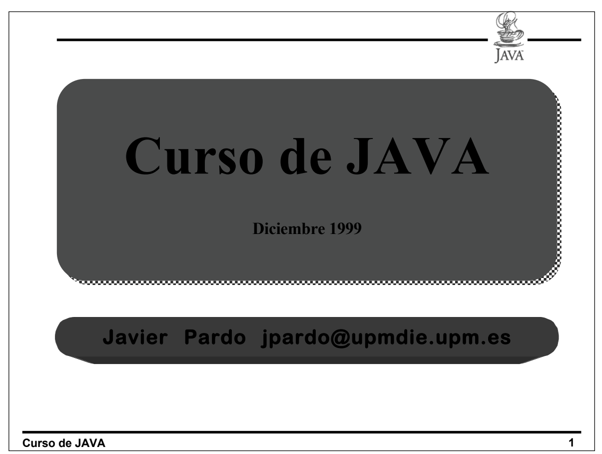
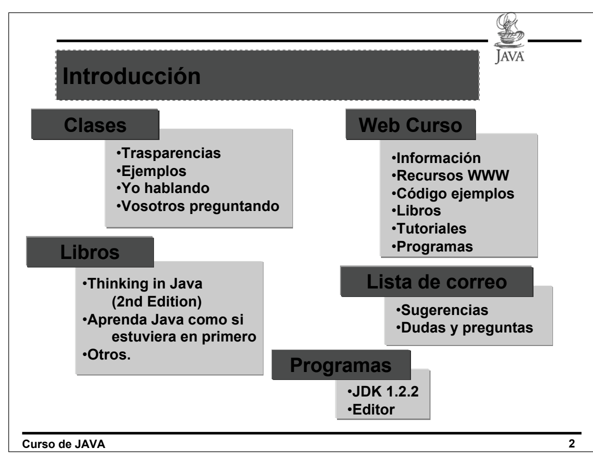
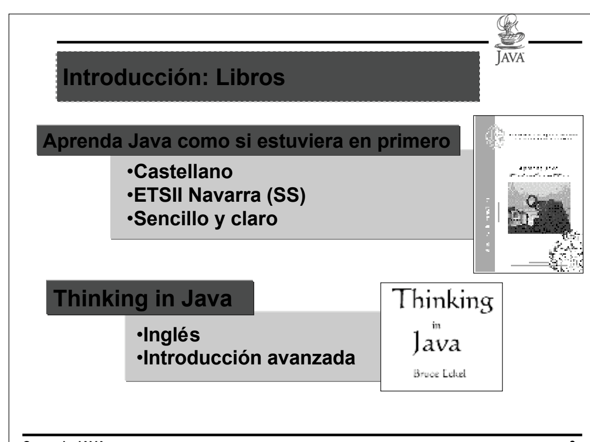
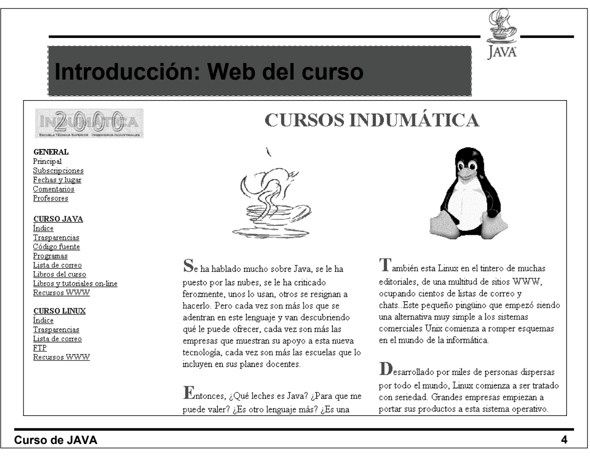
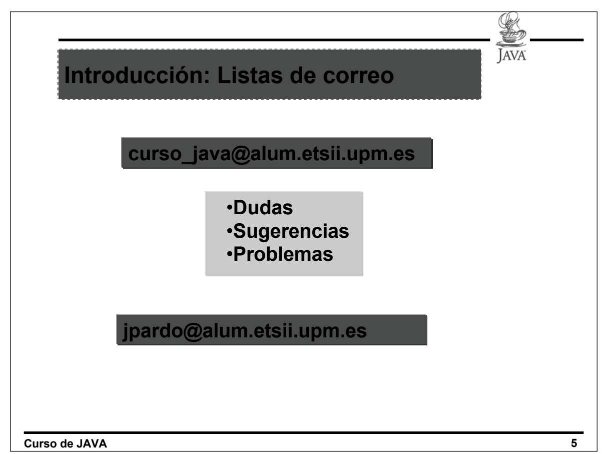
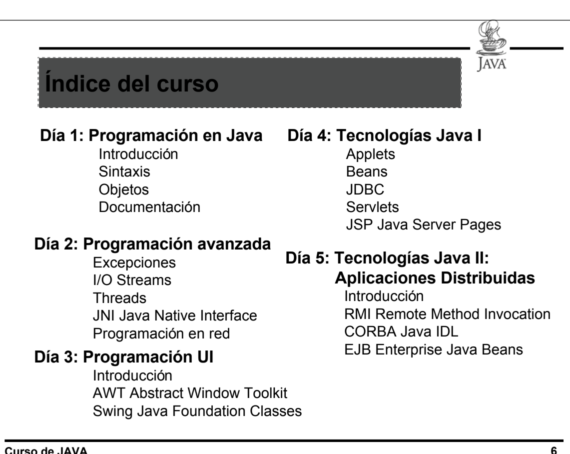

