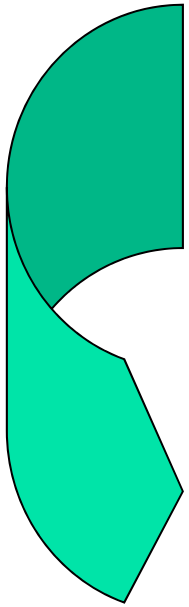
Paquetes (I)

- Espacio de nombres:
  - Clases
  - Interfaces
- Ejemplo: `package graphics;`

Si no,
paquete por defecto

- Evitar colisiones de nombres:
  - com.company.region.package
- Nombres cualificados:
  - graphics.Rectangle <> graphics2.Rectangle

Paquetes (II)



- ¿Cómo usarlos?:
  - 1) Importar el elemento:

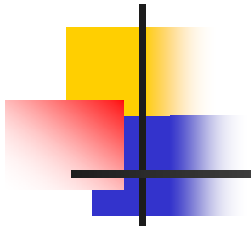
    import graphics.Circle;
  - 2) Importar el paquete completo:

    import graphics.*;
  - 3) Indicar el paquete cuando se necesita:

    graphics.Circle c = new graphics.Circle(...);

```
Circle c = new Circle(...);
```

Nuevos Elementos Introducidos con Java 1.5

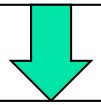


- Genéricos
- Bucle for-each
- Autoboxing/unboxing
- Enumeraciones seguras

Genéricos

```
// Removes 4-letter words from c. Elements must be strings
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
      if (((String) i.next()).length() == 4)
        i.remove();
}
```

Riesgo de *ClassCastException*



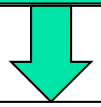
```
// Removes the 4-letter words from c
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
      if (i.next().length() == 4)
        i.remove();
}
```

Más claro y robusto

Bucle *For-Each*

```
void cancelAll(Collection<TimerTask> c) {
    for (Iterator<TimerTask> i = c.iterator(); i.hasNext(); )
        i.next().cancel();
}
```

Riesgo de errores humanos (varias referencias al iterador)



```
void cancelAll(Collection<TimerTask> c) {
    for (TimerTask t : c)
        t.cancel();
}
```

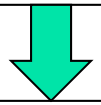
Más claro y sencillo

Bucle *For-Each*

¡NoSuchElementException!

```
List suits = ...;
List ranks = ...;
List sortedDeck = new ArrayList();

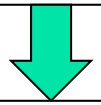
for (Iterator i = suits.iterator(); i.hasNext(); )
    for (Iterator j = ranks.iterator(); j.hasNext(); )
        sortedDeck.add(new Card(i.next(), j.next()));
```



```
for (Iterator i = suits.iterator(); i.hasNext(); ) {
    Suit suit = (Suit) i.next();
    for (Iterator j = ranks.iterator(); j.hasNext(); )
        sortedDeck.add(new Card(suit, j.next()));
}
```

Bucle *For-Each*

```
for (Iterator i = suits.iterator(); i.hasNext(); ) {
    Suit suit = (Suit) i.next();
    for (Iterator j = ranks.iterator(); j.hasNext(); )
        sortedDeck.add(new Card(suit, j.next()));
}
```



Mucho más sencillo y claro

```
for (Suit suit : suits)
    for (Rank rank : ranks)
        sortedDeck.add(new Card(suit, rank));
```

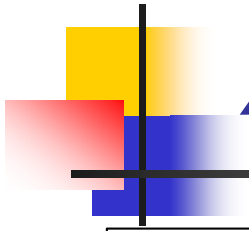
Bucle *For-Each*

```
int sum(int[] a) {
    int result = 0;
    for (int i : a)
        result += i;
    return result;
}
```

También es aplicable a *arrays*

```
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
      if (i.next().length() == 4)
        i.remove();
}
```

Aquí no es aplicable (necesitamos acceso al iterador)



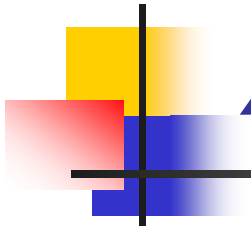
Autoboxing/Unboxing

```
public class Frequency {
   public static void main(String[] args) {
      Map<String, Integer> m = new TreeMap<String, Integer>();
      for (String word : args) {
          Integer freq = m.get(word);
          m.put(word, (freq == null ? 1 : freq + 1));
      }
      System.out.println(m);
   }
}
```

Autoboxing/Unboxing

```
public static List<Integer> asList(final int[] a) {
    return new AbstractList<Integer>() {
        public Integer get(int i) { return a[i]; }
        // Throws NullPointerException if val == null
        public Integer set(int i, Integer val) {
            Integer oldVal = a[i];
            a[i] = val;
            return oldVal;
        }
        public int size() { return a.length; }
    };
}
```

Cierto coste asociado a estas operaciones



Autoboxing/Unboxing

- Alguna advertencia:
  - *Autounboxing* un *Integer* con valor *null* lanzará una *NullPointerException*
  - El operador == compara referencias con expresiones de tipo *Integer* y valores con expresiones de tipo *int*
  - Hay un cierto coste asociado a las operaciones de *unboxing* y *autoboxing*

Enumeraciones Seguras

```
public static final int SEASON_WINTER = 0;
public static final int SEASON_SPRING = 1;
public static final int SEASON_SUMMER = 2;
public static final int SEASON_FALL   = 3;
```

Patrón Enumeración

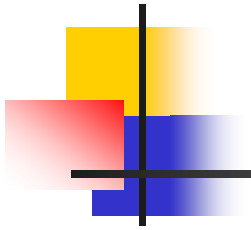
Problemas:

- No hay un tipado seguro: puedo sumar dos valores, puedo pasar un valor fuera del rango 0-3
- No hay espacio de nombres: tengo que prefijar con “SEASON_”
- Como son constantes, se compilan en tiempo de compilación en los clientes (por tanto, si cambian –por ejemplo, si se añaden constantes con valores intermedios- hay que recompilar)

Los valores enteros asignados son arbitrarios: no tiene sentido imprimirlos

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

Manejo de Excepciones



Ideas Básicas

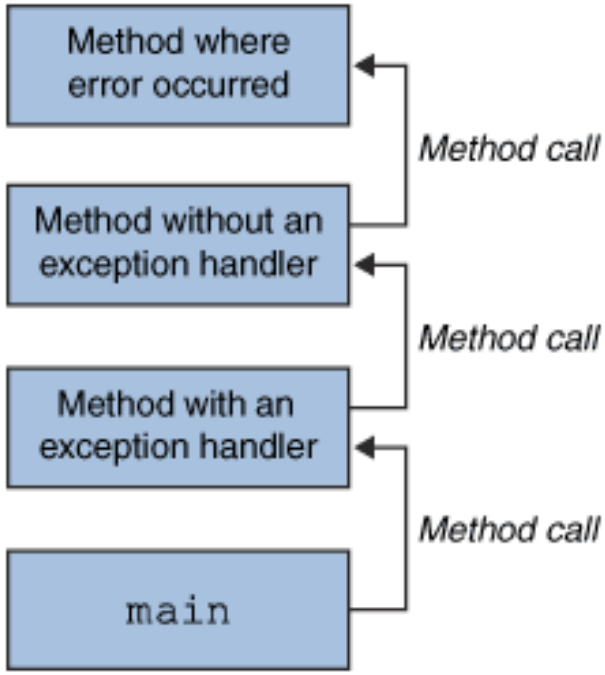
- Excepción: objeto lanzado si ocurre un evento excepcional
- Contiene información del problema
- Interrumpen la ejecución del bloque actual
- Se propagan hacia arriba hasta que son atrapadas

Secuencia de Llamadas

-**Manejador de excepciones**

-Tipo de excepción del manejador

-**Capturar la excepción**

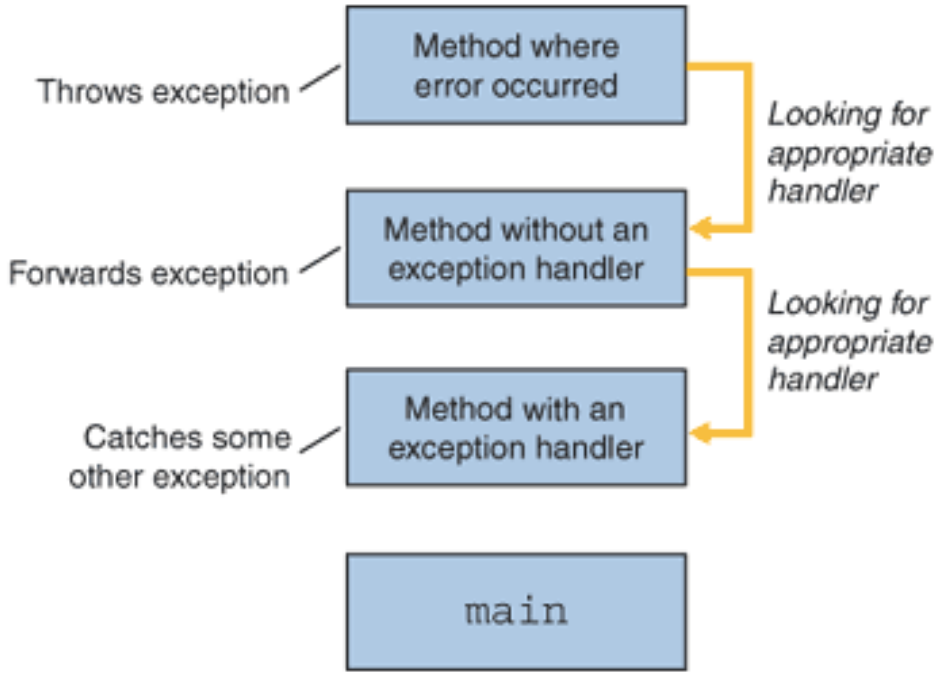


Secuencia de Propagación

-**Manejador de excepciones**

-Tipo de excepción del manejador

-**Capturar la excepción**



Estructura de Manejo de Excepciones (I)

```
try {
    sentencia/s
}
catch (tipoExcepción nombre) {
    sentencia/s
}
finally {
    sentencia/s
}
```

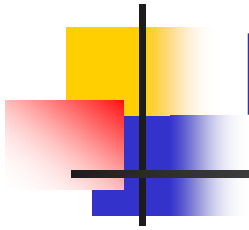
Subclase de *Throwable*

Manejador de excepción

Se dejan de ejecutar las sentencias del bloque *try* que siguen a la que lanza la excepción.
Se pasa el control al *catch* correspondiente.

cleanup code
(ejecutado después del *try*)

Estructura de Manejo de Excepciones (II)



- Bloque *try*
  - Instrucciones que pueden lanzar excepciones
- Bloque *catch*
  - Atrapa excepciones de cierto tipo
  - Puede haber varios
  - La excepción se trata en el primer bloque *catch* que la atrape
  - **e.printStackTrace()**, e.getMessage()
- Bloque *finally*
  - Cierre de ficheros, bases de datos, etc.

