PROGRAMACIÓN CON JAVA

Índice

- Introducción a Java
- Java básico
- Objetos con Java
- Conceptos avanzados

INTRODUCCIÓN A JAVA

- Historia
- Características de Java
- Java Development Kit
- Herramientas del kit
- Entornos de desarrollo

Bibliografía

- Java Tutorial
  - http://java.sun.com/docs/books/tutorial/index.html
- Thinking in Java
  - http://www.mindview.net/Books/TIJ/
- Aprenda Java como si estuviera en 1º
  - http://www.entrebits.com/descargas/programas/aprenda-java-como-si-estuviera-en-primero/
- Curso de iniciación a Java
  - http://www.gui.uva.es/~jal/java/index.html

Una breve historia

- Iniciativa de Sun Microsystems en 1991
- Lenguaje para ser empleado en dispositivos pequeños (set top boxes)
  - Poca memoria
  - Poca potencia
  - Independencia de la plataforma

Una breve historia

- Fracaso y oportunidad
- Con el auge de la WWW, crearon el navegador HotJava
  - Características de Java
  - Primero en usar applets
- El éxito vino porque Netscape aceptó los applets

Características

- Simple
- Orientado a objetos
- Distribuido
- Robusto
- Seguro
- Independiente de la plataforma
- Multihilo nativo

Simple

- Sintaxis de C / C++
  - Gran base de programadores
- Se eliminan sus problemas
  - Aritmética de punteros
  - Recolección de basura

Orientado a objetos

- Paradigma de programación que basado en modelar el sistema como clases e instancias de las clases
- Evolución del modelo imperativo

Robusto

- Gran cantidad de comprobaciones en tiempo de ejecución
  - Límite de arrays
  - Tamaño de la pila
  - Acceso a variables
  - Tipo de objetos

Seguro

- Seguridad a nivel de código
  - Se comprueba que el código es coherente antes de ejecutarse
- Seguridad en ejecución
  - Gestores de seguridad limitan acceso a recursos

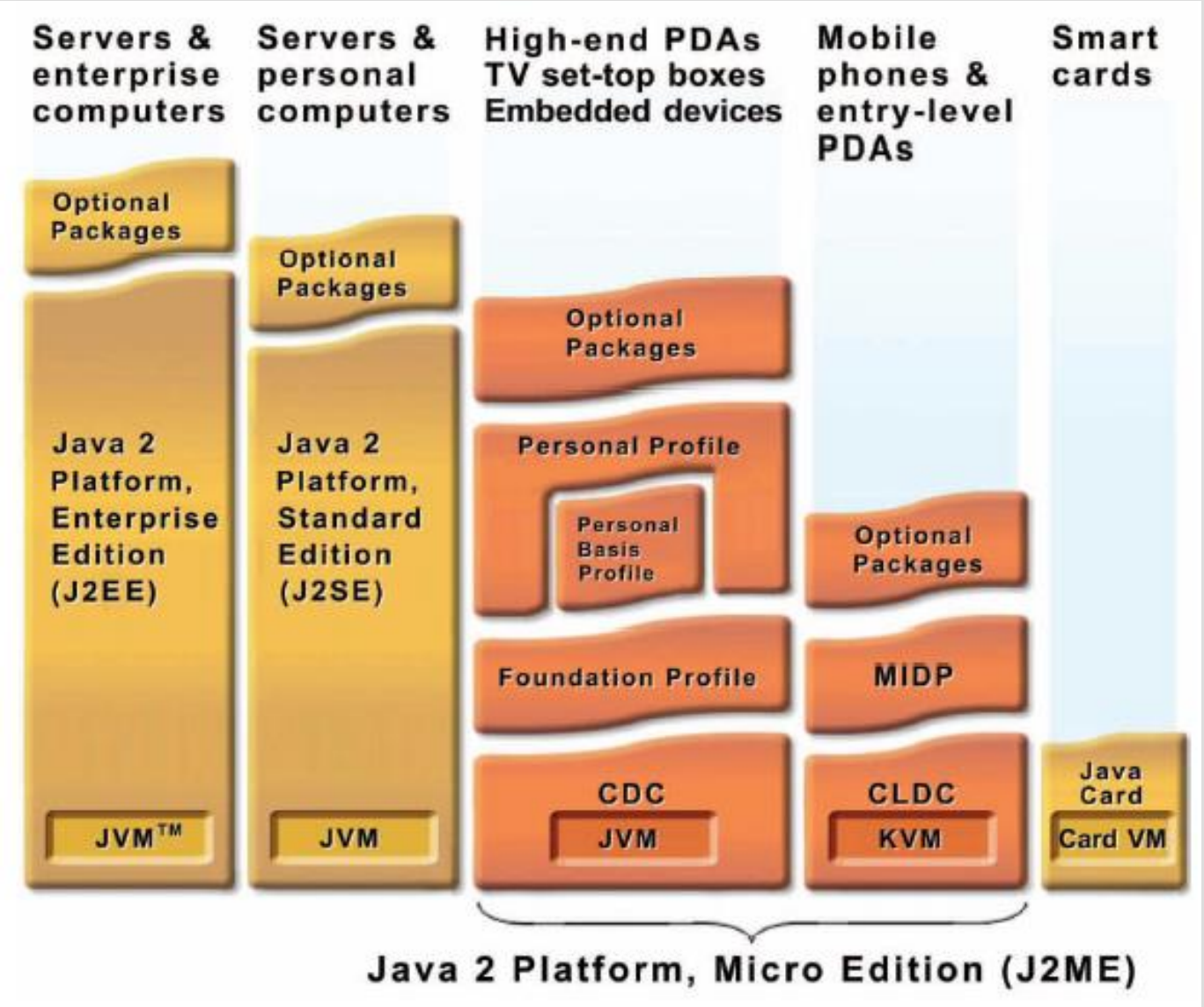
Independiente de la plataforma

- Java sigue la política de WORE
  - Escribe (compila) el código una vez y podrás ejecutarlo donde sea
  - Multiplataforma mediante el uso de un código intermedio (bytecodes)
- Java Virtual Machine
  - Interpreta el código bytecode

Multihilo nativo

- Java soporta hilos de forma nativa la máquina virtual
- Mapeados sobre el sistema operativo (native) o simulados (green)

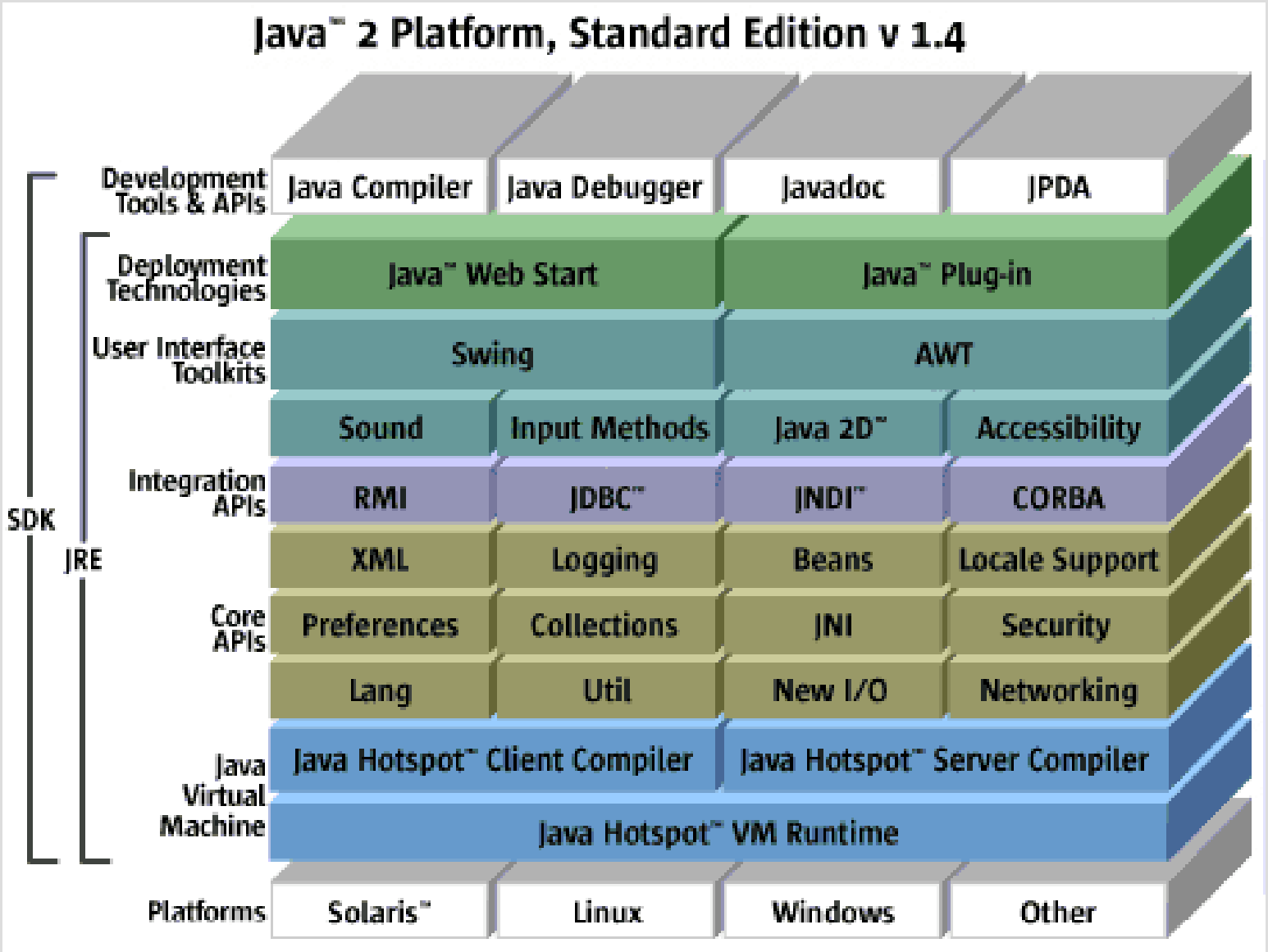
Ediciones JAVA



Java Development Kit

- J2SE (Java 2 Standard Edition)
- Nivel intermedio de programación
- Versión 1.4
  - Java 2 a partir de la versión 1.2
  - La última versión es la 1.5

Java Development Kit



Java Development Kit

- No incluye entorno de edición
- Varios:
  - NetBeans / SunOne
  - Eclipse
  - JDeveloper / JBuilder
  - IBM VisualAge