JAVA
Curso de JAVA
Diciembre 1999
Javier Pardo jpardo@upmdie.upm.es
Curso de JAVA
1

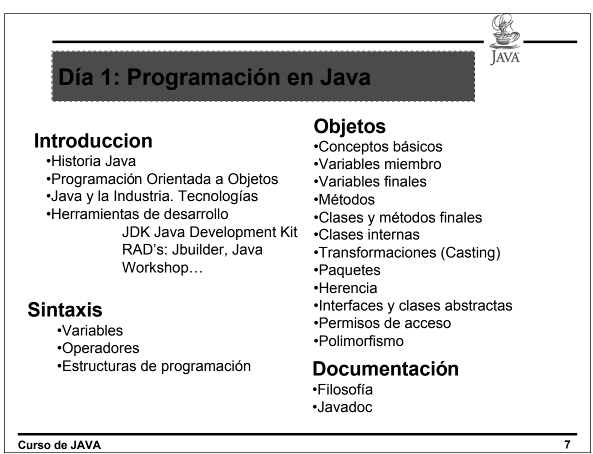

JAVA
Introducción
Clases
•Trasparencias
•Ejemplos
•Yo hablando
•Vosotros preguntando
Web Curso
•Información
•Recursos WWW
•Código ejemplos
•Libros
•Tutoriales
•Programas
Libros
•Thinking in Java
(2nd Edition)
•Aprenda Java como si
estuviera en primero
•Otros.
Lista de correo
•Sugerencias
•Dudas y preguntas
Programas
•JDK 1.2.2
•Editor
Curso de JAVA
2

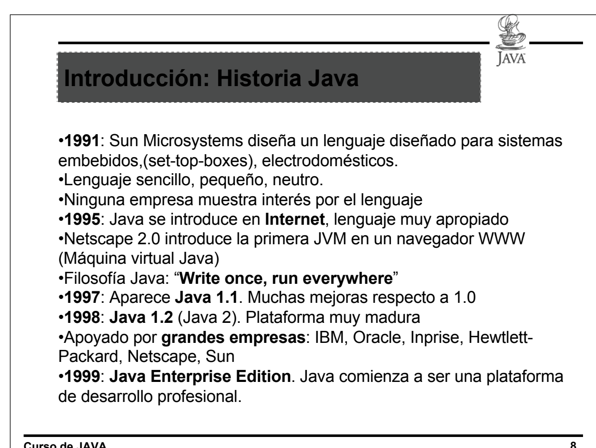

JAVA
Introducción: Libros
Aprenda Java como si estuviera en primero
•Castellano
•ETSII Navarra (SS)
•Sencillo y claro
Thinking in Java
•Inglés
•Introducción avanzada
Thinking
in
Java
Bruce Eckel
Curso de JAVA
3


JAVA
Introducción: Web del curso
CURSOS INDUMÁTICA
GENERAL
Principal
Subscripciones
Fechas y lugar
Comentarios
Profesores
CURSO JAVA
Índice
Trasparencias
Código fuente
Programas
Lista de correo
Libros del curso
Libros y tutoriales on-line
Recursos WWW
CURSO LINUX
Índice
Trasparencias
Lista de correo
FTP
Recursos WWW
Se ha hablado mucho sobre Java, se le ha puesto por las nubes, se le ha criticado ferozmente, unos lo usan, otros se resignan a hacerlo. Pero cada vez son más los que se adentran en este lenguaje y van descubriendo qué le puede ofrecer, cada vez son más las empresas que muestran su apoyo a esta nueva tecnología, cada vez son más las escuelas que lo incluyen en sus planes docentes.
Entonces, ¿Qué leches es Java? ¿Para que me puede valer? ¿Es otro lenguaje más? ¿Es una
También esta Linux en el tintero de muchas editoriales, de una multitud de sitios WWW, ocupando cientos de listas de correo y chats..Este pequeño pingüino que empezó siendo una alternativa muy simple a los sistemas comerciales Unix comienza a romper esquemas en el mundo de la informática.
Desarrollado por miles de personas dispersas por todo el mundo, Linux comienza a ser tratado con seriedad. Grandes empresas empiezan a portar sus productos a esta sistema operativo.
Curso de JAVA
4

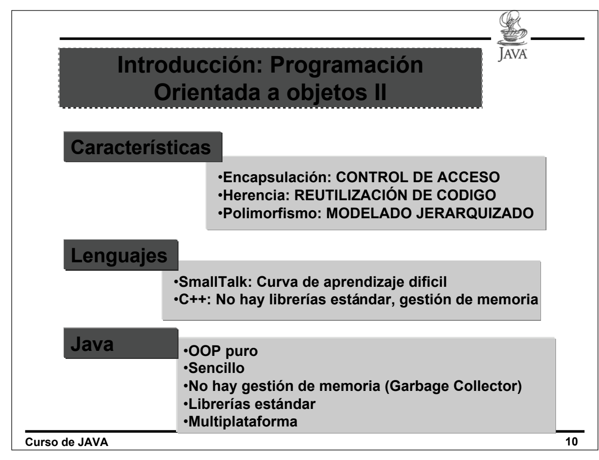

JAVA
Introducción: Listas de correo
curso_java@alum.etsii.upm.es
•Dudas
•Sugerencias
•Problemas
jpardo@alum.etsii.upm.es
Curso de JAVA
5