Checked Exceptions

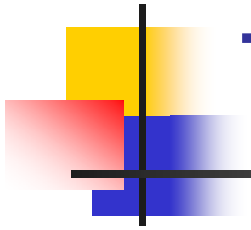
Tipos de Excepciones (I)

- Las excepciones siguientes no requieren manejador:
  - *RuntimeException* (e.g., *NullPointerException, NotSuchIndexException*)

Errores de programación

  - *Error* (e.g., *java.io.IOError*)

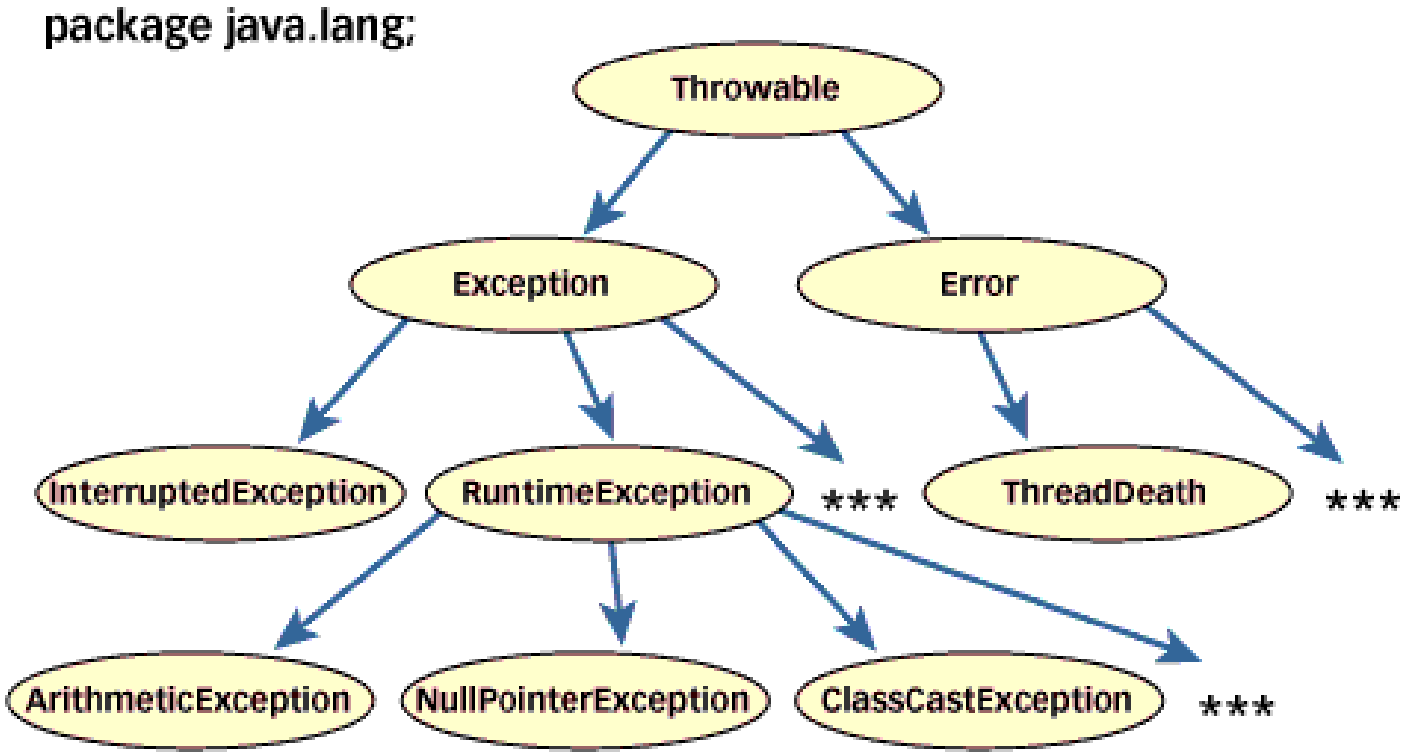
Condiciones externas excepcionales



Tipos de Excepciones (II)

- El resto debemos capturarlas:
  - *Checked exceptions*
- Si no queremos, declarar que el método lanza la excepción
- También podemos relanzar excepciones:
  - throw ioe;
- O lanzar excepciones propias:
  - throw new Exception("...");

Tipos de Excepciones (III)



Ejemplo

```
try {
        System.out.println("Entering try statement");
        out = new PrintWriter(
                  new FileWriter("OutFile.txt"));
            for (int i = 0; i < SIZE; i++)
                out.println("Value at: " + i + " = "
                            + vector.elementAt(i));

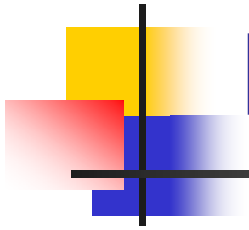
    } catch (ArrayIndexOutOfBoundsException e) {
         System.err.println("Caught "
                    + "ArrayIndexOutOfBoundsException: "
                    +   e.getMessage());

    } catch (IOException e) {
         System.err.println("Caught IOException: "
                            +  e.getMessage());

    } finally {
         if (out != null) {
             System.out.println("Closing PrintWriter");
             out.close();

         }
         else {
             System.out.println("PrintWriter not open");
         }
     }
```

Entorno

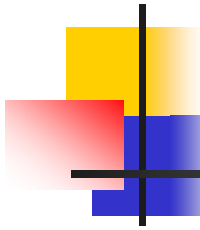


Elementos

- *Java runtime* :
  - *Virtual machine* : *java*
  - *Class libraries*
- *Javac compiler* : *javac*

Java Runtime Environment (*JRE*)

Java Development Kit (*JDK*)



Distribuciones

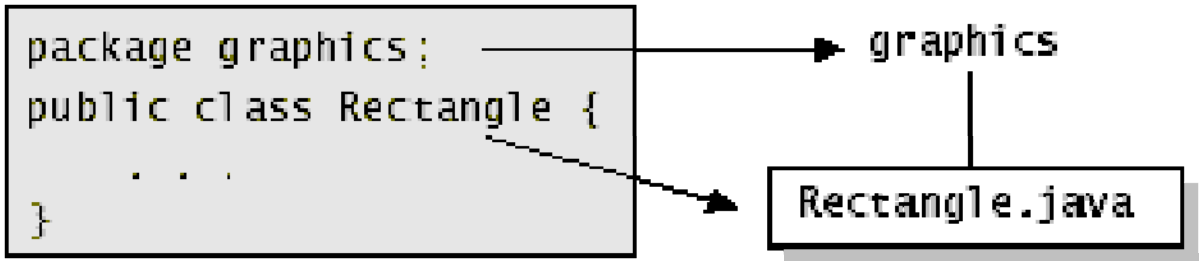
J2SE = Java 2 Standard Edition

J2EE = Java 2 Enterprise Edition

J2ME = Java 2 Micro Edition

Estructura de Directorios (I)

JERARQUÍA DE FICHEROS

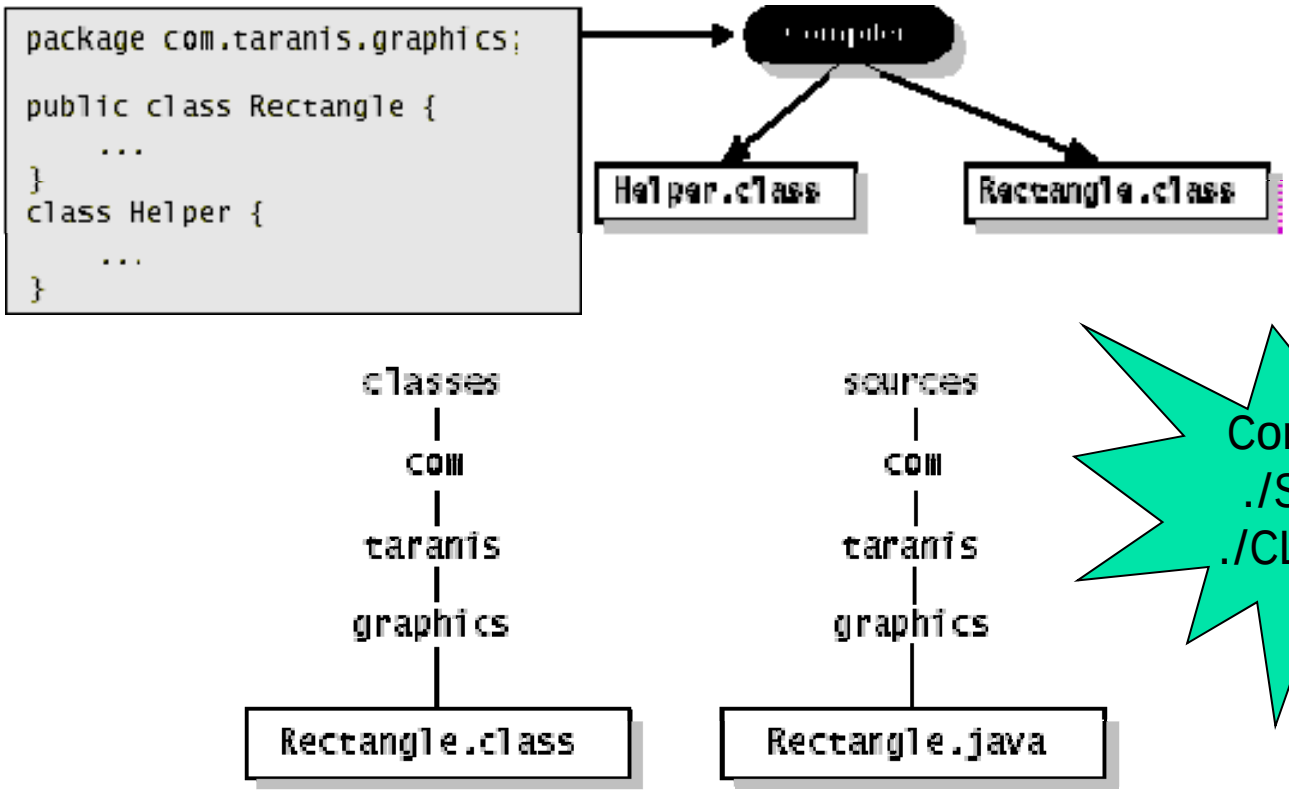


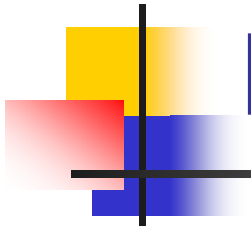
Nombre clase
graphics.Rectangle

Path al fichero
graphics/Rectangle.java

Estructura de Directorios (II)

COMPILACIÓN

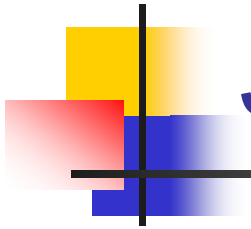




Referencia a Clases

CLASSPATH=...