Java Development Kit

- Diferencias
- Java 2 SDK
  - Incluye lo necesario para el desarrollo de aplicaciones
- Java 2 JRE
  - Incluye sólo lo necesario para la ejecución de aplicaciones

El CLASSPATH

- Dónde se encuentran las clases para ejecutar la aplicación
- Dos maneras:
  - Variable de entorno CLASSPATH
  - Argumento (-cp o -classpath)
- Por defecto, las clases del JDK (JRE)

Compilar con el JDK

- La aplicación es **javac**
- javac <fichero.java>
  - -d <destino>
  - -classpath <jar_o_directorio>

Ejecutar con el JDK

- La aplicación es **java**
- java <clase>
  - -cp <jar_o_directorio>
- java -jar <fichero.jar>
  - -cp <jar_o_directorio>

Ejecutar con el JDK

- La aplicación es **appletviewer**
- appletviewer <fichero.html>
- O abrir el fichero HTML con un navegador

Ejemplos de Applets

- Duke bailando
  - http://java.sun.com/docs/books/tutorial/getStarted/index.html
- Pintando en pantalla
  - http://java.sun.com/docs/books/tutorial/java/concepts/practical.html
- Applets de ejemplo sobre la máquina virtual Java
  - http://pl.changwon.ac.kr/~pl/seminar/jvm/insidejvm/applets/

Ficheros JAR

- Java Archive
- Son ficheros ZIP que contienen clases, fuentes, etc.
- Tienen un directorio META-INF dónde reside el manifiesto (MANIFEST.MF)
  - Describe el contenido del JAR

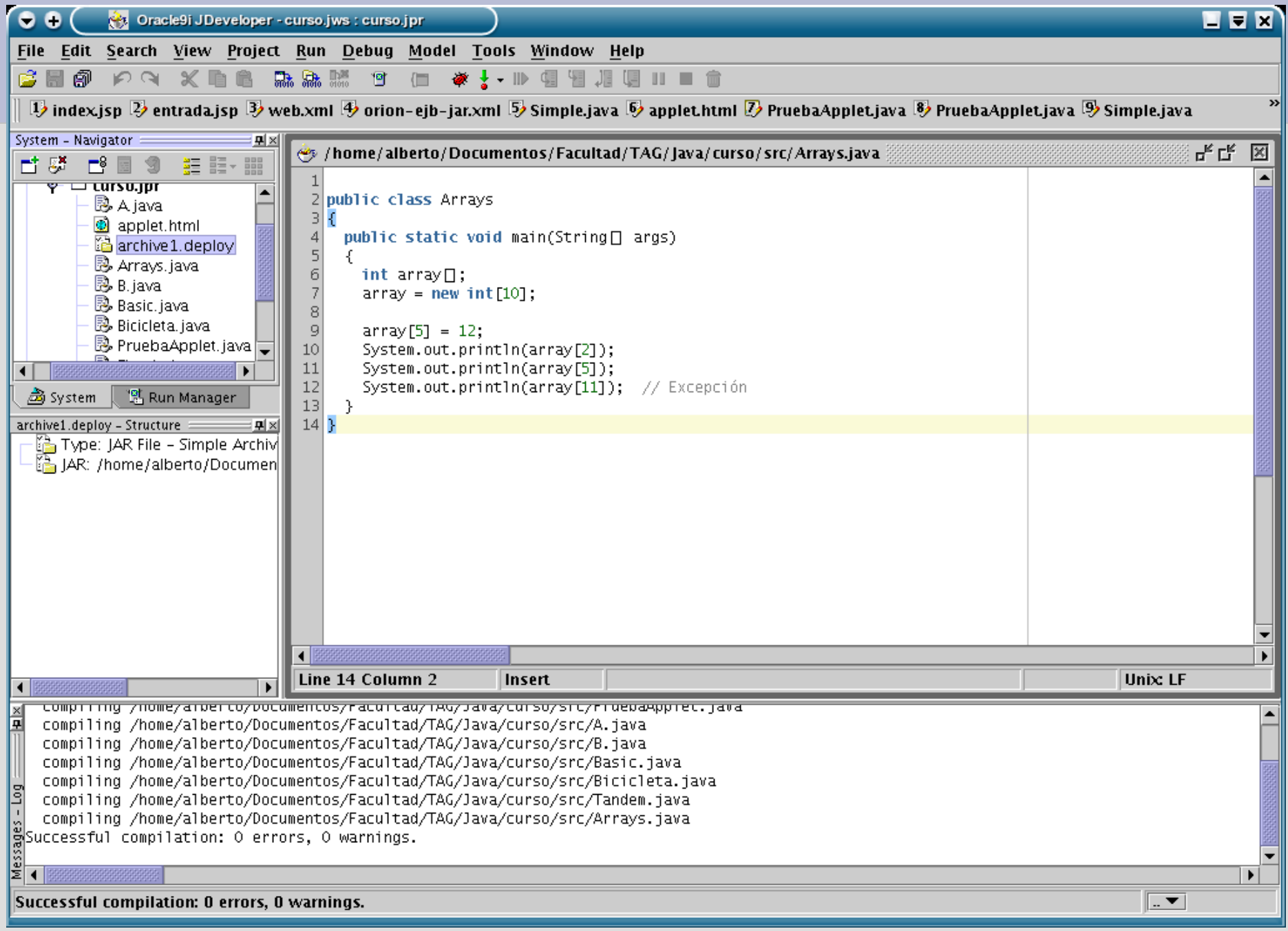
La importancia del API

- Un lenguaje muere si no tiene un buen API (Interfaz para la programación de aplicaciones)
- Java tiene mucha funcionalidad nativa
  - http://java.sun.com/j2se/1.4.2/docs/api/index.html
- APIs para otras funcionalidades

JDeveloper

- Desarrollado por Oracle
  - Empezó como fork del JBuilder
- Gratuito
- Desarrollo de aplicaciones
- Depurador
- Diseño de ventanas

JDeveloper

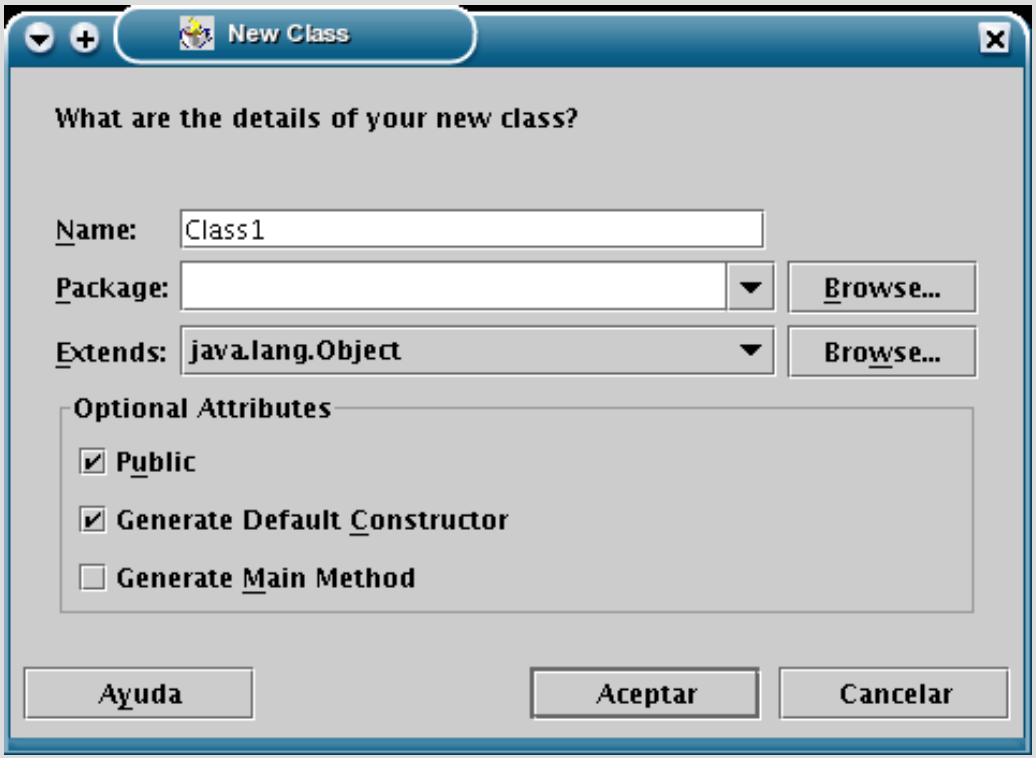

Oracle9i JDeveloper - curso.jws : curso.jpr
File Edit Search View Project Run Debug Model Tools Window Help
index.jsp entrada.jsp web.xml orion-ejb-jar.xml Simple.java applet.html PruebaApplet.java PruebaApplet.java Simple.java
System - Navigator
curso.jpr
A.java
applet.html
archive1.deploy
Arrays.java
B.java
Basic.java
Bicicleta.java
PruebaApplet.java
System
Run Manager
archive1.deploy - Structure
Type: JAR File - Simple Archiv
JAR: /home/alberto/Documen
/home/alberto/Documentos/Facultad/TAG/Java/curso/src/Arrays.java
1
2 public class Arrays
3 {
4 public static void main(String[] args)
5 {
6 int array[];
7 array = new int[10];
8
9 array[5] = 12;
10 System.out.println(array[2]);
11 System.out.println(array[5]);
12 System.out.println(array[11]); // Excepción
13 }
14 }
Line 14 Column 2
Insert
Unix LF
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/PruebaApplet.java
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/A.java
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/B.java
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/Basic.java
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/Bicicleta.java
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/Tandem.java
compiling /home/alberto/Documentos/Facultad/TAG/Java/curso/src/Arrays.java
Successful compilation: 0 errors, 0 warnings.
Messages - Log
Successful compilation: 0 errors, 0 warnings.