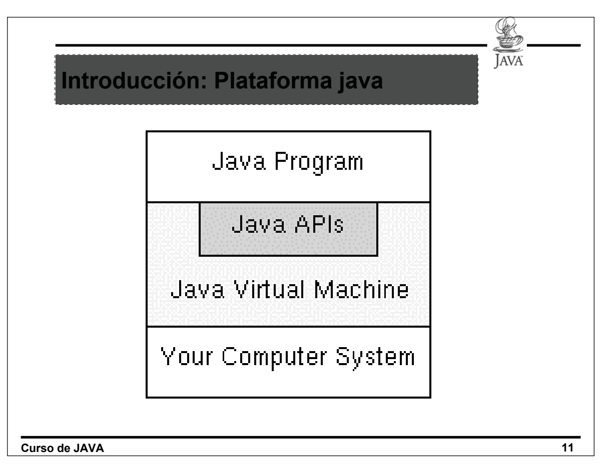

JAVA
Índice del curso
Día 1: Programación en Java
Introducción
Sintaxis
Objetos
Documentación
Día 2: Programación avanzada
Excepciones
I/O Streams
Threads
JNI Java Native Interface
Programación en red
Día 3: Programación UI
Introducción
AWT Abstract Window Toolkit
Swing Java Foundation Classes
Día 4: Tecnologías Java I
Applets
Beans
JDBC
Servlets
JSP Java Server Pages
Día 5: Tecnologías Java II:
Aplicaciones Distribuidas
Introducción
RMI Remote Method Invocation
CORBA Java IDL
EJB Enterprise Java Beans
Curso de JAVA
6

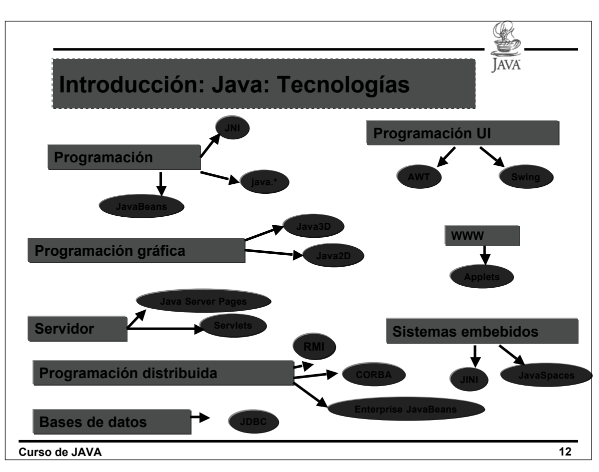

Día 1: Programación en Java
Introduccion
•Historia Java
•Programación Orientada a Objetos
•Java y la Industria. Tecnologías
•Herramientas de desarrollo
JDK Java Development Kit
RAD's: Jbuilder, Java Workshop…
Sintaxis
•Variables
•Operadores
•Estructuras de programación
Objetos
•Conceptos básicos
•Variables miembro
•Variables finales
•Métodos
•Clases y métodos finales
•Clases internas
•Transformaciones (Casting)
•Paquetes
•Herencia
•Interfaces y clases abstractas
•Permisos de acceso
•Polimorfismo
Documentación
•Filosofía
•Javadoc
Curso de JAVA
7

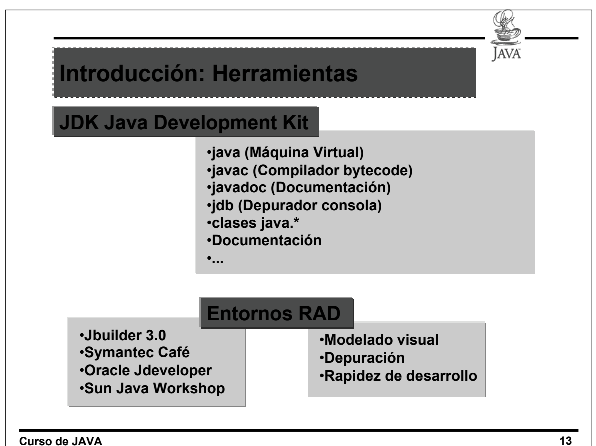

Introducción: Historia Java
•1991: Sun Microsystems diseña un lenguaje diseñado para sistemas embebidos,(set-top-boxes), electrodomésticos.
•Lenguaje sencillo, pequeño, neutro.
•Ninguna empresa muestra interés por el lenguaje
•1995: Java se introduce en Internet, lenguaje muy apropiado
•Netscape 2.0 introduce la primera JVM en un navegador WWW (Máquina virtual Java)
•Filosofía Java: "Write once, run everywhere"
•1997: Aparece Java 1.1. Muchas mejoras respecto a 1.0
•1998: Java 1.2 (Java 2). Plataforma muy madura
•Apoyado por grandes empresas: IBM, Oracle, Inprise, Hewtlett-Packard, Netscape, Sun
•1999: Java Enterprise Edition. Java comienza a ser una plataforma de desarrollo profesional.
Curso de JAVA
8

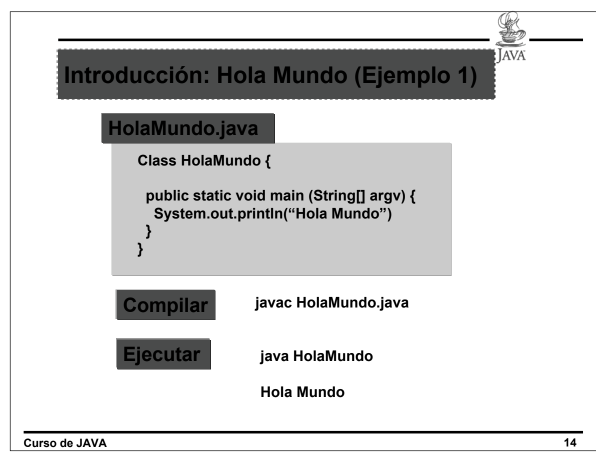
Introducción: Programación Orientada a objetos
Directores proyecto
•Rapidez desarrollo
•Menores costes
•Mantenimiento sencillo
Diseñadores y analistas
•Modelado rápido
•Diseño claro
Programadores
•Elegancia
•Claridad
•Reutilización código
•...
Inconvenientes
•Curva de aprendizaje
Curso de JAVA
9