- Lista de directorios y ficheros .zip y .jar
- Se buscan en orden
- Se incluyen ya por defecto:
  - las clases propias de Java
  - el directorio actual

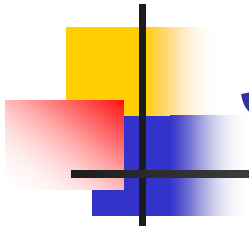


Javac (I)

- Opciones de javac:
  - -classpath classpath
  - -d directory

```
javac -d c:\myclasses *.java
```

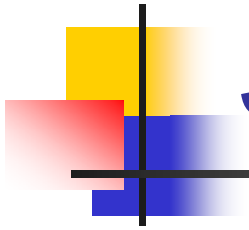
  - -encoding encoding
  - -sourcepath sourcepath
  - -help
  - -target 1.1, ...



Javac (II)

```
C:>javac -classpath \examples;\lib\Banners.jar \
  \examples\greetings\Hi.java
```

Ficheros a compilar (con la extensión .java)



Java (I)

- Opciones de java:
  - -cp classpath (variable CLASSPATH)
  - -Dproperty=value
  - -jar
  - -version
  - -?, -help
  - Opciones no estándar: -Xmsn, -Xmxn, etc.

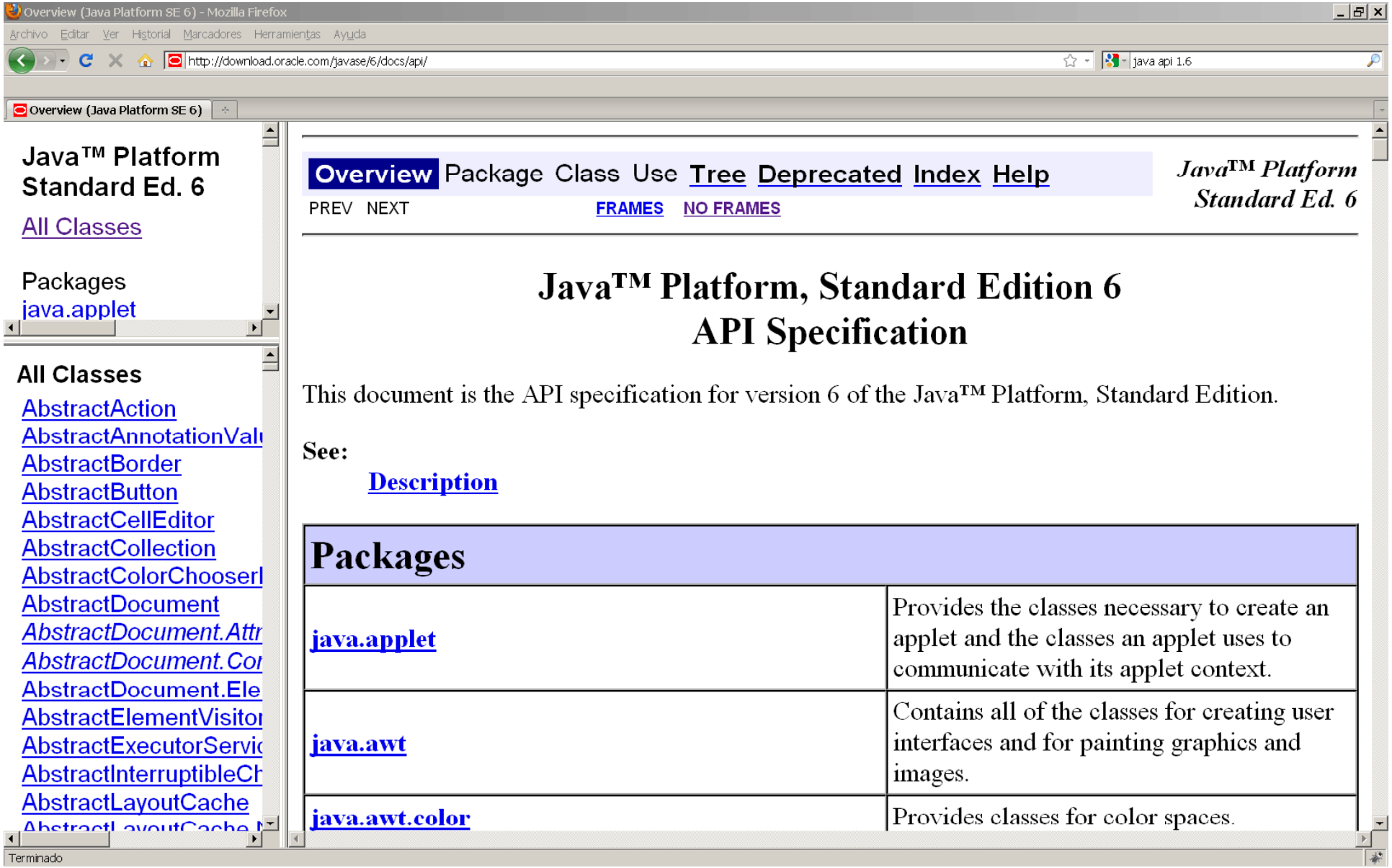
Initial size (bytes)

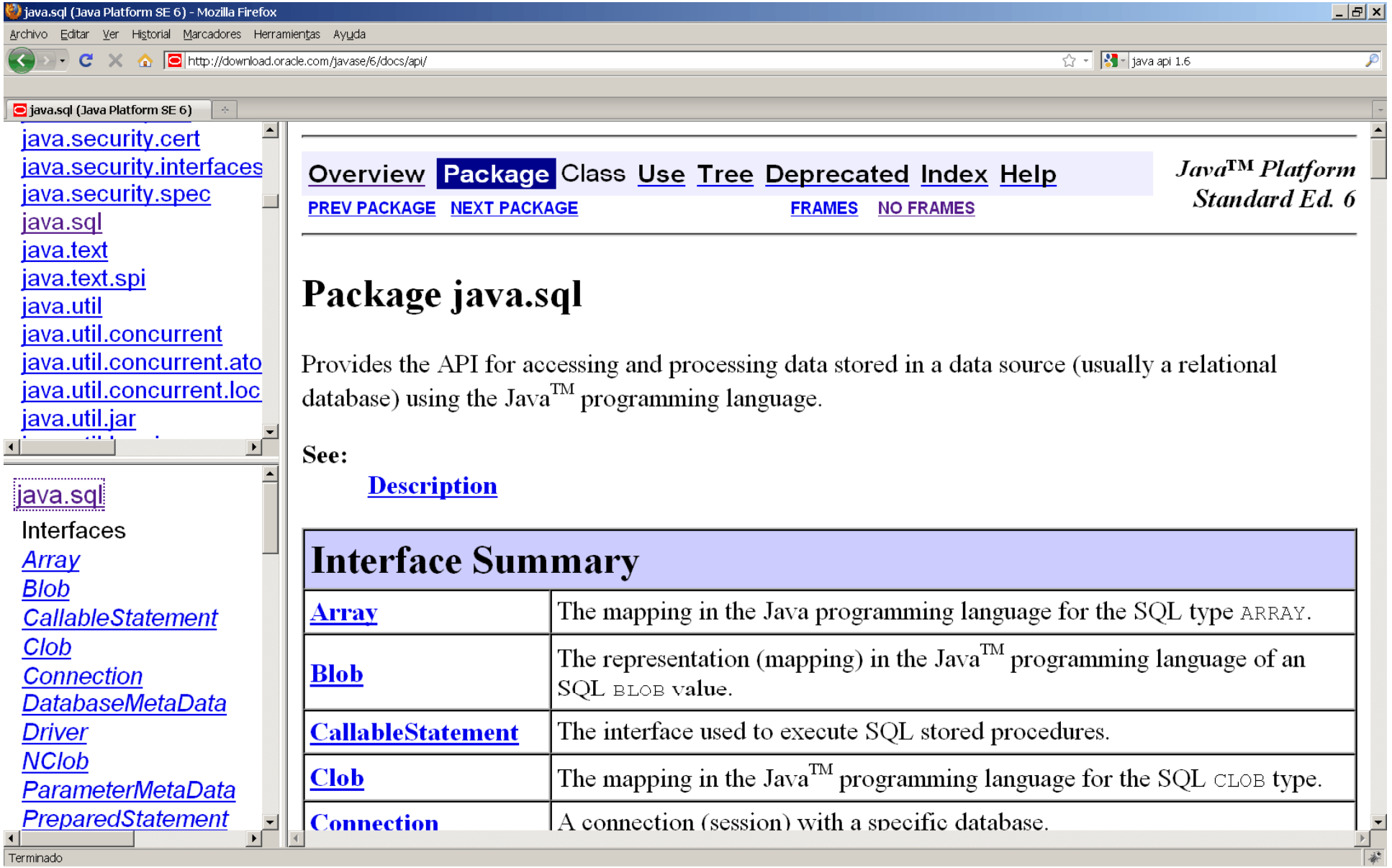
Minimum size (bytes)

Java (II)

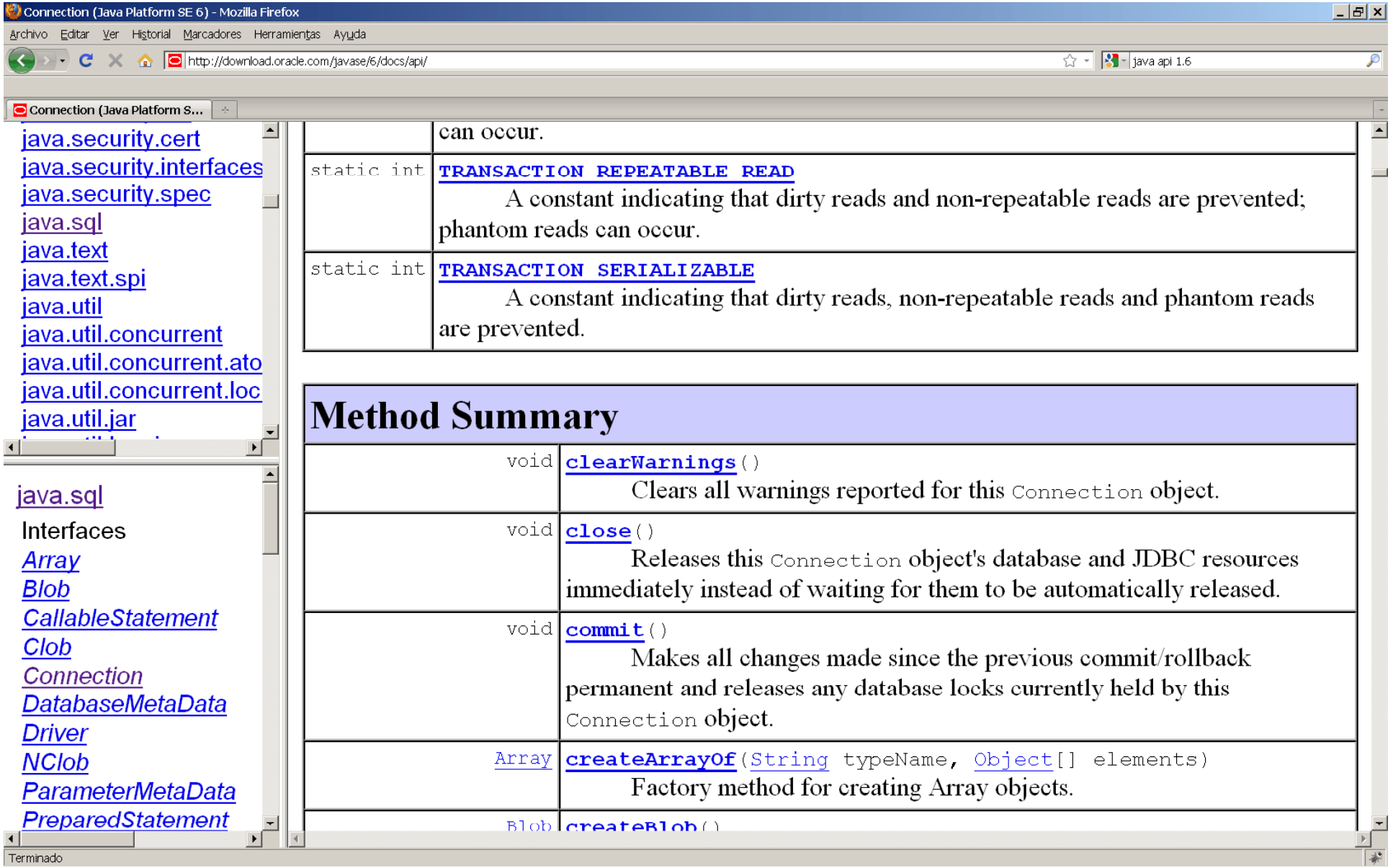
```
C:>java -cp \examples;\lib\Banners.jar \
  greetings.Hi
```