JDeveloper

- Ejecutar JDeveloper
- Abrir el fichero curso.jws
  - File -> Open
- Ejemplos del curso
- Desarrollar nuevos ejemplos

JDeveloper

- File -> New
- General -> Simple Files -> Java Class



JAVA BÁSICO

- Ficheros Java
- Variables y tipos de datos
- Ámbito y visibilidad
- Operadores y expresiones
- Sentencias (condicionales y bucles)
- Comienzo y terminación de programas

Código de ejemplo

```
public class Basic {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Ficheros JAVA

- Ficheros de texto
- Los ficheros describen clases
  - Como mucho, una única clase pública
- Los ficheros deben llevar el mismo nombre que la clase pública

Comentarios

- Comentario de una línea: //
  - int usu = 0; // el número de usuarios
- Comentario de bloque: /* .... */
  - /* Esto no se tendrá
    en cuenta */
- Comentario javadoc: /** .... */
  - Para comentar el código

La definición de la clase

- En Java, todo son clases
- Así que cada fichero define una clase
  - public class MiClase
- La clase tiene atributos y métodos
- El método main es especial
  - public static void main(String[] args) {...}
  - Es el punto de arranque de una clase

Variables

- Almacenan el estado de los objetos
- Pueden definirse en cualquier parte del código, antes de su uso
- Deben empezar con letra, guión bajo (_) o dólar ($)
- Sensible a mayúsculas/minúsculas

Variables

- Dí que variables son válidas

  - ´ int
  - √ anInt
  - √ i
  - √ i1
  - ´ 1
  - √ thing1
  - ´ 1thing
  - ´ ONE-HUNDRED
  - √ ONE_HUNDRED
  - √ something2do

Variables

- Las variables tienen un tipo que define el rango de valores y las operaciones que pueden hacerse
  - Java es fuertemente tipado
- Tipos primitivos
- Tipos referencia

Tipos de datos primitivos

- Enteros (se inicializan a 0)
  - Byte: un byte con signo ((byte)12)
  - Short: dos bytes con signo ((short)1231)
  - Int: cuatros bytes con signo (1238921)
  - Long: ocho bytes con signo (728478283L)
- Reales (se inicializan a 0.0)
  - Float: punto flotante 32 bits (1.2342F)
  - Double: punto flotante 64 bits (123.131)

Tipos de datos primitivos

- Booleanos (se inicializa a false)
  - true / false
- Carácter (se inicializa al carácter nulo)
  - 'S', 'a'
- El tamaño de datos está definido y es independiente de la plataforma

Tipos de datos referencia

- Objetos
  - Instancias de clases
- Arrays
  - Colección de elementos del mismo tipo, sean básicos o complejos
- Se inicializan a null

Inicialización

- Las variables pueden inicializarse en la definición

```
int a = 2;
char b = 'a';
double c = 123.13123;
Bicicleta d = new Bicicleta();
```

Constantes

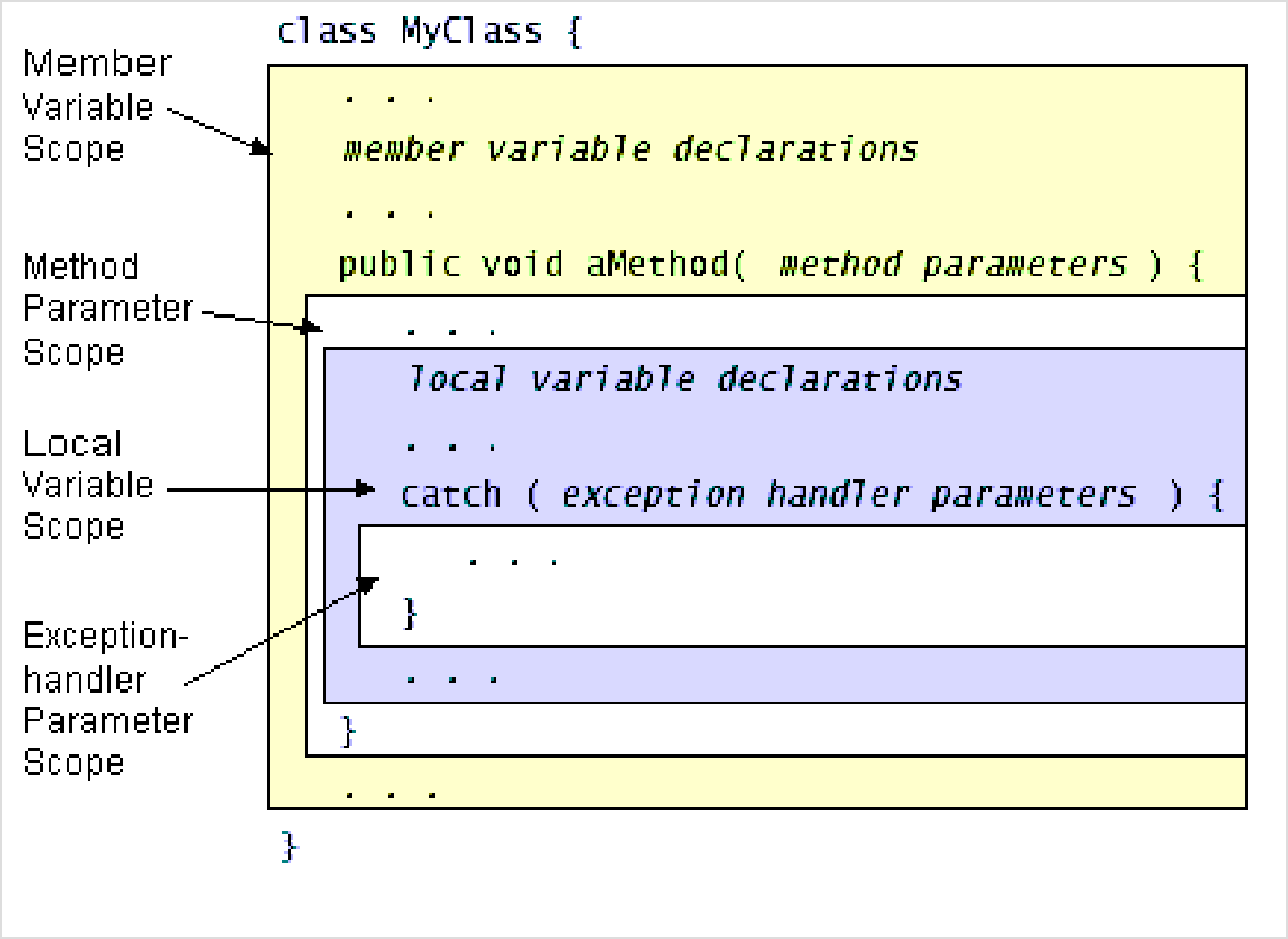
- Variables que no pueden cambiar de valor una vez establecido
- Modificador *final*

```
final int a;
a = 12;
final double b = 456.4546456;
```

Ámbito

- Ámbito: parte de programa en el que una variable está definida
- Variable miembro (de una clase)
- Parámetro (de un método)
- Variable local (de un método)
- Excepciones

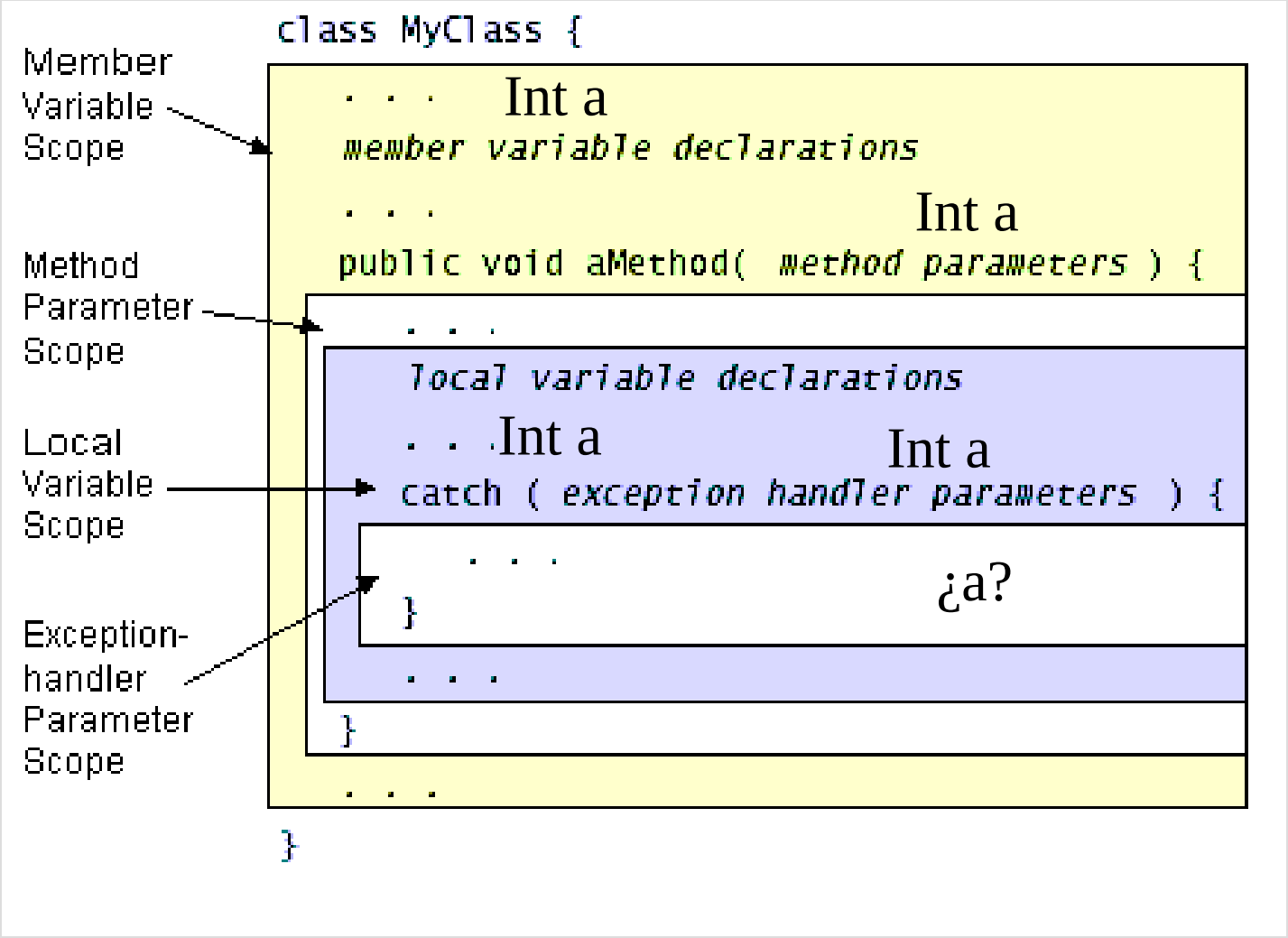
Ámbitos


class MyClass {
Member Variable Scope
. . .
member variable declarations
. . .
Method Parameter Scope
public void aMethod(method parameters) {
. . .
local variable declarations
. . .
Local Variable Scope
catch (exception handler parameters) {
. . .
}
Exception-handler Parameter Scope
. . .
}
. . .
}

Visibilidad

- Visibilidad: parte de programa en el que una variable es accesible sin ser calificada
- Las variables se pueden ocultar por otras con el mismo nombre en ámbitos más anidados

Visibilidad



Variables