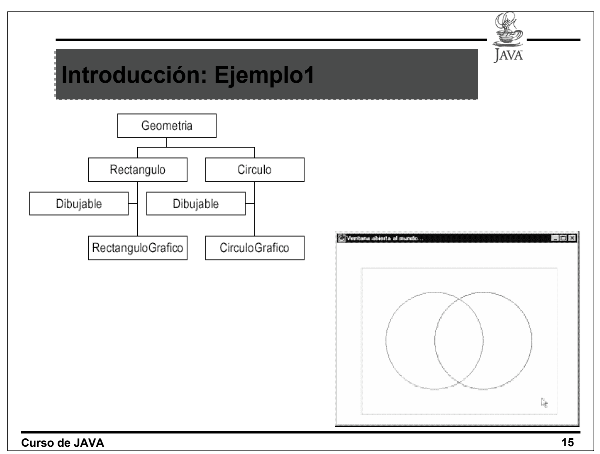

Introducción: Programación Orientada a objetos II
Características
•Encapsulación: CONTROL DE ACCESO
•Herencia: REUTILIZACIÓN DE CODIGO
•Polimorfismo: MODELADO JERARQUIZADO
Lenguajes
•SmallTalk: Curva de aprendizaje dificil
•C++: No hay librerías estándar, gestión de memoria
Java
•OOP puro
•Sencillo
•No hay gestión de memoria (Garbage Collector)
•Librerías estándar
•Multiplataforma
Curso de JAVA
10

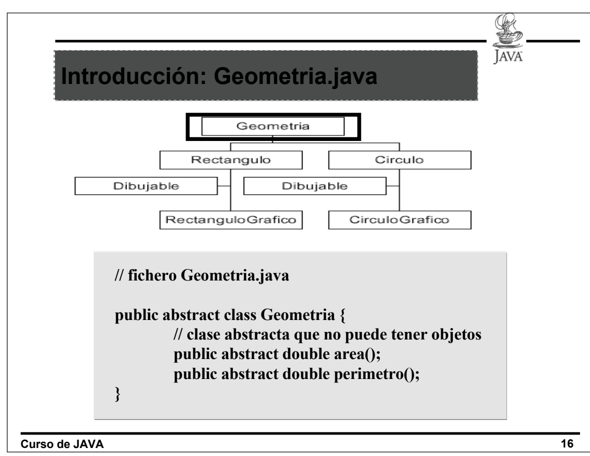

Introducción: Plataforma java
Java Program
Java APIs
Java Virtual Machine
Your Computer System
Curso de JAVA
11


Introducción: Java: Tecnologías
Programación
JNI
java.*
JavaBeans
Programación UI
AWT
Swing
Programación gráfica
Java3D
Java2D
WWW
Applets
Servidor
Java Server Pages
Servlets
Sistemas embebidos
JINI
JavaSpaces
Programación distribuida
RMI
CORBA
Enterprise JavaBeans
Bases de datos
JDBC
Curso de JAVA
12









JAVA

Introducción: Rectangulo.java

```
// fichero Rectangulo.java

class Rectangulo extends Geometria {
        // definición de las variables miembro
        private static int numRectangulos = 0;
        protected double x1, y1, x2, y2;
        // constructor por defecto (sin argumentos)
        // se define mediante una llamada al constructor general
        public Rectangulo(){ this(0, 0, 1.0, 1.0); }
        // constructor de la clase
        public Rectangulo(double p1x, double p1y, double p2x, double p2y) {
                x1 = p1x;
                x2 = p2x;
                y1 = p1y;
                y2 = p2y;
                numRectangulos++;
        }
        public double area() { return (x1-x2)*(y1-y2); }
        public double perimetro() { return 2.0 * ((x1-x2)+(y1-y2)); }
} // fin de la clase Rectangulo
```

Curso de JAVA 17

JAVA

Introducción: Circulo.java

```
// fichero Circulo.java

public class Circulo extends Geometria {
        static int numCirculos=0;
        public static final double PI=3.14159265358979323846;
        public double x, y, r;

        public Circulo() { this(0.0, 0.0, 1.0); }
        public Circulo(double r) { this(0.0, 0.0, r); }
        public Circulo(double x, double y, double r) {
                this.x=x; this.y=y; this.r=r;
                numCirculos++;
        }
        public Circulo(Circulo c) { this(c.x, c.y, c.r); }
        // método de objeto para comparar círculos
        public Circulo elMayor(Circulo c) {
                if (this.r>=c.r) return this; else return c;
        }
        // método de clase para comparar círculos
        public static Circulo elMayor(Circulo c, Circulo d) {
                if (c.r>=d.r) return c; else return d;
        }
        public double area() { return PI * r * r; }
        public double perimetro() { return 2.0 * PI * r; }
} // fin de la clase Circulo
```

Curso de JAVA 18

Introducción: Dibujable.java

Geometria
Rectangulo
Circulo
Dibujable
Dibujable
RectanguloGrafico
CirculoGrafico

```
// fichero Dibujable.java

import java.awt.Graphics;

public interface Dibujable {
        public void dibujar(Graphics dw);
        public void setPosicion(double x, double y);
}
```