No se especifica el .class

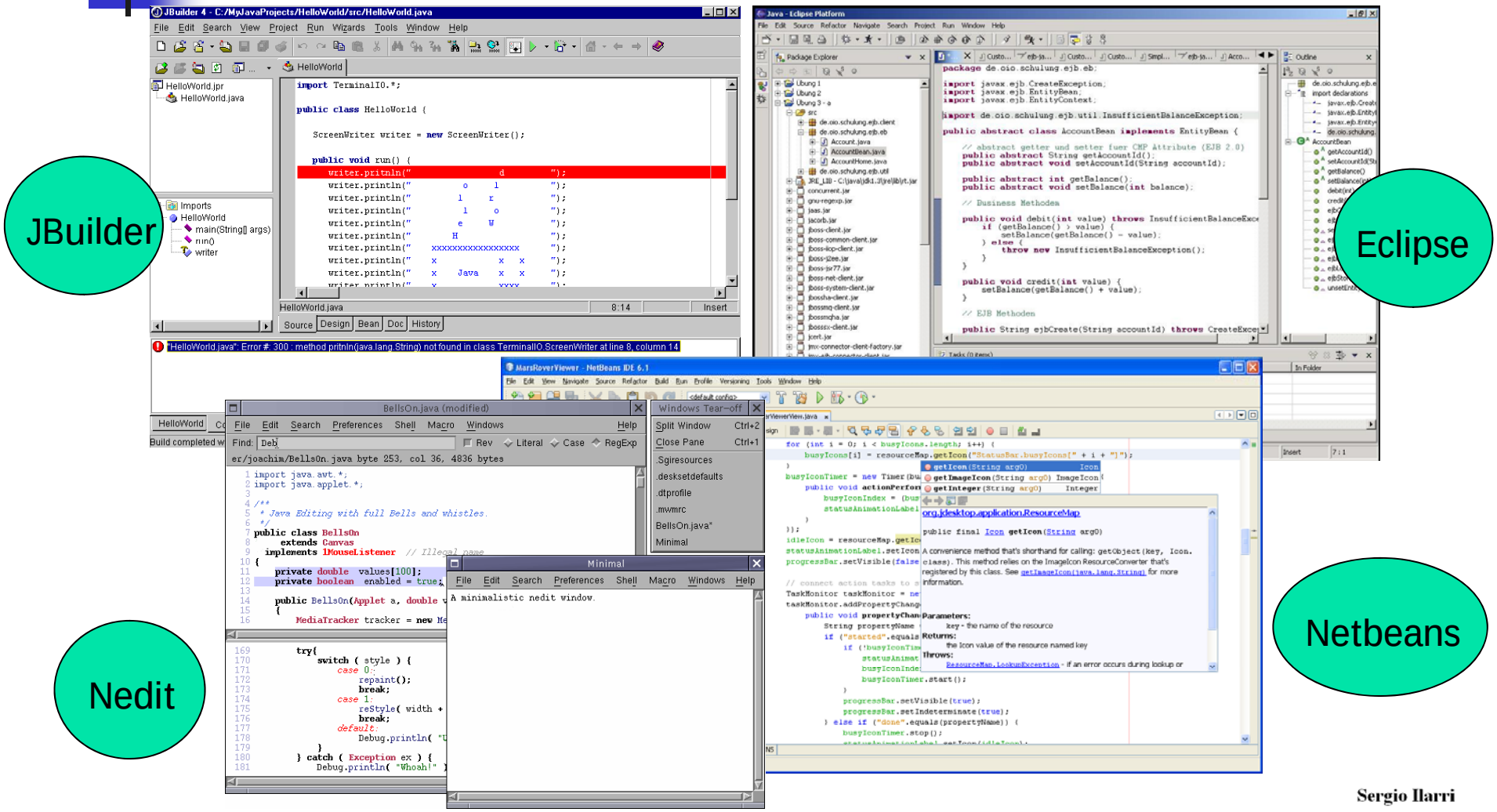

Overview (Java Platform SE 6) - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://download.oracle.com/javase/6/docs/api/
java api 1.6
Overview (Java Platform SE 6)
Java™ Platform Standard Ed. 6
All Classes
Packages
java.applet
All Classes
AbstractAction
AbstractAnnotationVal
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserI
AbstractDocument
AbstractDocument.Attr
AbstractDocument.Cor
AbstractDocument.Ele
AbstractElementVisitor
AbstractExecutorServic
AbstractInterruptibleCh
AbstractLayoutCache
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES
Java™ Platform Standard Ed. 6
Java™ Platform, Standard Edition 6
API Specification
This document is the API specification for version 6 of the Java™ Platform, Standard Edition.
See:
Description
Packages
java.applet
Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt
Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color
Provides classes for color spaces.
Terminado

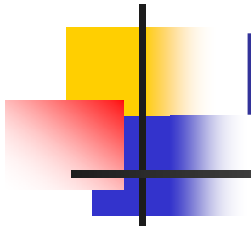

java.sql (Java Platform SE 6) - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://download.oracle.com/javase/6/docs/api/
java api 1.6
java.sql (Java Platform SE 6)
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.text.spi
java.util
java.util.concurrent
java.util.concurrent.ato
java.util.concurrent.loc
java.util.jar
java.sql
Interfaces
Array
Blob
CallableStatement
Clob
Connection
DatabaseMetaData
Driver
NClob
ParameterMetaData
PreparedStatement
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 6
PREV PACKAGE NEXT PACKAGE
FRAMES NO FRAMES
Package java.sql
Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language.
See:
Description
Interface Summary
Array
The mapping in the Java programming language for the SQL type ARRAY.
Blob
The representation (mapping) in the Java™ programming language of an SQL BLOB value.
CallableStatement
The interface used to execute SQL stored procedures.
Clob
The mapping in the Java™ programming language for the SQL CLOB type.
Connection
A connection (session) with a specific database.
Terminado


Connection (Java Platform SE 6) - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://download.oracle.com/javase/6/docs/api/
java api 1.6
Connection (Java Platform S...
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.text.spi
java.util
java.util.concurrent
java.util.concurrent.ato
java.util.concurrent.loc
java.util.jar
java.sql
Interfaces
Array
Blob
CallableStatement
Clob
Connection
DatabaseMetaData
Driver
NClob
ParameterMetaData
PreparedStatement
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 6
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
java.sql
Interface Connection
All Superinterfaces:
Wrapper
public interface Connection
extends Wrapper
A connection (session) with a specific database. SQL statements are executed and results are returned within the context of a connection.
A Connection object's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the getMetaData method.
Note: When configuring a Connection, JDBC applications should use the appropritate Connection
Terminado


Connection (Java Platform SE 6) - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://download.oracle.com/javase/6/docs/api/
java api 1.6
Connection (Java Platform S...
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.text.spi
java.util
java.util.concurrent
java.util.concurrent.ato
java.util.concurrent.loc
java.util.jar
java.sql
Interfaces
Array
Blob
CallableStatement
Clob
Connection
DatabaseMetaData
Driver
NClob
ParameterMetaData
PreparedStatement
can occur.
static int
TRANSACTION_REPEATABLE_READ
A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
static int
TRANSACTION_SERIALIZABLE
A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented.
Method Summary
void
clearWarnings()
Clears all warnings reported for this Connection object.
void
close()
Releases this Connection object's database and JDBC resources immediately instead of waiting for them to be automatically released.
void
commit()
Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this Connection object.
Array
createArrayOf(String typeName, Object[] elements)
Factory method for creating Array objects.
Blob
createBlob()
Terminado

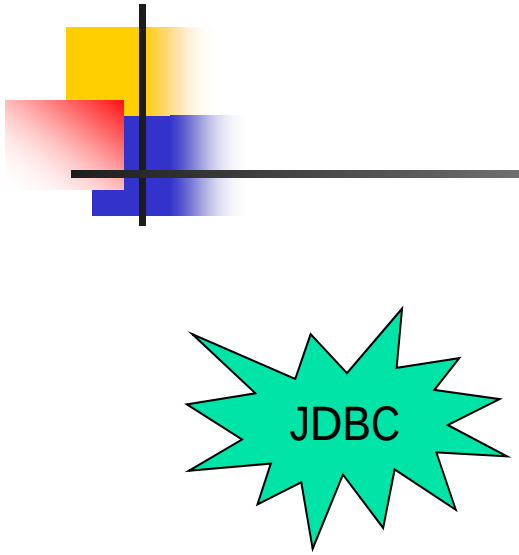
Cómo Editar Java





Propuesta

- Abrid la URL http://java.sun.com/
- Instalad el JDK 1.6
- Echad un vistazo a:
  - Directorios
  - Documentación
- Familiarizaros con el API
- Implementad el *HelloWorld*
- Compilad y ejecutad
- Probad algún otro ejemplo sencillo en texto