```
public class Basic {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

¿Cúal es? ¿Qué tipo tienen? Alcance

Variables

```
public class Basic {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

¿Cúal es? ¿Qué tipo tienen? Alcance

Variables

```
public class Basic {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

¿Cúal es? ¿Qué tipo tienen? Alcance

Operadores

- Unarios
  op1 operator
  operator op1
- Binarios
  op1 operator op2
- Ternarios
  op1 operator1 op2 operator2 op3

Operadores aritméticos

- Binarios
  - Suma: op1 + op2
  - Resta: op1 – op2
  - Multiplicacion: op1 * op2
  - División: op1 / op2
  - Módulo: op1 % op2

Operadores aritméticos

- Operaciones con enteros y reales
- El resultado depende de los operadores
  - Algún double -> double
  - Algún float -> float
  - Algún long -> long
  - Si no, int

Operadores aritméticos

- Unarios
  - Número negativo (-2, -123)
  - Convierte byte y short en int (+2, +65)
- Suma / resta unaria
  - ++op1
  - op1++
  - --op1
  - op1--

Operadores de comparación

- Devuelven booleanos
  - Igualdad: op1 == op2
  - Desigualdad: op1 != op2
  - Mayor que: op1 > op2
  - Mayor o igual que: op1 >= op2
  - Menor que: op1 < op2
  - Menor o igual que: op1 <= op2

Operadores de comparación

- Mucho cuidado con la igualdad
- Cuando se comparan variables referencia, se compara si ambos objetos son el mismo, no si son iguales (tienen el mismo estado)
- Error típico

Operadores booleanos

- Operan sobre booleanos y devuelven booleanos
  - AND: op1 && op2
  - OR: op1 || op2
  - NOT: !op1
  - XOR: op1 ^ op2

Operadores booleanos

- Java sólo evalúa si es necesario
- Si el primer operando de un AND es false, no evalúa el segundo y devuelve false
  - Lo mismo con OR
- Para obligar a evaluar el segundo operando, usar & (AND) y | (OR)

Operadores de desplazamiento

- Opera sobre enteros y devuelve enteros
  - Desplazar a izquierda: op1 << num
  - Desplazar a derecha
    op1 >> num (extiende signo)
    op1 >>> num (no extiende signo)

Operadores lógicos

- Operan sobre bits de los enteros
  - AND: op1 & op2
  - OR: op1 | op2
  - XOR: op1 ^ op2
  - Complemento: ~op1

Operadores de asignación

- Asigna el valor de una variable a otra
  - op1 = op2
- Deben ser compatibles en tipo
  - Enteros, reales, carácter
  - Misma clase o subclases
- Al asignar variables referencia, no se hace una copia del objeto

Operadores de asignación

- Asignación con operación

  op1 = op1 operador op2 (a = a + b)

  op1 operador= op2 (a += b)

- +, -, *, /, %, &, |, ^, <<, >>, >>>

Otros operadores

- ?: (if-then-else)
  if (a == b) then c else d;
  a == b ? c : d
- [] - indexación de arrays
- . (punto): acceso a métodos y variables
- (): para especificar los argumentos a métodos

Casting

- Cambiar el tipo de una variable
- Cambios de tipo automáticos
- De int a float, de float a int

```
int a;
float b;
a = (int) b;   // Se pierde información
b = (float) a;  // No es necesario
```

Expresiones

- Una expresión es un conjunto de variables, operadores e invocaciones a métodos que se evalúan como un único resultado
  - a
  - 1 + 2
  - 12 + a.getNumHoras() * 2

Expresiones

- Las expresiones, además de un valor, tienen un tipo asociado, que depende de las subexpresiones dentro de la expresión
- Una expresión se puede conjuntar con otras para formar una expresión mayor mediante el uso de operadores

Expresiones

- Las expresiones se pueden emplear en
  - Asignaciones
  - Invocaciones a métodos
  - Operandos

Orden de evaluación

- Las expresiones complejas pueden evaluarse de diferentes formas

  a + b – c * 4

  ¿((a + b) – c) * 4?
  ¿((a + b) – (c * 4))?

Orden de evaluación

- Se pueden emplear paréntesis para especificar el orden de evaluación
  - ((a + b) – c) * 4
- Existen las reglas de precedencia
  - * y / más precedencia que + y -
- Pero es mejor despejar la ambigüedad mediante el uso de paréntesis
  - a + b – (c * 4)

Asociatividad

- En operadores binarios, ¿cómo se leen los operadores?
- Asociatividad a la izquierda: suma
  - 1 + 2 + 3 + 4 => (((1 + 2) + 3) + 4)
- Asociatividad a la derecha
  - a = b = c => (a = (b = (c)))

Sentencias

- Una sentencia es una unidad completa de ejecución y termina en punto y coma
- Sentencias de expresión
  - Una expresión terminada en ;
- Sentencias de declaración
- Sentencias de control de flujo

Bloque

- Un bloque es un conjunto de cero o más sentencias agrupadas entre llaves

```
{
   int a = 1120;
}
```

- Un bloque es, funcionalmente, como una sentencia y puede aparecer dónde puedan aparecer sentencias

Control de flujo

- Un programa es un conjunto de sentencias
- Hasta ahora, podemos hacer programas que usen variables, pero no podemos hacer nada para romper el hilo secuencial

Condicionales

- Permiten ejecutar ciertas sentencias dependiendo de una condición
- If / else / else if
- Switch / case
- ?:

If / else / else if

- Sólo ejecuta el cuerpo si la condición es cierta
- La condición debe ser booleana

```
if (condición) {
        cuerpo
}
```

If / else / else if

- Es posible especificar qué hacer si la condición no se cumple mediante el else

```
if (condición) {
        cuerpo1
}
else {
        cuerpo2
}
```

If / else / else if

- Se pueden encadenar varios condicionales
- Aunque más de una condición sea cierta, sólo se ejecuta el cuerpo de la condición que aparece la primera
- Si no se verifica ninguna condición, se ejecuta el else final

If / else / else if

```
if (condición1) {
        cuerpo1
}
else if (condición2){
        cuerpo2
}
else if (condición3) {
        cuerpo3
}
else {
        cuerpo4
}
```

switch

- Modo compacto de los if else anidados
- Sólo permite condiciones de igualdad
  - Si la condición es igual a 'a', se ejecuta cuerpo 1
  - Si ninguna se verifica se ejecuta 'default'

```
switch (condición) {
case a: cuerpo1
case b: cuerpo2
default: cuerpo3
}
```

switch

- Mucho cuidado con switch
- Se ejecutan las sentencias desde el case que verifica la condición hasta el final del switch o hasta un break.

Bucles

- Permiten ejecutar repetidamente un conjunto de sentencias (cuerpo) mientras la condición de bucle se cumpla
- Tres tipos
  - while
  - do while
  - for

while

- while: ejecuta el cuerpo del bucle mientras la condición sea cierta
- La condición se evalúa antes de la iteración

```
while (condición) {
        cuerpo
}
```

do while

- do while: ejecuta el cuerpo del bucle mientras la condición sea cierta
- La condición se evalúa al final de la iteración, con lo que siempre se ejecuta al menos una vez

```
do{
        cuerpo
} while (condición)
```

for

- Ejecuta el cuerpo del bucle mientras la condición sea cierta
- Antes de la primera iteración, realiza la inicialización
  - Ámbito del cuerpo del for
- Tras cada iteración, ejecuta el incremento

for

- Cualquiera de los tres puede ser vacío

```
for (inicialización; condición; incremento) {
        cuerpo
}
```

```
inicialización
while (condición) {
        cuerpo
        incremento
}
```

Break

- Rompe la ejecución de un bucle y devuelve el control al exterior
- Puede proporcionarse una etiqueta y salir de bucles anidados

```
while (condición) {
        ....
        break;
}
```

```
externo: while (condición) {
                while (condición2) {
                    break externo;
}}
```

Continue

- Termina la ejecución de una pasada del bucle y devuelve el control al comienzo
  - Ejecuta el incremento del for
- Puede proporcionarse una etiqueta y terminar la iteración de bucles anidados

```
while (condición) {
        ....
        continue;
}
```

```
externo: while (condición) {
                while (condición2) {
                  continue externo;
}}
```

Return

- Termina la ejecución del método y sale de él
- Si el método tienen que devolver un valor, debe especificarse en la sentencia *return (return a)*
- Si no se especifica, al final del método hay un *return* implícito

Método main

- Es un método especial
- Punto de entrada del programa
- public static void main(String args[])
- args: vector de cadenas de texto que representa los parámetros pasados al programa

Terminar la ejecución

- Un programa se para cuando no existe ningún hilo de ejecución
- También se puede terminar la aplicación con el método System.exit(int)
  - El entero es el código de retorno

OBJETOS CON JAVA

- Estado y comportamiento
- Clases, atributos y métodos
- Herencia, this y super
- Interfaces
- Paquetes
- Sobrecarga

OBJETOS CON JAVA

- Constructores y destructores
- Seguridad
- Casting y comparación
- Arrays y vectores
- Strings y cadenas de texto

Introducción

- Modelar los programas como interacción entre objetos
- Los objetos se describen mediante clases
- Las clases se instancian para crear un nuevo objeto

Objetos

- Ejemplos a nuestro alrededor
  - Hojas, bolígrafos, el profesor, etc...
- Dos características
  - Tienen un estado
  - Tienen un comportamiento
- Un objeto es un conjunto software de variables y los métodos asociados

Estado

- El estado se guarda en una o más variables
- Una variable es un espacio de almacenamiento con un tipo de datos asociado que se identifica mediante un nombre

Comportamiento

- El comportamiento se implementa mediante métodos
- Los métodos son trozos de código (subrutinas) asociados con el objeto

Encapsulación

- Podemos ver los objetos java como unas variables que guardan el estado y unos métodos que lo cambian
- El acceso a las variables y métodos está regulado por los calificadores de acceso

Encapsulación

- Modularidad: el uso de la encapsulación permite el desarrollo modular de nuestra aplicación
- Ocultación: el acceso al estado del objeto está regulado por los métodos, que aseguran que el estado del objeto es consistente