Introducción: RectanguloGrafico.java

```
// Fichero RectanguloGrafico.java
import java.awt.Graphics;
import java.awt.Color;

class RectanguloGrafico extends Rectangulo implements Dibujable
{
        Color color;
        // constructor
        public RectanguloGrafico(double x1, double y1, double x2,double y2, Color unColor) {
                // llamada al constructor de Rectangulo
                super(x1, y1, x2, y2);
                this.color = unColor;
        }
        // métodos de la interface Dibujable
        public void dibujar(Graphics dw) {
                dw.setColor(color);
                dw.drawRect((int)x1, (int)y1, (int)(x2-x1), (int)(y2-y1));
        }
        public void setPosicion(double x, double y) {
                ;
        }
} // fin de la clase RectanguloGrafico
```


Introducción: CirculoGrafico.java

```
// fichero CirculoGrafico.java

import java.awt.Graphics;
import java.awt.Color;

public class CirculoGrafico extends Circulo implements Dibujable {
        // se heredan las variables y métodos de la clase Circulo
        Color color;
        // constructor
        public CirculoGrafico(double x, double y, double r, Color unColor) {
                // llamada al constructor de Circulo
                super(x, y, r);
                this.color = unColor;
        }
        // métodos de la interface Dibujable
        public void dibujar(Graphics dw) {
                dw.setColor(color);
                dw.drawOval((int)(x-r), (int)(y-r), (int)(2.0*r), (int)(2.0*r));
        }
        public void setPosicion(double x, double y) {
                ;
        }
} // fin de la clase CirculoGrafico
```


Introducción: PanelDibujo.java

```
// fichero PanelDibujo.java

import java.awt.*;
import java.util.Vector;
import java.util.Enumeration;

public class PanelDibujo extends Panel {
        private Vector v;
        // constructor
        public PanelDibujo(Vector vect) {
                super(new FlowLayout());
                this.v = vect;
        }
        public void paint(Graphics g) {
                Dibujable dib;
                Enumeration e;
                e = v.elements();
                while(e.hasMoreElements()){
                        dib=(Dibujable)e.nextElement();
                        dib.dibujar(g);
                }
  }
} // Fin de la clase DrawWindow
```

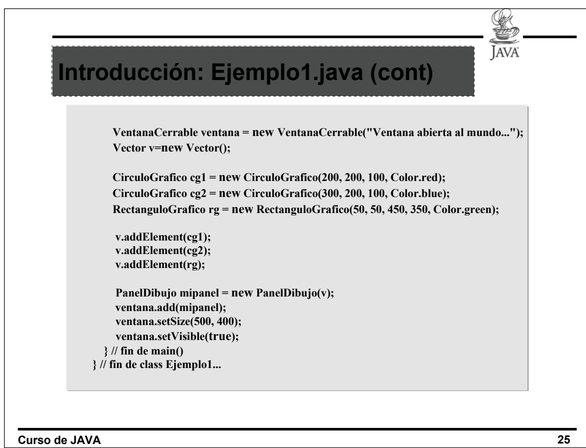

Introducción: VentanaCerrable.java

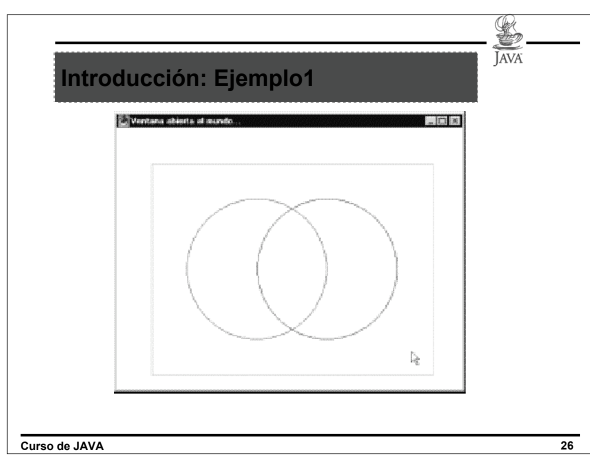
```
// Fichero VentanaCerrable.java

import java.awt.*;
import java.awt.event.*;

class VentanaCerrable extends Frame implements WindowListener {

public VentanaCerrable() {
        super();
}
public VentanaCerrable(String title) {
        super(title);
        setSize(500,500);
        addWindowListener(this);
}
        public void windowActivated(WindowEvent e) {;}
        public void windowClosed(WindowEvent e) {;}
        public void windowClosing(WindowEvent e) {System.exit(0);}
        public void windowDeactivated(WindowEvent e) {;}
        public void windowDeiconified(WindowEvent e) {;}
        public void windowIconified(WindowEvent e) {;}
        public void windowOpened(WindowEvent e) {;}
}
```