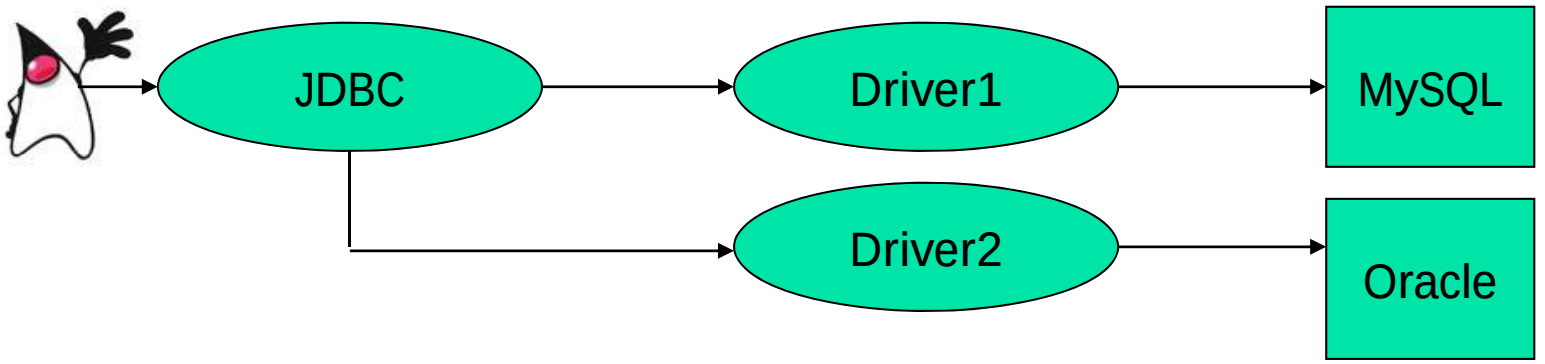


Acceso a Bases de Datos

Sergio Ilarri

JDBC

- *Java Database Connectivity*
- Parecido a ODBC pero en Java (sencillo)



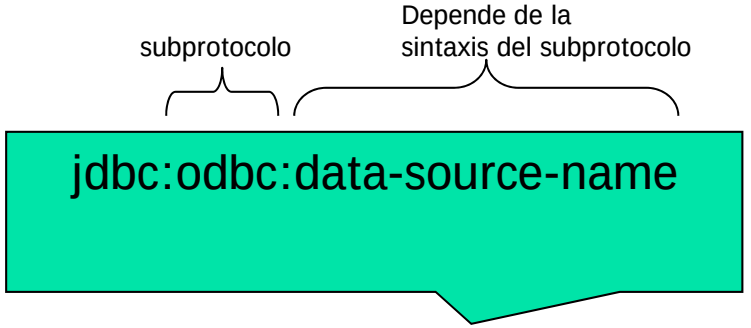
Conexión (I)

1) Cargar el driver JDBC-ODBC:

- `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
- `java -Djdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver AProgram`

Se registra en DriverManager

2) Conectarse a la fuente

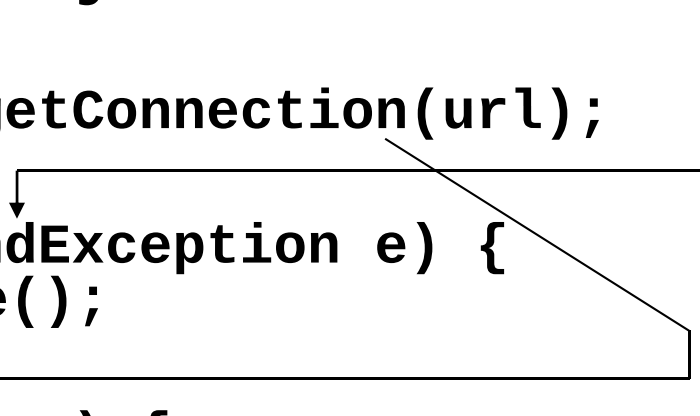



```
Connection con = DriverManager.getConnection(URL,
  username, password);
```

Selecciona el driver apropiado

Conexión (II)

```
String url = “...";
try {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection con =
     DriverManager.getConnection(url);
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
catch (SQLException e) {
   e.printStackTrace();
}
```



Conexión (III): Clase *Connection*

- Para interaccionar con la BD:
  - Ejecución de consultas
  - Obtención de resultados
- Métodos:
  - **`Statement createStatement()`**
  - **`PreparedStatement prepareStatement(String sql)`**
  - **`commit, rollback, setAutoCommit(bool)`**
  - **…**

Clase *Statement* (I)

- Para ejecutar consultas estáticas en SQL
- Métodos:
  - `ResultSet executeQuery(String)`
  - `boolean execute(String)`
  - `int executeUpdate(String)`
  - `...`

Devuelve *true* si es posible recuperar resultados (getResultSet)

UPDATE, INSERT, DELETE (devuelve el número de registros actualizados)

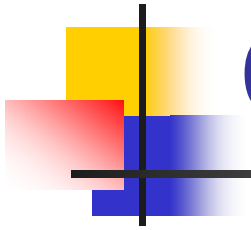
Clase *Statement* (II)

```
Statement stmt = con.createStatement();

stmt.execute("create table JoltData ("+
   "programmer varchar (32), “ +
   "day char (3),“ + "cups integer);");

stmt.execute("insert into JoltData values
   ('Josephine', 'Fri', 4);“);

stmt.close();
```

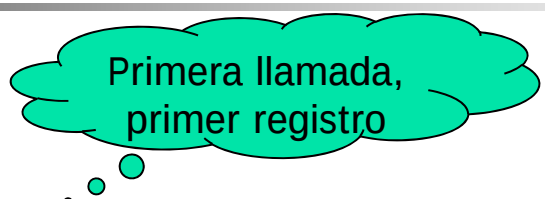


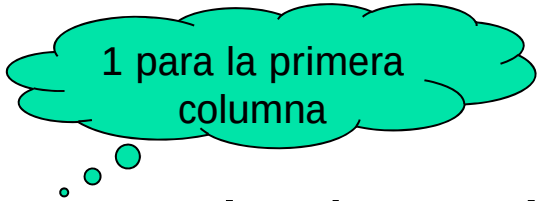
Clase *ResulSet* (I)

- Apunta a la tabla resultado
- Guarda el cursor a la fila actual
- Recuperación registro a registro

Clase *ResulSet* (II)

- Métodos:
  - boolean next()
  - void close()
  - Type getType(int/String columnIndex/name)
  - String getString/getBoolean(int columnIndex)
  - boolean wasNull(int columnIndex)
  - ...



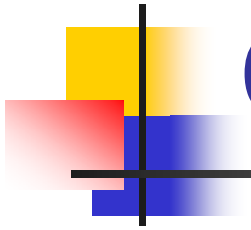


Clase *ResultSet* (III)

```
ResultSet result = stmt.executeQuery(
  "SELECT programmer,
  cups FROM JoltData ORDER BY cups DESC;");

result.next();

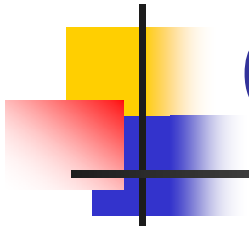
String name = result.getString("programmer");
int cups = result.getInt("cups");
System.out.println("Programmer “ + name +
  " consumed the most coffee: “ + cups +
  " cups.");
```



Clase *ResultSet* (IV)

Recorriendo el ResultSet

```
cups = 0;
while(result.next()) {
   cups += result.getInt("cups");
}
System.out.println("Total sales of“ +
   cups + " cups of coffee.");
```



Clase *PreparedStatement* (I)

- Extiende *Statement*
- Útil para hacer varias veces la misma operación de forma más eficiente
- La precompila el motor de la fuente de datos (si el SGBD lo soporta)
- También es útil cuando se tienen muchos argumentos para completar un comando SQL particular

Clase *PreparedStatement* (II)

1

```
PreparedStatement prep = con.prepareStatement(
    "INSERT into Data values (?, ?)");
```

Rellenar con setXXX(...)

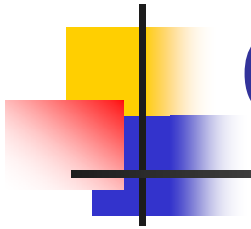
2

```
prep.setString(1, "Jim");
prep.setInt(2, 70);
```

Clase *PreparedStatement* (III)

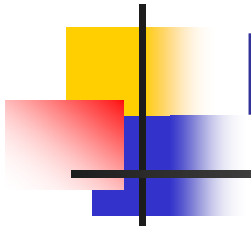
3

```
if (prep.executeUpdate() != 1) {
    throw new Exception ("Bad Update");
}
```



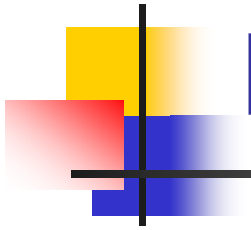
Clase *CallableStatement*

- Llamadas a procedimientos almacenados
- Similar a trabajar con PreparedStatement:
  - setXXX: parámetros de entrada
  - registerOutParameter: parámetros de salida
- Connection.prepareCall(...)



Metadatos (I)

- Connection:
  - DatabaseMetaData getMetaData()
- ResultSet:
  - ResultSetMetaData getMetaData()



Metadatos (II)

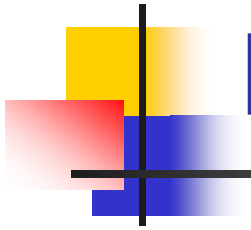
DatabaseMetaData

```
if (md==null) {
  System.out.println("No Database Meta Data");
}
else {
  System.out.println("Database Product Name: " +
    md.getDatabaseProductName());
  System.out.println(“Max. active connections: “ +
    md.getMaxConnections());
}
```

Metadatos (III)

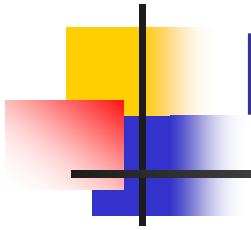
ResultSetMetaData

```
int numbers = 0;
for (int i=1;i<=columns;i++) {
  System.out.println(meta.getColumnLabel(i) +
     "\t" + meta.getColumnTypeName(i));
  if (meta.isSigned(i)) {
     numbers++;
  }
}
System.out.println ("Columns: " +
  columns + " Numeric: " + numbers);
```



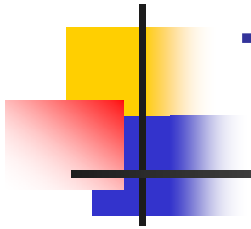
Excepciones

- SQLException
- SQLWarning (no crítico, no se lanza):
  - Se pueden consultar con *getWarnings()* en Connection, ResultSet, y Statement
  - Tipo especial de SQLWarning: DataTruncation



Recordad...