Mensajes

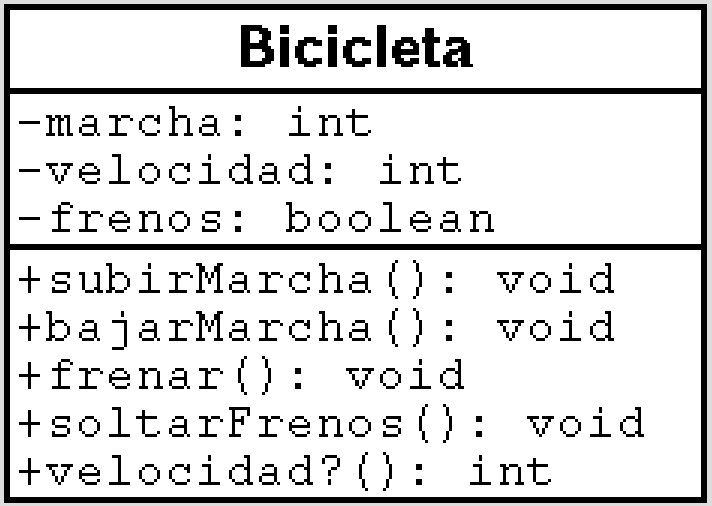
- Los objetos autónomos no son muy útiles: deben cooperar
- Se dice que pasan mensajes (invocación de métodos)
- Participantes
  - Emisor
  - Receptor
  - Contenido (parámetros)

Clase

- Cada objeto es independiente de otros objetos similares
- Pero todos comparten ciertas características
  - Pasan por estados similares
  - Tienen el mismo comportamiento

Clase

- Una clase es una plantilla, o prototipo, que define las variables y los métodos comunes a todos los objetos de cierto tipo



Instanciación

- Cuando una clase se instancia, se crea un nuevo objeto en el sistema
- Se crea una copia de todas las variables de instancia para almacenar el estado de este objeto

Estado

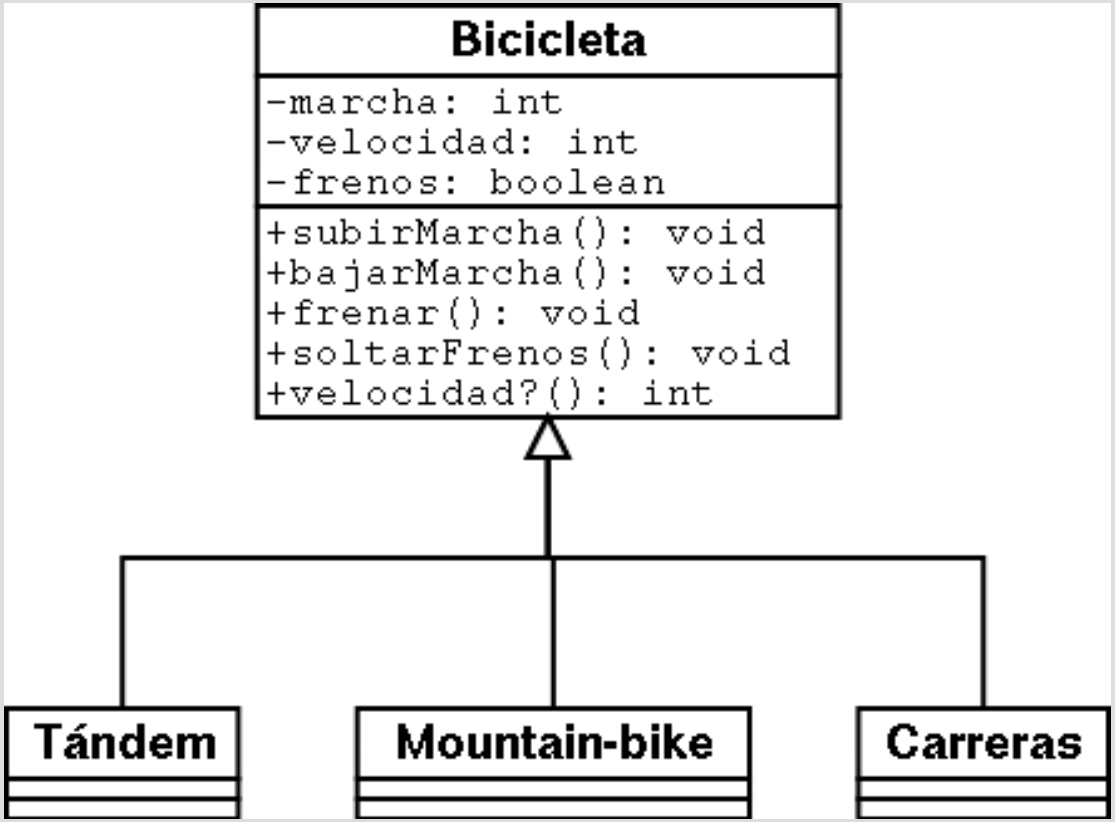
- El estado se guarda en variables
- Variables de instancia
  - Cada objeto tiene su valor propio e independiente del resto de los objetos
- Variables de clase o estáticas (static)
  - El valor es compartido por todos los objetos de la clase

Métodos

- Cambian el estado del objeto
- Métodos de instancia
  - Pueden acceder a variables de instancia o de clase
- Métodos de clase o estáticas (static)
  - Sólo pueden acceder a variables de clase

Herencia

- Se pueden definir clases en función de otras clases



Herencia

- Superclase: clase padre
  - Bicicleta es superclase de mountain-bike, tándem y carreras.
- Subclase: clase hija
  - Mountain-bike, tándem y carreras son subclases de bicicleta

Herencia

- Las subclases heredan de la superclase el estado y los comportamientos
  - Mountain-bike, tándem y carreras tienen las variables de marcha, velocidad y frenos y los métodos frenar, ...
- Pero pueden ser diferentes en algún aspecto

Herencia

- Las subclases pueden añadir nuevas variables y comportamientos
  - Para guardar un estado específico de la subclase
- Las subclases incluso pueden redefinir el comportamiento de un método para adaptarlo al nuevo estado

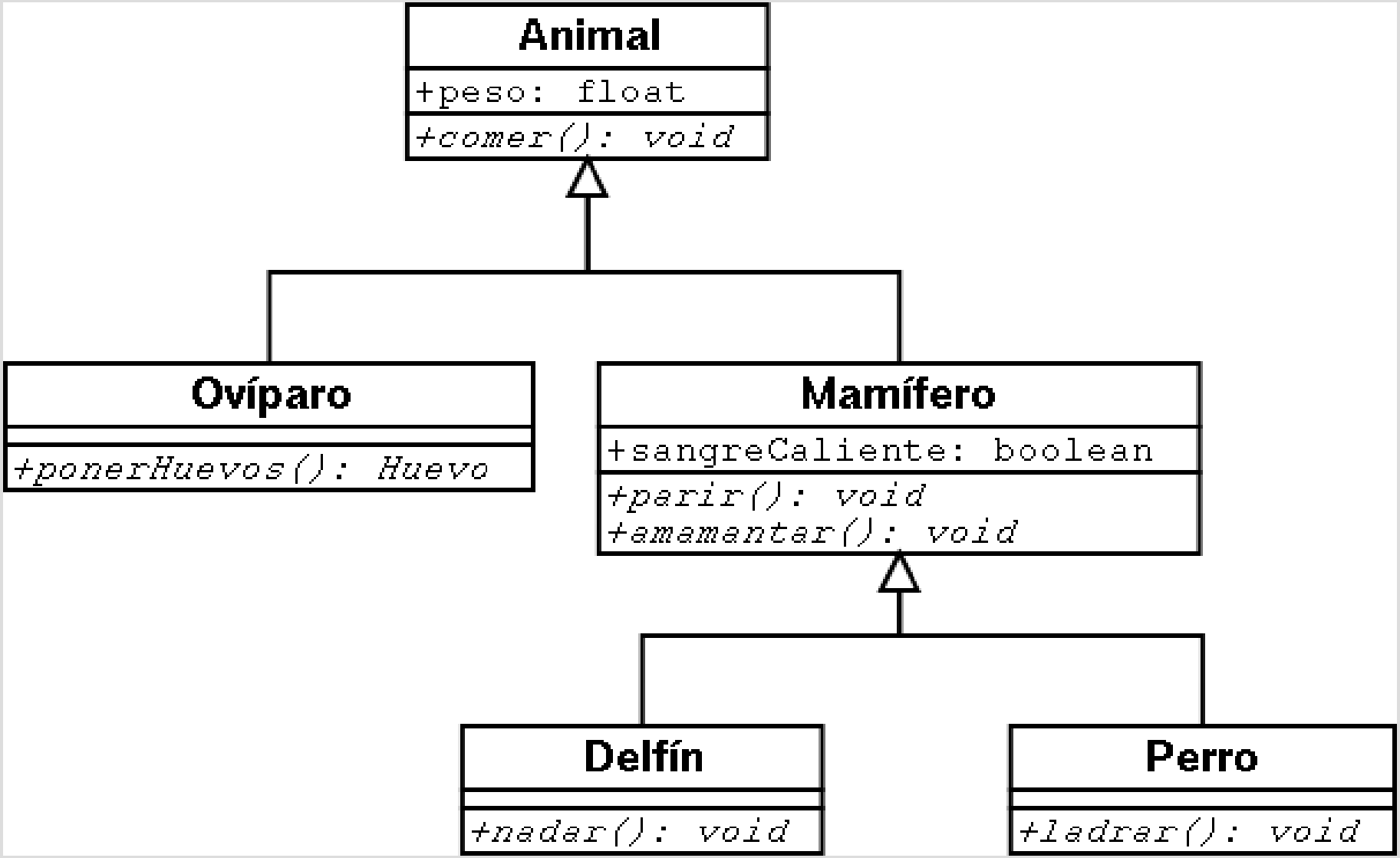
Herencia

- La relación superclase – clase – subclase forma una jerarquía
- Cuanto más abajo en la jerarquía, más especializada estará la clase
- En la cima de la jerarquía está *Object*

Herencia

- Se define con *extends*
- Java tiene herencia simple
  - Una clase sólo puede tener una única superclase

Herencia


Animal
+peso: float
+comer(): void
Ovíparo
+ponerHuevos(): Huevo
Mamífero
+sangreCaliente: boolean
+parir(): void
+amamantar(): void
Delfín
+nadar(): void
Perro
+ladrar(): void

Herencia

- La subclase puede redefinir lo métodos de la superclase
  - Para adaptarlos a su definición
- Para redefinir un método, sólo hay que crear un método en la subclase con la misma firma (nombre + argumentos) el de la superclase