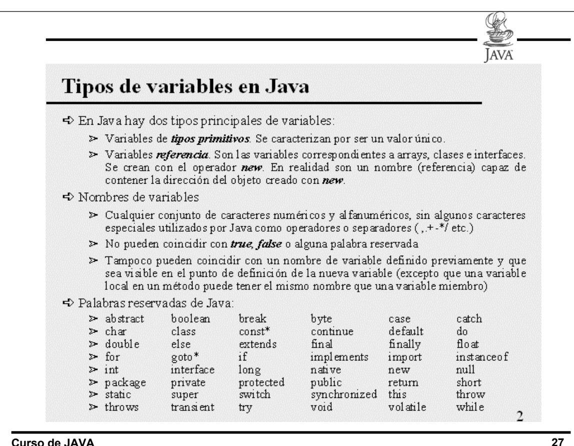

Introducción: Ejemplo1.java

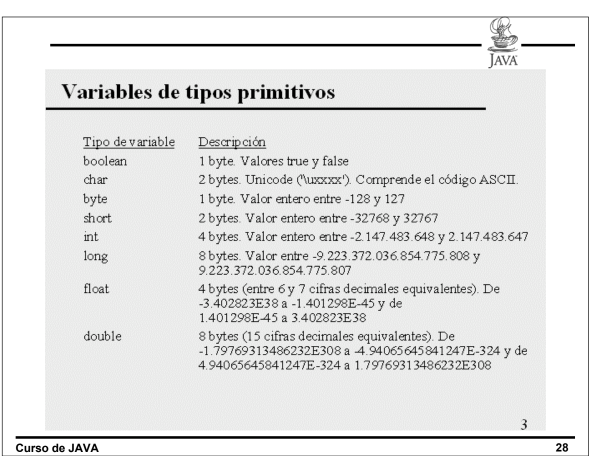
```
// fichero Ejemplo1.java
import java.util.Vector;
import java.awt.*;

class Ejemplo1 {
   public static void main(String arg[]) throws InterruptedException
      {
         System.out.println("Comienza main()...");
         Circulo c = new Circulo(2.0, 2.0, 4.0);
         System.out.println("Radio = " + c.r + " unidades.");
         System.out.println("Centro = (" + c.x + "," + c.y + ") unidades.");
         Circulo c1 = new Circulo(1.0, 1.0, 2.0);
         Circulo c2 = new Circulo(0.0, 0.0, 3.0);
         c = c1.elMayor(c2);
         System.out.println("El mayor radio es " + c.r + ".");
         c = new Circulo();   //  c.r = 0.0;
         c = Circulo.elMayor(c1, c2);
         System.out.println("El mayor radio es " + c.r + ".");
         System.out.println("Termina main()...");
...
```

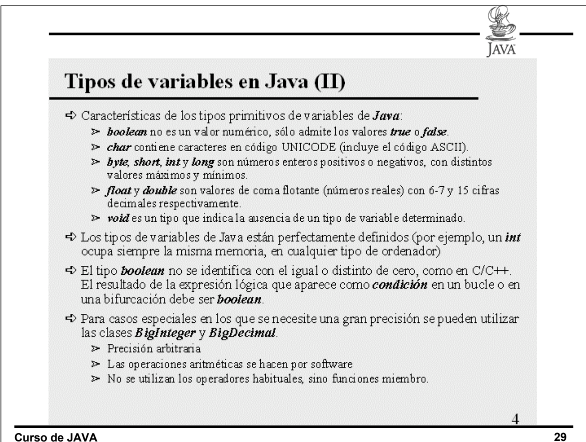
Continuación...

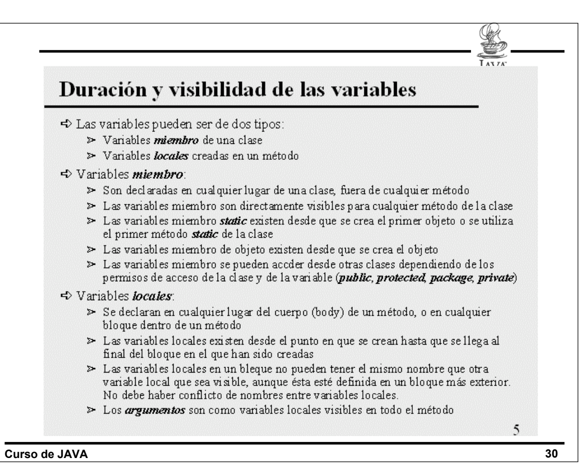

JAVA
Introducción: Ejemplo1.java (cont)
VentanaCerrable ventana = new VentanaCerrable("Ventana abierta al mundo...");
Vector v=new Vector();
CirculoGrafico cg1 = new CirculoGrafico(200, 200, 100, Color.red);
CirculoGrafico cg2 = new CirculoGrafico(300, 200, 100, Color.blue);
RectanguloGrafico rg = new RectanguloGrafico(50, 50, 450, 350, Color.green);
v.addElement(cg1);
v.addElement(cg2);
v.addElement(rg);
PanelDibujo mipanel = new PanelDibujo(v);
ventana.add(mipanel);
ventana.setSize(500, 400);
ventana.setVisible(true);
} // fin de main()
} // fin de class Ejemplo1...
Curso de JAVA
25