- Llamar a close() para cerrar:
  - Connection
  - Statement
  - ResultSet



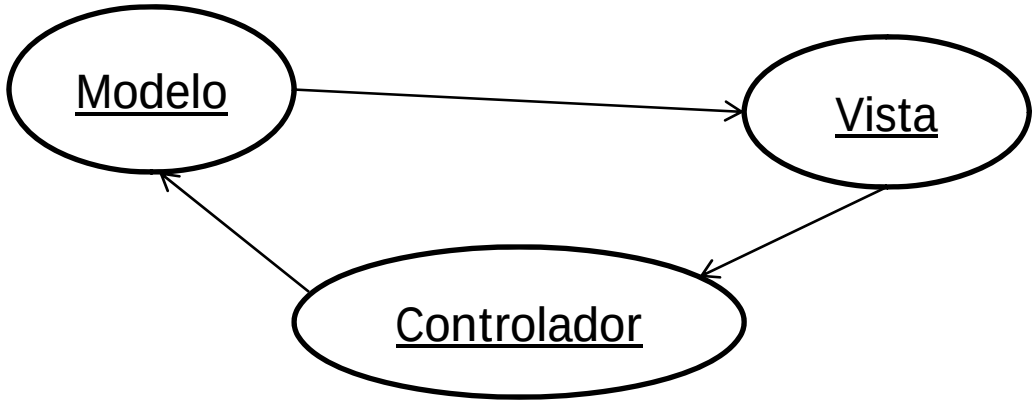
Tipos Java y SQL

Java.sql.Types	SQL Types
BIGINT	BIGINT
BINARY	CHAR FOR BIT DATA
BIT1	CHAR FOR BIT DATA
BLOB	BLOB (JDBC 2.0 and up)
CHAR	CHAR
CLOB	CLOB (JDBC 2.0 and up)
DATE	DATE
DECIMAL	DECIMAL
DOUBLE	DOUBLE PRECISION
FLOAT	DOUBLE PRECISION2
INTEGER	INTEGER
LONGVARBINARY	LONG VARCHAR FOR BIT DATA
LONGVARCHAR	LONG VARCHAR
NULL	Not a data type; always a value of a particular type
NUMERIC	DECIMAL
REAL	REAL
SMALLINT	SMALLINT
TIME	TIME
TIMESTAMP	TIMESTAMP
VARBINARY	VARCHAR FOR BIT DATA
VARCHAR	VARCHAR

Swing

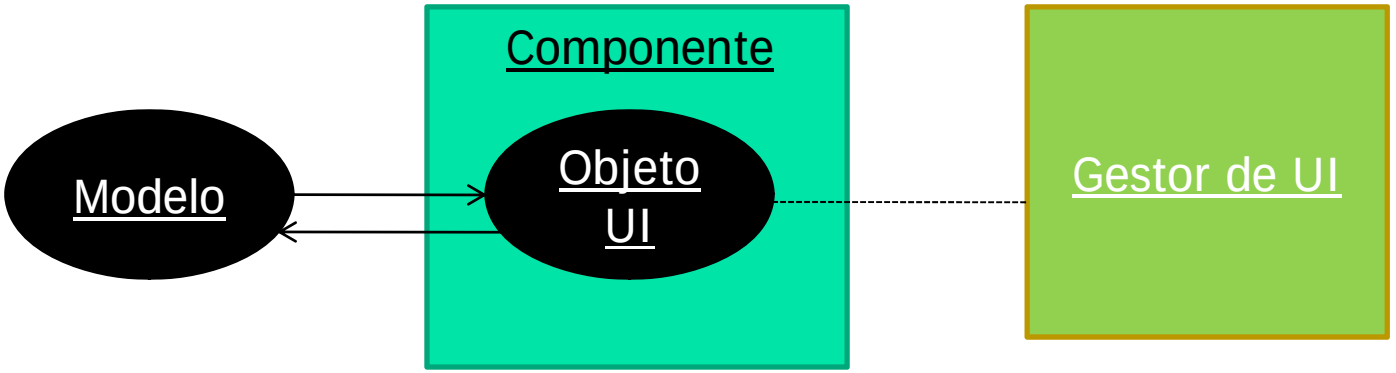
Arquitectura MVC

- *Model-View-Controller*
- División de una aplicación visual en 3 partes:
  - Modelo: representa los datos de la aplicación
  - Vista: es la representación visual de los datos
  - Controlador: recibe entradas del usuario a través de la vista y actualiza el modelo en consecuencia



Arquitectura MVC

- El primer prototipo de Swing seguía ese estilo
- Pero luego se determinó que la vista y el controlador requerían un fuerte acoplamiento
- *Separable model architecture* (*quasi-MVC*)
  - *UI* (User Interface) *Object* = *UI delegate* = *delegate object*
    - Vista + controlador



http://java.sun.com/products/jfc/tsc/articles/architecture/

Ejemplo: HelloWorld

```
import javax.swing.*;
public class HelloWorldSwing {
 private static void createAndShowGUI() {
  JFrame.setDefaultLookAndFeelDecorated(true);
  JFrame frame = new JFrame("HelloWorldSwing");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  JLabel label = new JLabel("Hello World");
  frame.getContentPane().add(label);
  frame.pack();
  frame.setVisible(true);
 }
 public static void main(String[] args) {
  javax.swing.SwingUtilities.invokeLater(new Runnable() {
   public void run() {createAndShowGUI();}
  });
 }
}
```

Top-level container (otros: *JDialog*, *JApplet*)

Se añade al *content pane*

Evita problemas (*thread-safe*)

Ejemplo: SwingApplication (I)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SwingApplication implements ActionListener {

  private static String labelPrefix =
    "Number of button clicks: ";
  private int numClicks = 0;
  final JLabel label = new JLabel(labelPrefix + "0    ");

  ...
}
```

Implementa manejadores de eventos *action* , luego puede registrarse como *listener* de un objeto gráfico

Ejemplo: SwingApplication (II)

```
public Component createComponents() {
  JButton button = new JButton("I'm a Swing button!");
  button.setMnemonic(KeyEvent.VK_I);
  button.addActionListener(this);
  label.setLabelFor(button);
  JPanel pane = new JPanel(new GridLayout(0, 1));
  pane.add(button);
  pane.add(label);
  pane.setBorder(BorderFactory.createEmptyBorder(30, 30, 10, 30);
  return pane;
}
```

Este objeto va a escuchar eventos *action* del botón

Píxels: top, left, right, bottom

JPanel : contenedor para agrupar componentes
GridLayout : *layout manager*
EmptyBorder : crear "hueco"/separación

Ejemplo: SwingApplication (III)

```
public void actionPerformed(ActionEvent e) {
  numClicks++;
  label.setText(labelPrefix + numClicks);
}
```

Implementa el *interface* ActionListener

Cuando se haga *click* en el botón

Hay un único *event-dispatching thread*
(para el tratamiento de eventos y repintado).
Por tanto, ¡hay que ser rápidos!

Ejemplo: SwingApplication (IV)

```
private static void createAndShowGUI() {
  JFrame.setDefaultLookAndFeelDecorated(true);
  JFrame frame = new JFrame("SwingApplication");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  SwingApplication app = new SwingApplication();
  Component contents = app.createComponents();
  frame.getContentPane().add(contents, BorderLayout.CENTER);
  frame.pack();
  frame.setVisible(true);
}