Herencia

```
public class Padre {
        public int metodo(int a) {...}
}
```

```
public class Hija extends Padre{
        public int metodo(int a) {...}
}
```

Herencia

- La superclase puede evitar la redefinición mediante el modificador *final*

```
public class Padre {
        public final int metodo(int a) {...}
}
```

this

- Dentro de un método, hace referencia al objeto al que se aplica el método
- Sólo aparece en métodos de instancia
  - Si no hay instancia, no hay this
- Se puede usar para acceder a variables aunque estén ocultas

this

```
public class Prueba {
        private int a;
        public void metodo() {
                int a = 0;   // oculta la variable de instancia
                a = 3;       // accede a la variable local
                this.a = 2;  // accede a la variable de instancia
        }
}
```

super

- Hace referencia a la superclase del objeto
- Muy útil al redefinir métodos
- En los constructores, para seleccionar el constructor de la superclase

super

```
public class Prueba {
        public void metodo() {
                System.out.println(“Hola”);
        }
}
```

```
public class Subprueba extends Prueba {
        public void metodo() {
                super.metodo();        // Accede a la superclase
                System.out.println(“Adios”);
        }
}
```

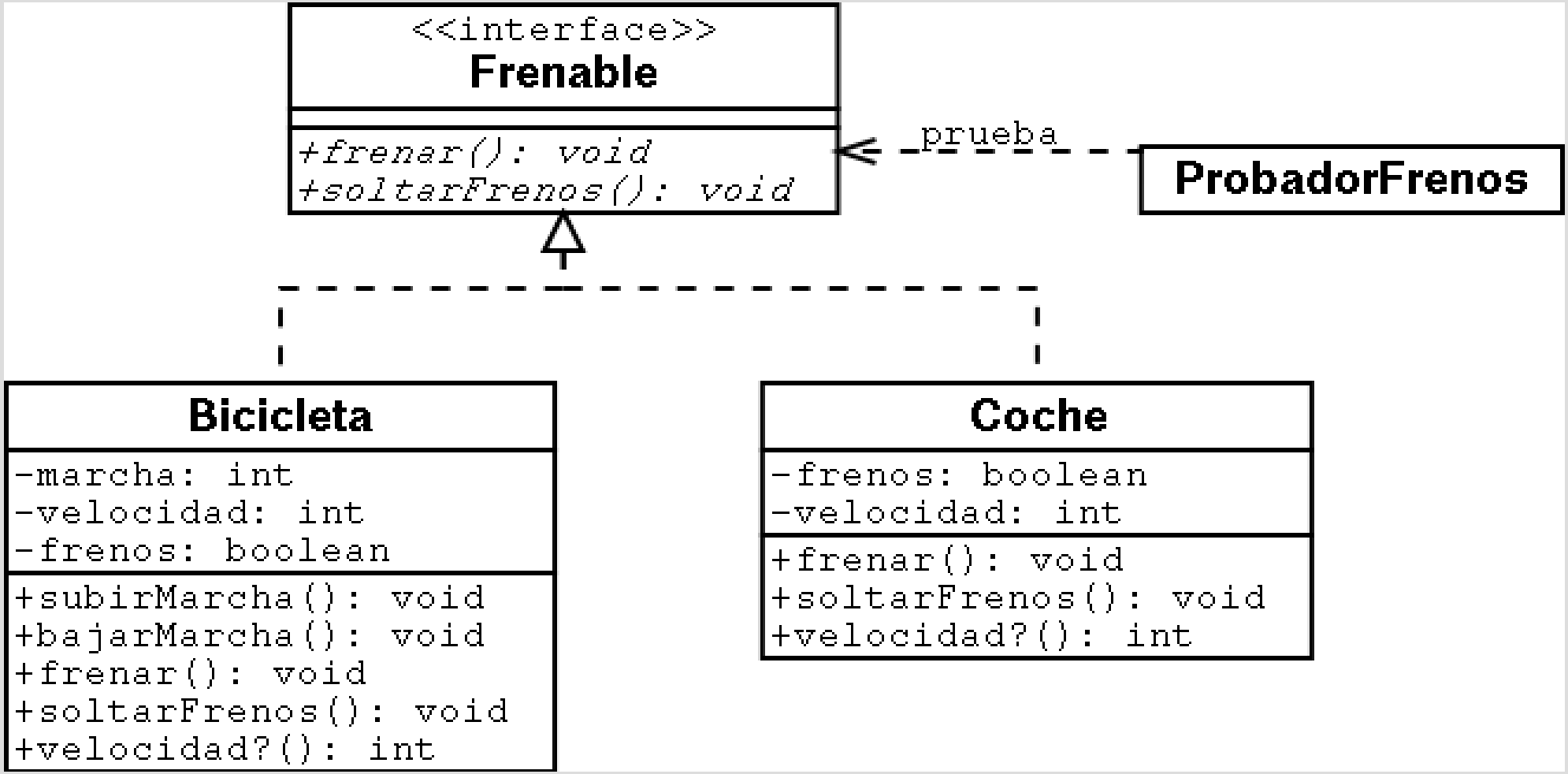
Interfaces

- Contrato que una clase (la que implementa la interfaz) se compromete a cumplir
- Seguro para los clientes, que saben qué comportamiento se encontrarán en las clases que implementan la interfaz

Interfaces

- Las interfaces definen constantes y métodos que usarán e implementarán las clases
- Las interfaces NO DEFINEN el cuerpo del método, sólo su firma (nombre y argumentos)

Interfaces



ejemplo

Interfaces

- Las interfaces también tienen jerarquía
- Una interfaz puede extender de otra interfaz
  - Hereda la definición de los métodos y las constantes
- La subinterfaz puede añadir nuevos métodos y constantes

Interfaces

- Una clase implementa una interfaz
  - public class Clase implements Interfaz
- La clase DEBE dar cuerpo a todos los métodos definidos en la interfaz
  - Si no, error de compilación

Interfaces

- Puede haber variables con tipo de la interfaz
  - Interfaz a;
- Pero no se puede instancia
  - a = new Interfaz() // ERROR
- Se deben asignar objetos de clases que implementan la interfaz

Paquetes

- Subclases: organización funcional
- Paquetes: organización administrativa
- Agrupación de clases a juicio del desarrollador
- Jerarquía: javax.swing.table

Paquetes

- Para definir a qué paquete pertenece una clase, usar la sentencia *package*
- *Separamos subpaquetes con puntos*

```
package cursillo.2004.ejemplos
public class Ejemplo extends Padre {
....
}
```

Paquetes

- Para usar clases de otros paquetes
- Referencia explícita
  - cursillo.2004.ejemplos.Ejemplo1 a;
- Importación
  - import cursillo.2004.ejemplos.Ejemplo1 Ejemplo1 a;

Paquetes

- Se pueden importar clases específicas
  - import cursillo.2004.ejemplos.Ejemplo1
- O todas las clases de un paquete
  - import cursillo.2004.ejemplos.*
- El asterisco no importa subpaquetes
- Por defecto, se importa java.lang.*

Usando objetos

- Declarar el objeto
  - <clase> <identificador>
  - Bicicleta miBicicleta;
- Pueden usarse clases o interfaces
  - Las interfaces no se pueden instanciar
  - Se instancia una clase que la implemente

Usando objetos

- La variable aún no apunta a ningún objeto
- Instanciar la clase
  - <identificador> = new <clase>(<args>)
  - miBicicleta = new Bicicleta();
- Se indica el constructor a emplear
  - Inicializa la clase

Usando objetos

¿Qué hay mal en este programa?

```
public class SomethingIsWrong {
    public static void main(String[] args) {
        Rectangle myRect;
        myRect.width = 40;
        myRect.height = 50;
        System.out.println("myRect's area is " + myRect.area());
    }
}
```

NUNCA SE CREA UN OBJETO. myRect no apunta a nada.

Variables de instancia

- Acceso a variables de instancia: <nombre>.<atributo>
  - int cambio = miCoche.marcha
- Modificar variables de instancia
  - miCoche.marcha = 2;

Variables de clase

- Acceso a variables de instancia: <nombre>.<atributo>
  - int cambio = miCoche.numRuedas
- Mediante la clase <clase>.<atributo>
  - int ruedas = Coche.numRuedas

Métodos de instancia

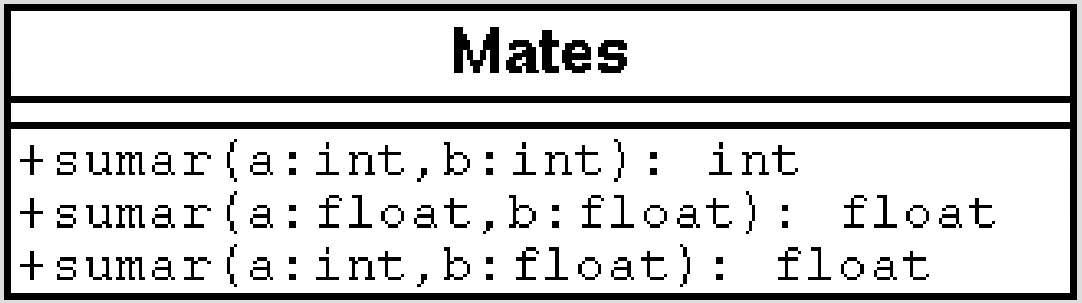
- Acceso a métodos de instancia: <nombre>.<método>(argumentos)
  - miCoche.subirMarcha();
  - miCoche.setMarcha(2);
  - miCoche.getMatrícula();

Métodos de clase

- Invocación de métodos de clase
  <nombre>.<método>(argurmentos)
  - miCoche.getNumRuedas();
- Mediante la clase
  <clase>.<método>(argumentos)
  - Coche.getNumRuedas()

Sobrecarga

- Pueden existir varios métodos con el mismo nombre, pero diferentes argumentos
- En tiempo de compilación se elige el método a invocar por los parámetros reales proporcionados



Constructores

- Método especial que se invoca cuando alguien crea un objeto de la clase
  - <acceso><nombre_clase>(<args>)
- Mismo nombre que la clase
- No tiene tipo de retorno
- Sirve para inicializar el objeto

Constructores

- Si no se especifica ningún constructor, Java crea el constructor vacío
  - public Clase() {;}
- Se pueden sobrecargar los constructores para aceptar diferentes tipos de argumentos

Constructores

- Se puede invocar desde el constructor a otro constructor de la clase
  - this(argumentos)
- Se puede invocar al constructor de la superclase para configurar las variables heredades
  - super(argumentos)
- Deben ser la primera instrucción