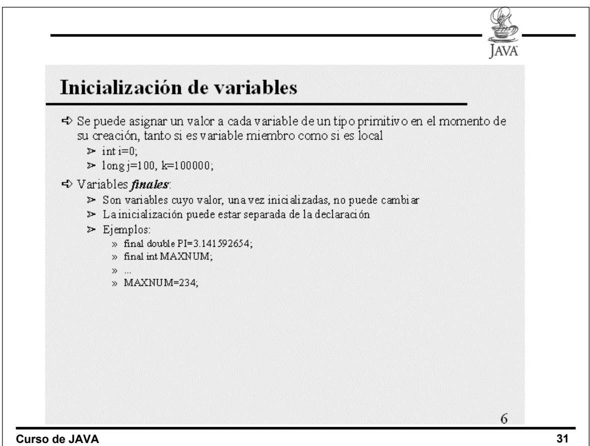

JAVA
Introducción: Ejemplo1
Ventana abierta al mundo...
Curso de JAVA
26

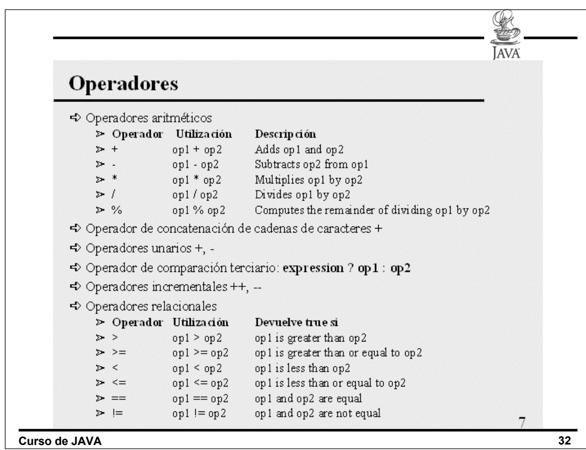

JAVA
Tipos de variables en Java
En Java hay dos tipos principales de variables:
Variables de tipos primitivos. Se caracterizan por ser un valor único.
Variables referencia. Son las variables correspondientes a arrays, clases e interfaces. Se crean con el operador new. En realidad son un nombre (referencia) capaz de contener la dirección del objeto creado con new.
Nombres de variables
Cualquier conjunto de caracteres numéricos y alfanuméricos, sin algunos caracteres especiales utilizados por Java como operadores o separadores (,.+-*/ etc.)
No pueden coincidir con true, false o alguna palabra reservada
Tampoco pueden coincidir con un nombre de variable definido previamente y que sea visible en el punto de definición de la nueva variable (excepto que una variable local en un método puede tener el mismo nombre que una variable miembro)
Palabras reservadas de Java:
abstract boolean break byte case catch
char class const* continue default do
double else extends final finally float
for goto* if implements import instanceof
int interface long native new null
package private protected public return short
static super switch synchronized this throw
throws transient try void volatile while
2
Curso de JAVA
27

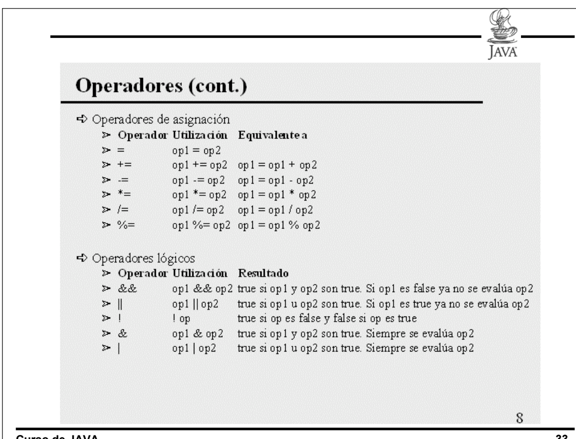

JAVA
Variables de tipos primitivos
Tipo de variable Descripción
boolean 1 byte. Valores true y false
char 2 bytes. Unicode ('\uxxxx'). Comprende el código ASCII.
byte 1 byte. Valor entero entre -128 y 127
short 2 bytes. Valor entero entre -32768 y 32767
int 4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647
long 8 bytes. Valor entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807
float 4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38
double 8 bytes (15 cifras decimales equivalentes). De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308
3
Curso de JAVA
28

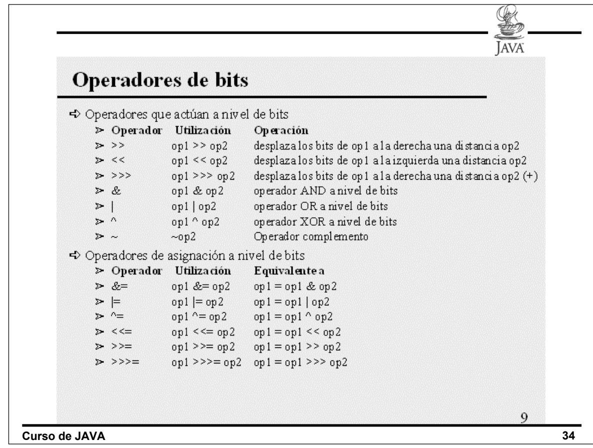

JAVA
Tipos de variables en Java (II)
Características de los tipos primitivos de variables de Java:
boolean no es un valor numérico, sólo admite los valores true o false.
char contiene caracteres en código UNICODE (incluye el código ASCII).
byte, short, int y long son números enteros positivos o negativos, con distintos valores máximos y mínimos.
float y double son valores de coma flotante (números reales) con 6-7 y 15 cifras decimales respectivamente.
void es un tipo que indica la ausencia de un tipo de variable determinado.
Los tipos de variables de Java están perfectamente definidos (por ejemplo, un int ocupa siempre la misma memoria, en cualquier tipo de ordenador)
El tipo boolean no se identifica con el igual o distinto de cero, como en C/C++. El resultado de la expresión lógica que aparece como condición en un bucle o en una bifurcación debe ser boolean.
Para casos especiales en los que se necesite una gran precisión se pueden utilizar las clases BigInteger y BigDecimal.
Precisión arbitraria
Las operaciones aritméticas se hacen por software
No se utilizan los operadores habituales, sino funciones miembro.
4
Curso de JAVA
29