public static void main(String[] args) {
  javax.swing.SwingUtilities.invokeLater(new Runnable() {
    public void run() {createAndShowGUI();}
  });
 }
```

Algunos Manejadores de Eventos

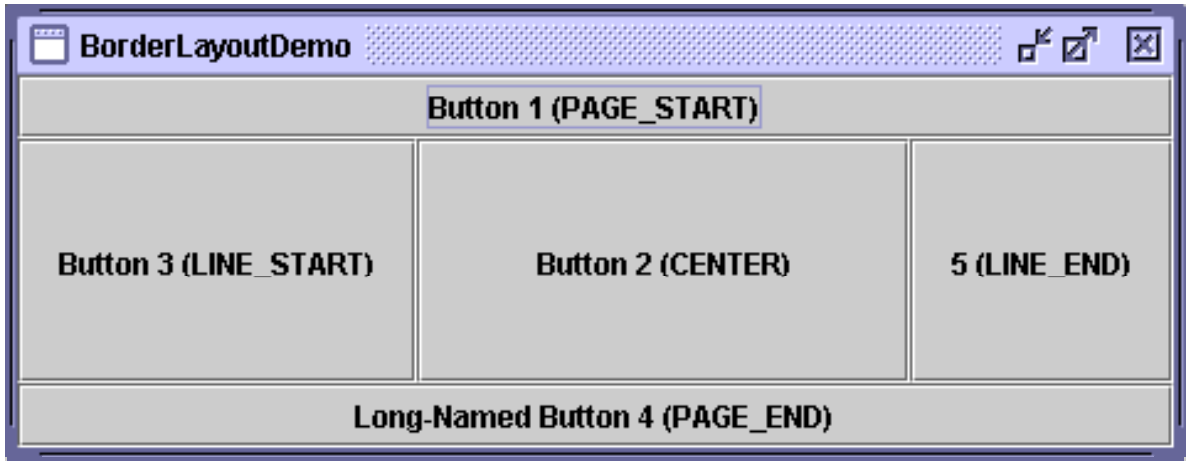
Acción que desencadena el evento	Manejador
El usuario hace click en un botón, presiona *Enter* al escribir texto en un campo, o selecciona una opción de menú	*ActionListener*
El usuario cierra una ventana	*WindowListener*
El usuario presiona un botón del ratón cuando el cursor está sobre un componente	*MouseListener*
El usuario mueve el ratón sobre un componente	*MouseMotionListener*
Un componente se hace visible	*ComponentListener*
Un componente obtiene el foco del teclado	*FocusListener*
Cambia la opción seleccionada en una lista	*ListSelectionListener*
Cambia cualquier propiedad de un componente (ej.: el texto de una etiqueta)	*PropertyChangeListener*

Hay muchos ejemplos de distintos tipos de manejadores de eventos en:

http://java.sun.com/docs/books/tutorial/uiswing/events/intro.html

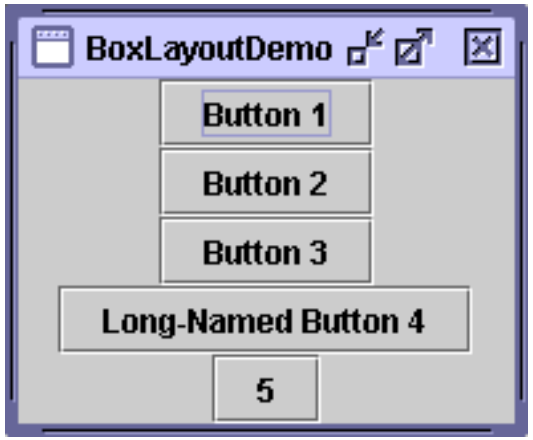
Layout Managers (I)

java.awt
javax.swing

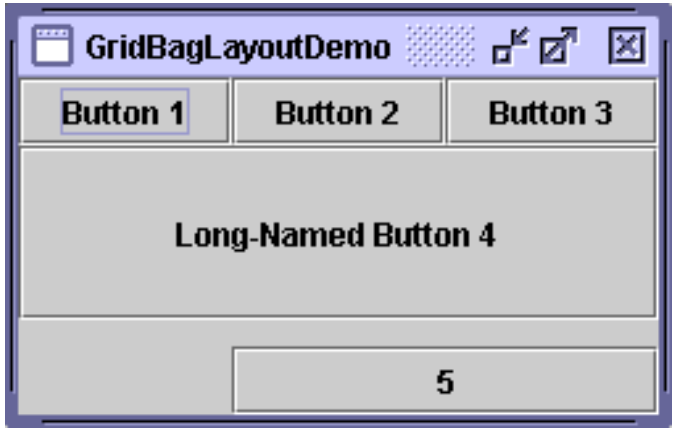


5 áreas: *top*, *bottom*, *right*, *left*, *center*
El área *center* acapara todo el espacio sobrante

Por defecto en los *content pane*
(diálogos, *frames* y *applets*)



1 única fila
o columna



-Sitúa en una rejilla

-Las filas y columnas pueden tener diferentes alturas y anchuras

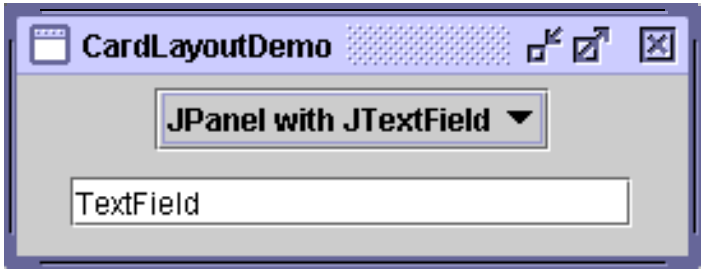
-Un componente puede ocupar varias celdas

Layout Managers (II)

1 única fila

Si se llena, empieza otra

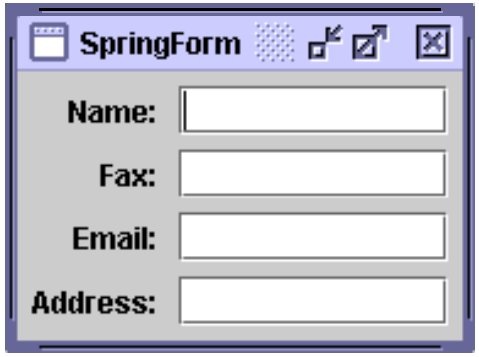
Por defecto en un *JPanel*



Contiene componentes distintos dependiendo de la selección

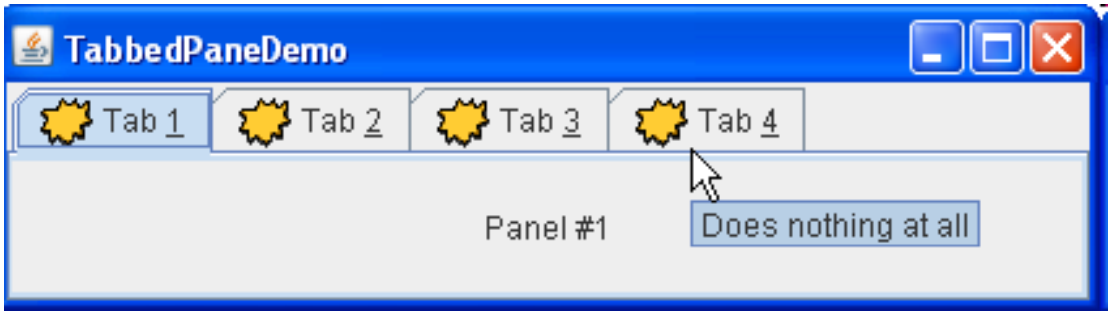
Layout Managers (III)

Componentes de igual tamaño
Rejilla de filas y columnas



Permite especificar relaciones (distancias)
entre componentes

Layout Managers (VI)



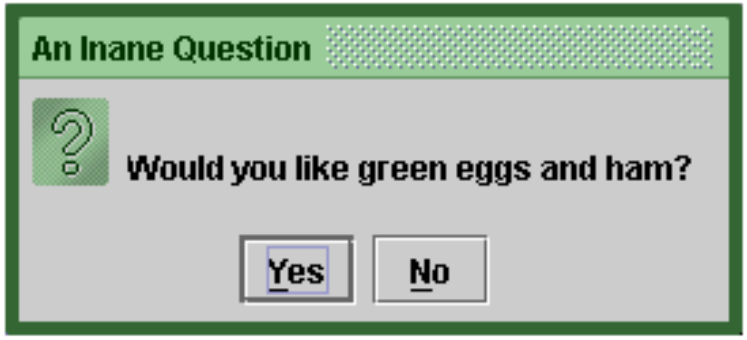
No es recomendable utilizar posicionamiento absoluto

Componentes (I)

Guía visual e interactiva de los componentes de Swing:
http://download.oracle.com/javase/tutorial/ui/features/components.html
Los componentes Jxxx, menos los contenedores de nivel superior, heredan de:
javax.swing.JComponent

1) Contenedores de nivel superior

Applet

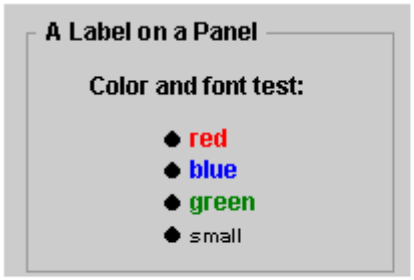


Dialog

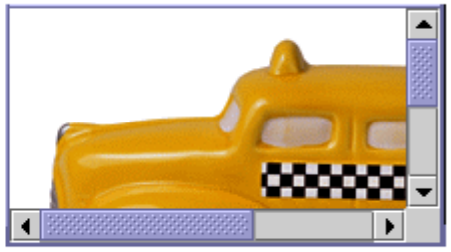
Frame

Componentes (II)

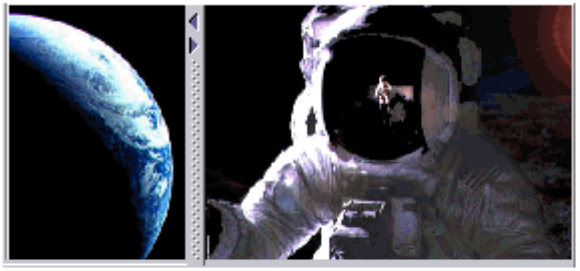
2) Contenedores de propósito general



Panel



Scroll pane



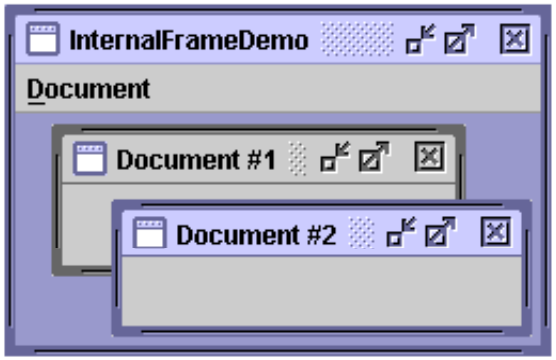
Split pane

Tabbed pane

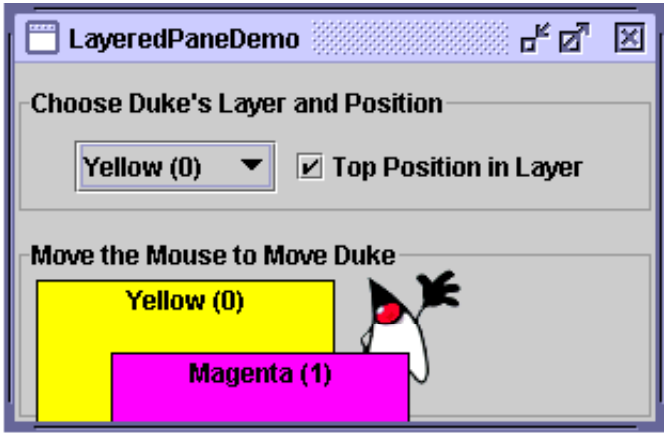
Tool bar

Componentes (III)

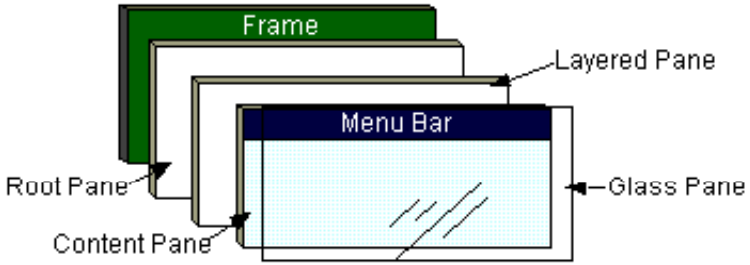
3) Contenedores de propósito especial



Internal frame



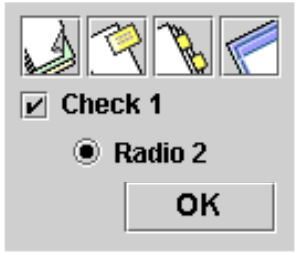
Layered pane



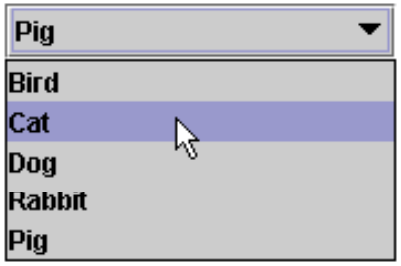
Root pane

Componentes (IV)

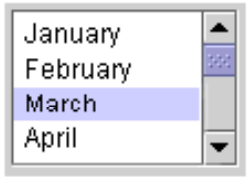
4) Controles básicos



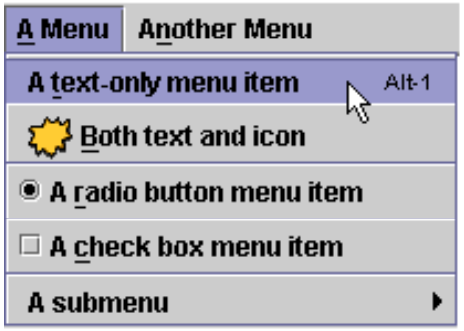
Buttons



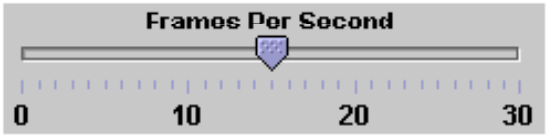
Combo box



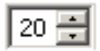
List



Menu



Slider

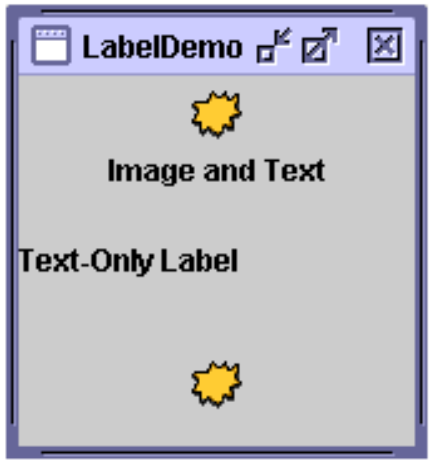


Spinner

Text field or Formatted text field

Componentes (V)

5) Elementos de información no editables



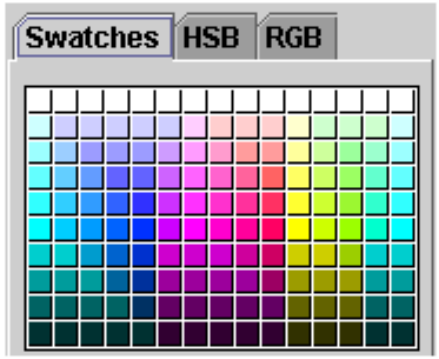
Label

Progress bar

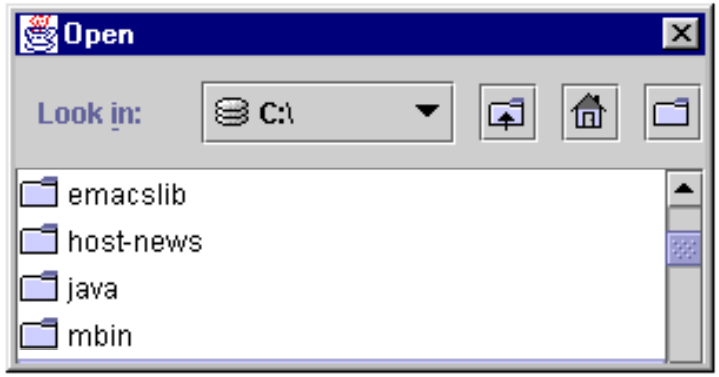
Tool tip

Componentes (VI)

6) Elementos de información interactivos



Color chooser

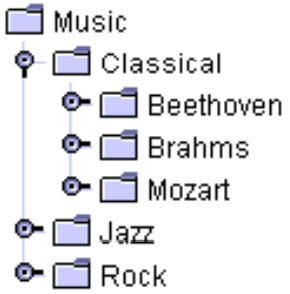


File chooser

First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

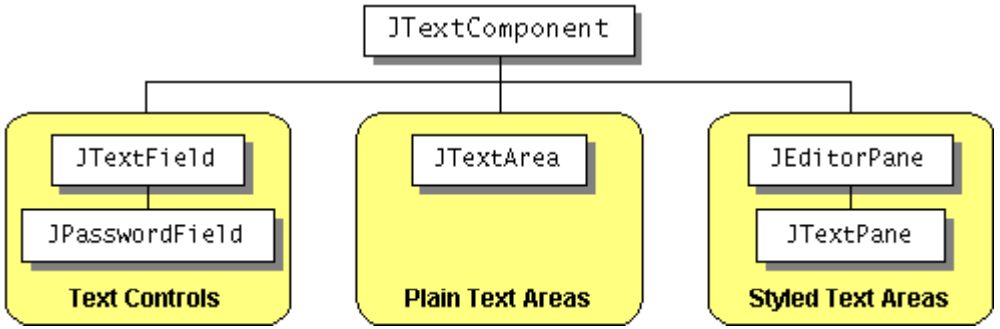
Table

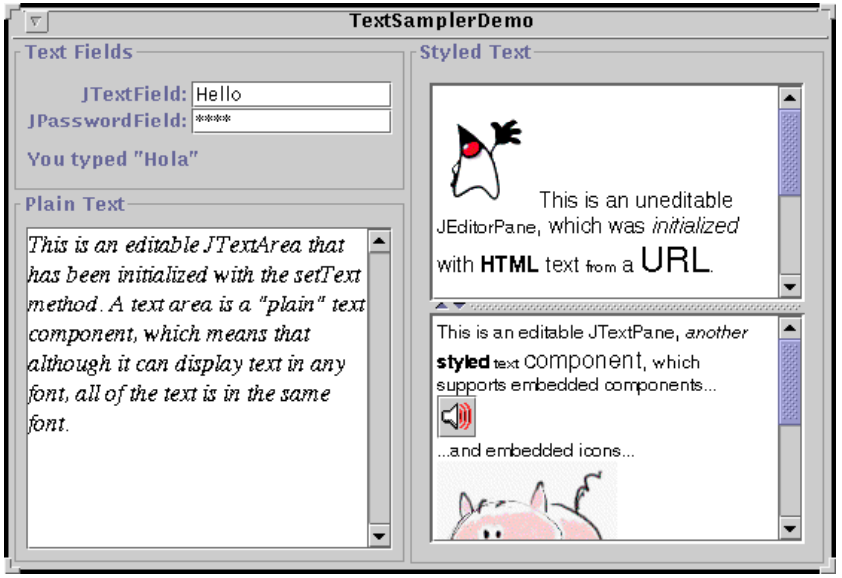
Text



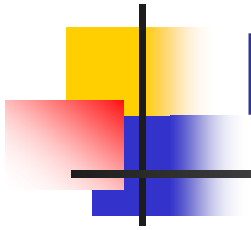
Tree

Componentes (VII): Texto



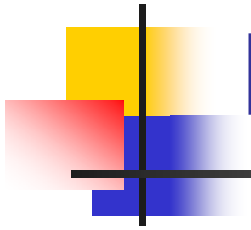


Componentes (VIII): Nombres (I)



- Botones (javax.swing.JButton)
- Casillas de verificación (javax.swing.JCheckBox)
- Campos de texto de una línea (javax.swing.JTextField)
- Campos de texto y edición de varias líneas (javax.swing.JTextArea)
- Etiquetas (javax.swing.JLabel)
- Listas (javax.swing.JList)
- Menús contextuales (javax.swing.Popup)

Componentes (IX): Nombres (II)

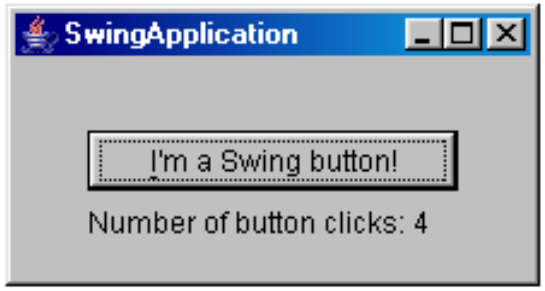


- Barras de desplazamiento (javax.swing.JScrollBar)
- Sliders (javax.swing.JSlider)
- Áreas de dibujo (java.awt.Canvas)
- Menus (javax.swing.JMenu,javax.swing.JMenuBar javax.swing.JMenuItem, javax.swing.JCheckBoxMenuItem)
- Contenedores (javax.swing.JPanel, javax.swing.JWindow and its subclasses)
- ...

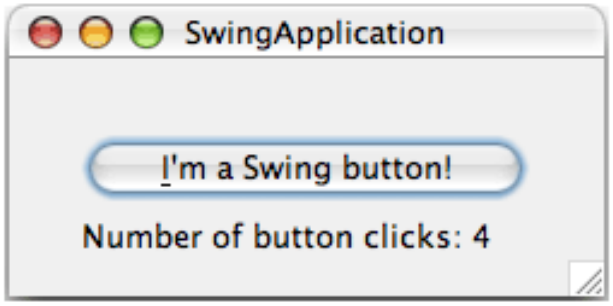
Algunos *Look and Feels*

Java look and feel

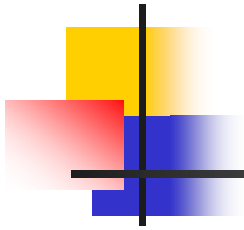
GTK+ look and feel



Windows look and feel



Mac OS look and feel



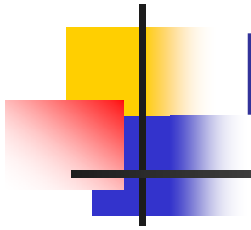
Ejemplo: Dibujar Círculo (I)