Limpieza de objetos

- Con Java no hay que liberar la memoria explícitamente
- El recolector de basura se encarga
- Subsistema que mira qué objetos no están referenciados para eliminarlos
  - Ninguna variable apunta al objeto

Limpieza de objetos

- Para hacer que una variable no apunte a un objeto
  - Asignar otro objeto (a = b)
  - Asignar el valor nulo (a = null)
- Antes de eliminar el objeto, se invoca el destructor

Destructores

- Método public void finalize()
  - Declarado en Object y heredado por todas las clases
- Cuerpo: liberar recursos
- No se garantiza
  - Ni en el momento en que se ejecuta
  - Ni que se ejecute

Seguridad en acceso

- Tanto los atributos como los métodos tienen un calificador de acceso
- Public
  - Pueden acceder todos los objetos
- Private
  - Sólo puede acceder el código de la misma clase

Seguridad en acceso

- Protected
  - Sólo puede acceder el código de la misma clase, de subclases o de clases en el mismo paquete
- Package protected
  - Sólo puede acceder el código de la misma clase o de clases en el mismo paquete

Casting

- Cambiar el tipo de un objeto
- No es necesario cuando es “hacia arriba”
- Obligatorio cuando es “hacia abajo”
  - Se comprueba en tiempo de ejecución

Casting

```
Bicicleta a = new Bicicleta();
Tandem b = new Tandem();

a = b; // No es necesario, porque un tándem es una bicicleta

b = (Tandem) a;  // Es necesario porque a es una bicicleta, pero
                                    // el compilador no sabe que hay un tan
                                    // Sólo se sabe en tiempo de ejecución.
                                    // que le damos la pista al compilador. S
                                    // tiempo de ejecución no hay un Tánde
                                    // habrá un error de tipos en tiempo de e
```

instanceof

- instanceof: nos informa si una variable es instancia de determinada clase
  - a instanceof Clase
  - Tiene en cuenta la transitividad
- Interesante para evitar errores de conversión (casting)

Operadores de comparación

- Comparar objetos
- Mismo objeto (=)
- Mismo contenido (equals)
- Código hash (hashCode)
- Métodos definidos en Object y redefinibles en subclases
  - = > equals > hashCode

Operadores de comparación

```
public int hashCode() {
 int h = hash;
 if (h == 0) {
  int off = offset;
  char val[] = value;
  int len = count;
  for (int i = 0; i < len; i++) {
   h = 31*h + val[off++];
  }
   hash = h;
 }
 return h;
}
```

```
public boolean equals(Object anObject) {
 if (this == anObject) return true;
 if (anObject instanceof String) {
  String anotherString = (String)anObject;
  int n = count;
  if (n == anotherString.count) {
   char v1[] = value;
   char v2[] = anotherString.value;
   int i = offset;
   int j = anotherString.offset;
   while (n-- != 0) {
    if (v1[i++] != v2[j++]) return false;
   }
    return true;
   }
  }
  return false;
}
```

Clase Object

- finalize
- equals, hashCode
- toString
- clone
- wait – notify
- getClass

Arrays

- Colección de elementos del mismo tipo
- <tipo> <nombre>[]
  - int precios[];
- Inicialización:
  <var> = new <tipo>[<num>]
  - precios = new int[80] //array de 80 precios
  - bicicletas = new Bicicletas[10];

Arrays

- Si los elementos del array son tipos primitivos, se crean y se inicializan a 0
- Si los elementos del array son tipos referencia (Clases e Interfaces), sólo se reserva espacio para los punteros
- Deberemos crear los objetos uno a uno para el array (con el operador *new*)

Arrays

- Obtener el valor:
  <nombre>[<posición>]
  - int a = precios[8]
- Guardar valor:
  <nombre>[<posición>] = valor
  - precios[6] = 50;

Arrays multidimensionales

- <tipo> <nombre>[][]...
  - int precios[][];
- Inicialización:
  <var> = new <tipo>[<num>][<num>]...
  - precios = new int[80][40] //array de 80x40

Arrays

- Obtener el valor:
  <nombre>[<posición>][<posición>]
  - int a = precios[8][10]
- Guardar valor:
  <nombre>[<pos>][<pos>] = valor
  - precios[6][12] = 50;

Arrays

- Los arrays no son dinámicos
  - Tienen tantos elementos como se indique al crearlo y no pueden cambiar
- Los arrays van desde 0 hasta tam – 1
- Para saber el tamaño: array.length
- Si te pasas de índice, excepción
  - ArrayIndexOutOfBoundsException

Arrays

```
String[] ciudades = {
    "Madrid", "Barcelona", "Bilbo", "Donosti",
    "Gasteiz", "Iruña"
};
```

¿Cúal es el índice de Bilbo? 2

¿Cómo es la expresión para acceder a Bilbo? ciudades[2]

¿Cuál es el resultado de ciudades.length? 6

¿Cuál es el índice del último elemento? 5

¿Cuál es el valor de la expresión ciudades[3]? Donosti

Arrays

```
public class WhatHappens {
    public static void main(String[] args) {
        StringBuffer[] stringBuffers = new StringBuffer[10];

        for (int i = 0; i < stringBuffers.length; i ++) {
            stringBuffers[i].append("StringBuffer at index " + i);
        }
    }
}
```

Se crea espacio para el array, pero no se crean los objetos StringBuffer del array, por lo que todos son nulos.

Vectores

- Clase Vector
  - En paquete java.util
- Implementa un contenedor de objetos
- Es dinámico
- Puedes obtener los objetos por posición

Iteradores

- Clase Iterator
  - En paquete java.util
- Permite recorrer secuencialmente el contenido de un vector
- hasNext()
- next()

Cadenas de caracteres

- Java proporciona una clase
  - String
- Los literales cadena: “Hola a todos”
- Se pueden asignar cadenas a variables directamente
  - String hola = “Hola mundo”;

Cadenas de caracteres

- El operador '+' concatena cadenas
  - “hol” + “a Mundo”: “hola Mundo”
  - String a = “Hola”;
    String b = “Mundo”;
    String c = a + ' ' + b; // “Hola Mundo”
- Algunos métodos de la clase String
  - length()
  - equals()

Cadenas de caracteres

- El método toString()
- Definido en Object
  - Da el nombre de la clase y la posición de memoria del objeto
- Se puede redefinir para que devuelva lo que queramos

CONCEPTOS AVANZADOS

- Excepciones
- Polimorfismo
- Abstracción

Excepciones

- Sirven para informar que se ha producido una situación extraña y que debe tratarse
- Se rompe la ejecución y se salta a un manejador de excepciones
- Mejor que comprobar valores de retorno

Excepciones

```
try  {
        cuerpo1
}
catch (excepción) {
        cuerpo2
}
finally {
        cuerpo3
}
```

```
throw excepción
```

```
public class UsaExcepciones {
  public void metodo() throws excepción {
     ...
  }
}
```

Excepciones

- cuerpo1 está monitorizado para excepciones
- Si se produjera una, se compararía la excepción contra la descrita en el catch
- Si es asignable, se ejecuta cuerpo2
- Si no se gestiona, se sigue buscando un gestor para la excepción

Excepciones

- Independientemente de si ha habido o no excepciones, siempre se ejecuta cuerpo4
- Las excepciones son clases, como el resto, sólo que Java las trata diferente
- Se crean con *new* y se lanzan con *throw*

Excepciones

- Las excepciones heredan de la clase Throwable
- Sólo instancias de clases que hereden de esa superclase pueden aparecer en cláusulas *throw*, *throws* o *catch*
- *throws* indica que el método no trata la excepción y que la delega hacia arriba

Excepciones Error

- Excepciones muy inusuales y que no se suelen tratar
  - VirtualMachineError
  - OutOfMemoryError
  - StackOverflowError
  - LinkageError
- No es obligatorio capturarlas

Excepciones Exception

- Excepciones que deben tratarse
  - IOException
  - RemoteException
  - NotSerializableException
- Es obligatorio capturarlas
  - Error de compilación

Excepciones RuntimeException

- Excepciones que derivan de RuntimeException
  - ArrayIndexOutOfBoundsExcepion
  - NullPointerException
- No es obligatorio capturarlas
  - Pero es recomendable
  - No hay error de compilación

Excepciones

- Podemos crear nuestras propias excepciones, creando clases que heredan (extends) de Throwable
  - O alguna subclase de ésta

Excepciones

- El control se transfiere al primer *catch* que tenga como captura la clase o alguna superclase de la excepción lanzada
- Ordenar los catch de más a menos especificidad

Excepciones

```
public void metodo() {
        try {
                .....
        }
        catch (FileNotFoundException ex) {...}
        catch (IOException ex) {...}
        catch (Exception ex) {...}
        catch (Throwable ex) {...}
        finally {....}
}
```

Excepciones

- Podemos relanzar las excepciones dentro de los bloque *catch*
- El método termina y la excepción aparece en el código llamante

```
try {
        .....
}
catch (IOException ex) {throw ex;}
finally {...}
```

Polimorfismo

- Cualidad por la que el método a ejecutar se resuelve en tiempo de ejecución
- No es sencillo a primera vista saber el método que se ejecutará

Polimorfismo

```
public class A {
 public void metodo() {
  System.out.println(“Soy A”);
 }
}
```

```
public class B extends A {
 public void metodo() {
  System.out.println(“Soy B”);
 }
}
```

```
A a = new A();
B b = new B();
a.metodo();     // Soy A
b.metodo();    // Soy B
a = b;
a.metodo();     // Soy ??
```

A y B

Abstracción

- Un método es abstracto cuando no se escribe el cuerpo del mismo
  - Se define su firma sin dar implementación

```
public abstract class Padre {
        public abstract int metodo(int a) ;
}
```

Abstracción

- Una clase es abstracta cuando tiene uno o más métodos abstractos
  - No se puede instanciar
  - Puede ser abstracta sin tener métodos abstractos
  - Puede haber métodos abstractos mezclados con métodos no abstractos

Abstracción & Polimorfismo

```
public abstract class Hablar {
	public abstract void diHola() ;
	public void saluda(String quien) {
		this.diHola();
		System.out.println(quien);
	}
}
```

Abstracción & Polimorfismo

```
public class Castellano extends Hablar{
        public void diHola() {System.out.print("Hola");}
}
```

```
public class Euskera extends Hablar{
        public void diHola() {System.out.print("Kaixo");}
}
```

```
Castellano a = new Castellano();
Euskera b = new Euskera();
a.saluda("Juan");      // Hola Juan
b.saluda("Patxi");   // Kaixo Patxi
```

ejemplo

Introducción a SWING

- Interfaces gráficas en Java
- Segunda versión
  - Tras AWT
- Independiente de la plataforma
  - Se ve igual en todas
  - En AWT, la visualización dependía de la plataforma

Componentes

- Todos y cada uno de los elementos de SWING
- Por ejemplo:
  - JFrame
  - JLabel
  - JButton
  - etc.

Contenedores

- Aquellos componentes que pueden albergar otros componentes en su interior
- Los contenedores están formados por uno o más componentes
  - JFrame
  - JDialog
  - JPanel

JFrame

- La tradicional ventana
- Tiene un título (setText())
- Y luego componentes internos
  - frame.getContentPane().add(<compo>)
- Pero más fácil modelar con JDeveloper

JLabel

- Etiqueta no modificable
- Tiene un contenido
  - setText() / getText()
- No tiene un comportamiento destacable

JButton

- Presionar para realizar una acción
- Tiene un título
  - setText() / getText()
- Tiene un comportamiento
  - actionPerformed()

Eventos

- Sucesos asíncronos
  - No sabemos cuándo se producen
- Sistema de escuchadores
  - Listeners
- Un objeto dice a un componente que cuando pase algo, le avise
  - El botón se pulsa

Eventos

```
...
JButton boton = new Jbutton(“Púlsame”);
boton.addActionListener(new Escuchador());
...
```

```
Public class Escuchador implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
                ...
        }
}
```

Ejemplo

- Mostrar en pantalla una etiqueta con un número
- Cada vez que se pulse el botón, se incrementa y se muestra de nuevo

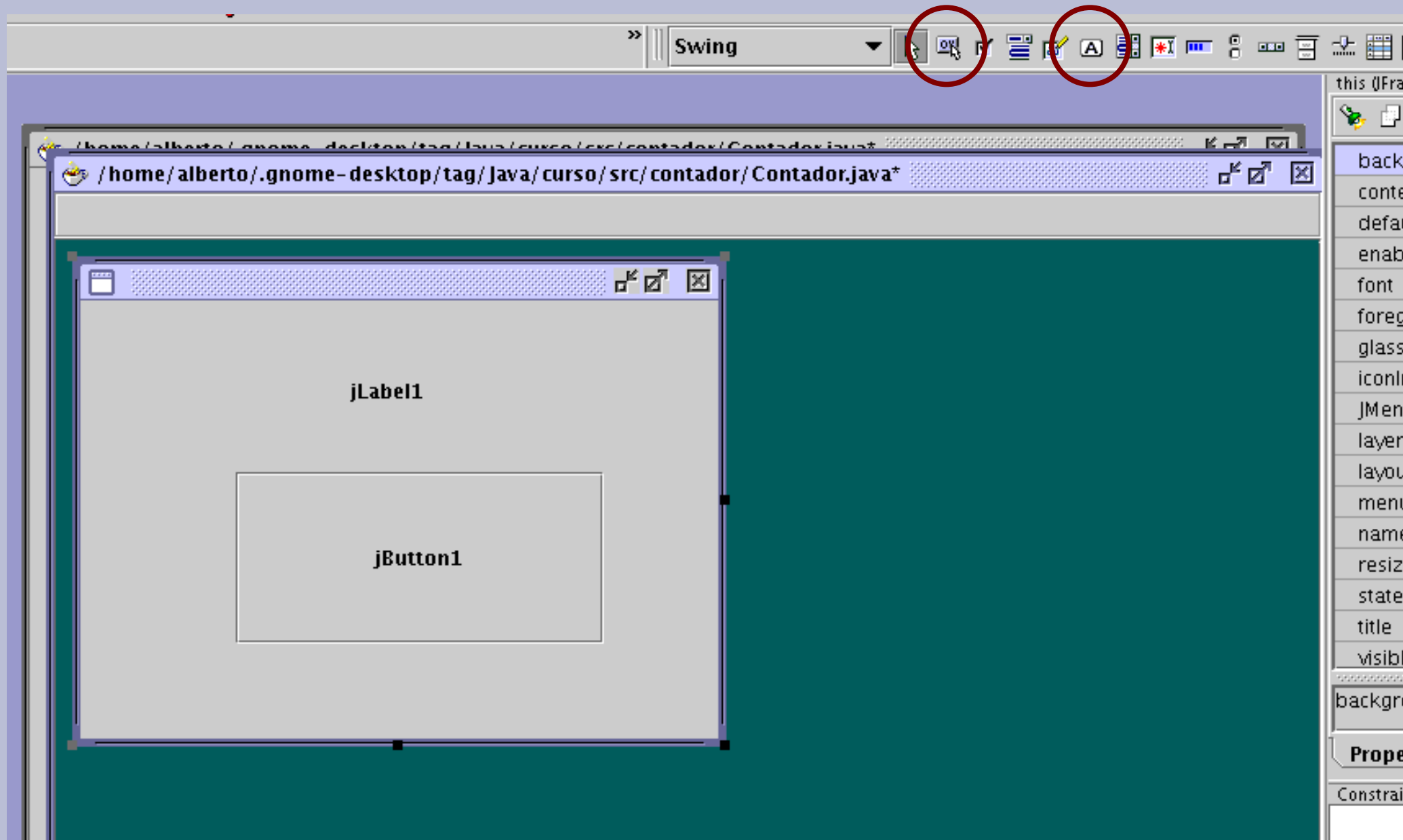
Ejemplo

- Clase Contador
  - Almacena la variable
  - Crea la ventana
  - Crea el botón
- En Jdeveloper
  - File -> New
  - Client Tier -> SWING/AWT -> Frame

Ejemplo


New Frame
What are the details of your new class?
Name:
Contador
Package:
contador
Browse...
Extends:
javax.swing.JFrame
Browse...
Optional Attributes
Title:
Contador
Menu Bar
Tool Bar
Status Bar
About Box
Help
OK
Cancel

Ejemplo


Swing
/home/alberto/.gnome-desktop/tag/Java/curso/src/contador/Contador.java*
jLabel1
jButton1

Ejemplo

- Poner de etiquetas
  - Seleccionar componente
  - Cambiar atributo text
- En el botón, "Pínchame"
- Dejar la etiqueta a vacío

Ejemplo

- En el botón, definir el comportamiento
- Seleccionar botón
- Seleccionar "Events" en la parte de las propiedades
- Seleccionar "actionPerformed", pinchar en los tres puntos y dar a OK

Ejemplo

- Dar el comportamiento
- Incrementar en uno la variable del contador
  - Crearla si es necesario
- Actualizar la etiqueta
  - Con setText(<texto>)
  - Crear un objeto String usando el entero en el constructor
  - La etiqueta se llama jLabel1

Ejemplo

- Cread un método main
  - public static void main(String[] args)
- Dentro del main
  - Crear un objeto de tipo Contador
  - Mostrarlo en pantalla invocando el método show()

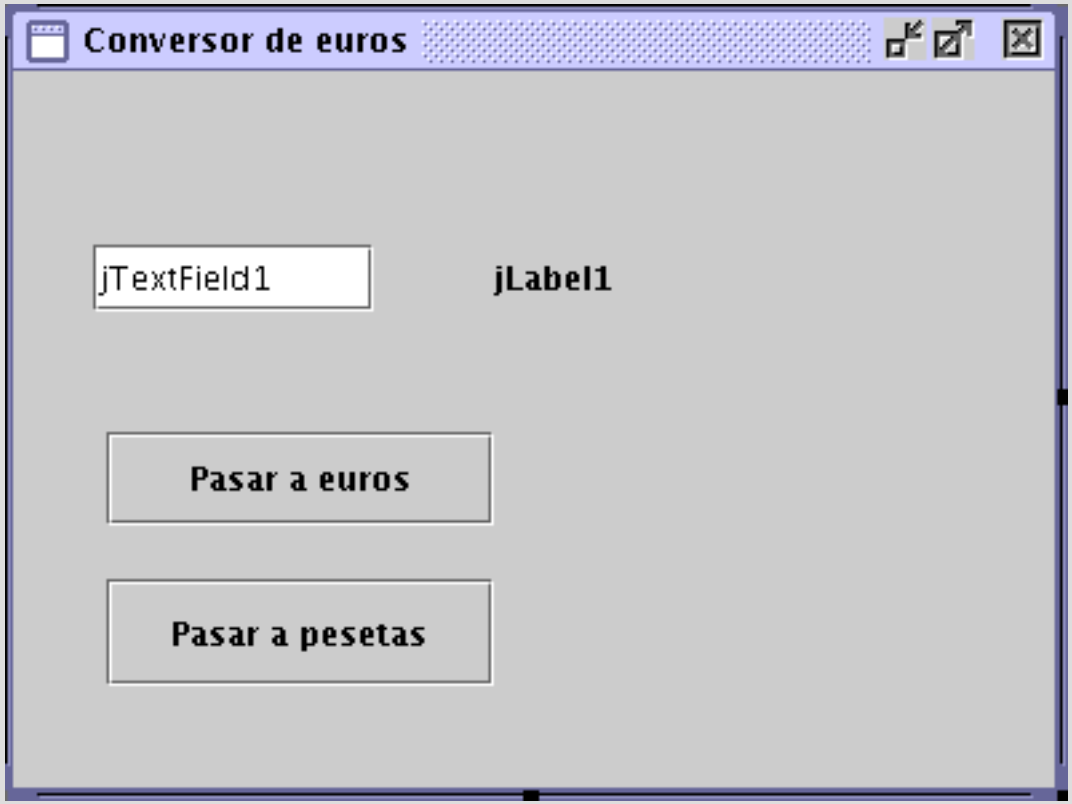
JTextField

- Puedes teclear en él
- Para obtener o cambiar el contenido
  - getText() / setText()
- Comportamiento si se pulsa Intro
  - actionPerformed

Calculadora de euros

- Ponemos la cantidad a convertir en un JTextField
- Tendremos dos botones
  - Uno para pasar de euros a pesetas
  - Y otro para pasar de pesetas a euros
- Cuando se pulsa el botón, se debe reflejar la conversión en un JLabel

Calculadora de euros



Calculadora de euros

- Igual que el ejemplo anterior
- Recoger el valor del JTextField
  - Con getText()
  - Devuelve un String
- Para pasar de String a real
  - Double.parseDouble(<texto>)
  - Lanza NumberFormatException
  - Si ocurre, escribir por la salida estándar