```
package is2.silarri;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CircleDrawer extends JFrame {
   ... /* Desarrollado en las dos transparencias siguientes */
   }

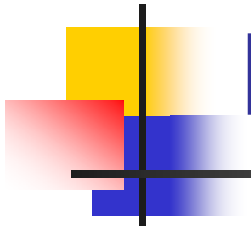
class PanelWithCircle extends JPanel {
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
       g.drawOval( 300, 300, 60, 60 );
   }
}
```



Ejemplo: Dibujar Círculo (II)

```
public CircleDrawer() {
        super("Circle example for IS2 - Sergio Ilarri, " +
                        "October 8, 2006");
        PanelWithCircle pwc = new PanelWithCircle();
        Container container = getContentPane();
        container.add(pwc);

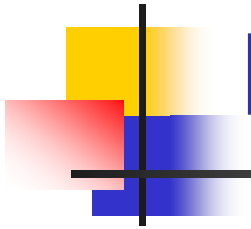
        setSize(600, 600);
        setVisible(true);
   }
```



Ejemplo: Dibujar Círculo (III)

```
public static void main(String[] args) {
        CircleDrawer cd = new CircleDrawer();
        cd.addWindowListener (new WindowAdapter() {
                public void windowClosing ( WindowEvent e )
                {
                        System.exit(0);
                }
        });
   }
```

REFERENCIAS



Referencias (I)

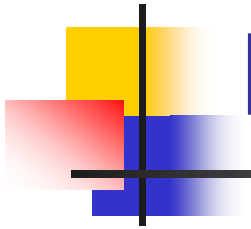
- Java Technology. http://java.sun.com/
- The Java Tutorials. http://java.sun.com/docs/books/tutorial/index.html
- Learning the Java Language. http://java.sun.com/docs/books/tutorial/java/index.html
- Thinking in Java, Bruce Eckel, Prentice Hall, 4th edition, ISBN 0131002872.
- Thinking in Java, 3th edition. Free download: http://mindview.net/Books/TIJ/DownloadSites
- Exploring Java, Patrick Niemeyer, Josh Peck, ISBN 1-56592-184-4, 426 pages.

Referencias (II)

- Java SE - Java Database Connectivity (JDBC).
  https://java.sun.com/javase/technologies/database.jsp

  http://java.sun.com/developer/onlineTraining/Database/JDBCShortCourse/index.html

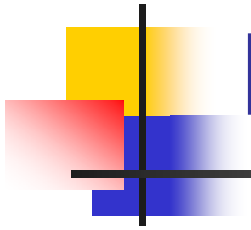
- The Java History Timeline.
  http://www.java.com/en/javahistory/timeline.jsp
- Java – wikipedia.
  http://en.wikipedia.org/wiki/Java_programming_language
- Does Java pass by reference or pass by value?
  http://www.javaworld.com/javaworld/javaqa/2000-05/03-qa-0526-pass.html



Referencias (III)

- TIOBE Programming Community Index. http://www.tiobe.com/
- Programming Language Trends. O'Reilly Radar. http://radar.oreilly.com/archives/2006/08/programming_language_trends_1.html
- Indeed. Job Trends. http://www.indeed.com/jobtrends

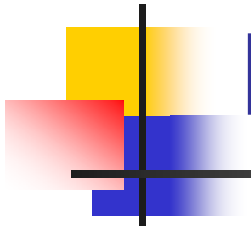
- Eclipse. http://www.eclipse.org/
- Jbuilder. http://www.borland.com/us/products/jbuilder/index.html



Referencias (IV)

- The Swing Tutorial. http://java.sun.com/docs/books/tutorial/ui/index.html

- Swing Second Edition, Matthew Robinson y Pavel Vorobiev, 2003, 912 páginas, Manning Publications Co., ISBN 193011088X. Versión previa gratuita en Word en: http://www.manning.com/robinson2/



Fin

GRACIAS POR
VUESTRA ATENCIÓN

Esto es sólo una guía... ahora es necesario practicar y consultar con el profesor los problemas que surjan.