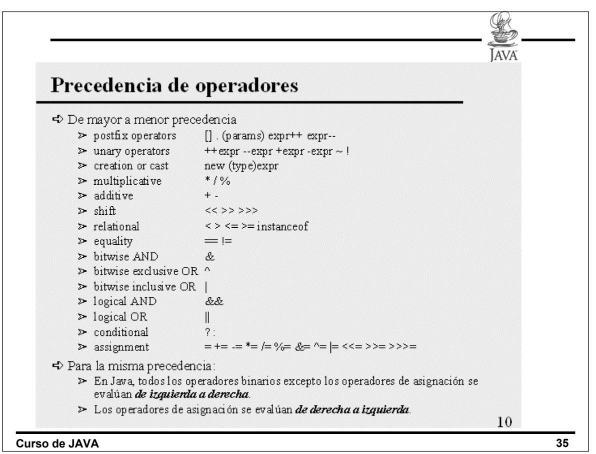

Duración y visibilidad de las variables
Las variables pueden ser de dos tipos:
Variables miembro de una clase
Variables locales creadas en un método
Variables miembro:
Son declaradas en cualquier lugar de una clase, fuera de cualquier método
Las variables miembro son directamente visibles para cualquier método de la clase
Las variables miembro static existen desde que se crea el primer objeto o se utiliza el primer método static de la clase
Las variables miembro de objeto existen desde que se crea el objeto
Las variables miembro se pueden accder desde otras clases dependiendo de los permisos de acceso de la clase y de la variable (public, protected, package, private)
Variables locales:
Se declaran en cualquier lugar del cuerpo (body) de un método, o en cualquier bloque dentro de un método
Las variables locales existen desde el punto en que se crean hasta que se llega al final del bloque en el que han sido creadas
Las variables locales en un bleque no pueden tener el mismo nombre que otra variable local que sea visible, aunque ésta esté definida en un bloque más exterior. No debe haber conflicto de nombres entre variables locales.
Los argumentos son como variables locales visibles en todo el método
5
Curso de JAVA
30











Control de ejecución: bifurcaciones

- Bifurcación ***if***:

```
if (booleanExpression) {
    statements;
}
```

- Bifurcación ***if else***:

```
if (booleanExpression) {
    statements1;
} else {
    statements2;
}
```

- Bifurcación ***if elseif else***:

```
if (booleanExpression1) {
    statements1;
} else if (booleanExpression2) {
    statements2;
} else if (booleanExpression3) {
    statements3;
} else {
    statements4;
}
```

11

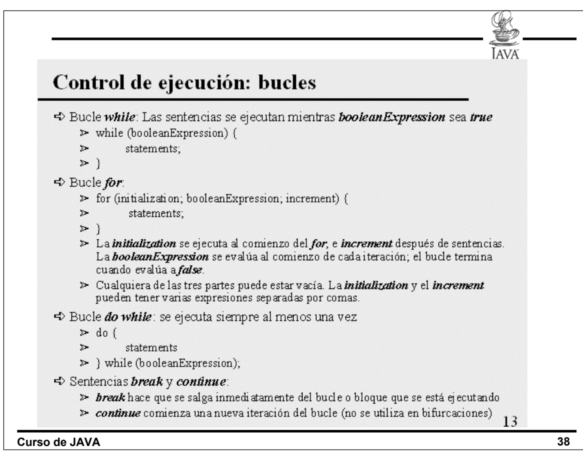
Control de ejecución: bifurcaciones (cont.)

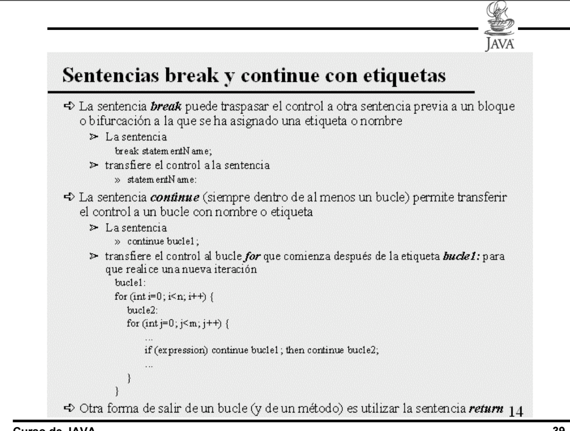
- Sentencia ***switch***:

```
switch (expression) {
        case value1:  statements1; break;
        case value2:  statements2; break;
        case value3:  statements3; break;
        case value4:  statements4; break;
        case value5:  statements5; break;
        case value6:  statements6; break;
        [default:: statements7;]
}
```

- Características:
  - Cada sentencia ***case*** se corresponde con un único valor de ***expression***. No se pueden establecer rangos o condiciones
  - Si una sentencia ***case*** no lleva una sentencia ***break*** detrás, el control pasa a la sentencia ***case*** siguiente, aunque su valor no se corresponda con ***expression***
  - Los valores no comprendidos en ninguna sentencia ***case*** se gestionan en ***default***, que es opcional.
  - En ausencia de ***break*** cuando se ejecuta una sentencia ***case*** se ejecutan también todas las que van a continuación, hasta que se llega a un ***break*** o hasta que se termina el ***switch***

12





Bloque try {...} catch {...} finally {...}

- Detección y tratamiento "in situ" de ***excepciones***: bloques **try {...} catch {...} finally {...}**

```
try {
    // Código que puede lanzar la excepción MyException
    MyException me = new MyException("MyException message");
    throw me;
} catch (MyException e1) {
    // Ocuparse de MyException simplemente dando aviso
    System.out.println(e1.getMessage());
} catch (My Other Exception moe) {
    ...
} [finally {
    // Sentencias que se ejecutarán en cualquier caso
   ...
}]
```

- El código dentro del bloque ***try*** está "vigilado".
  - Si se produce una situación anormal se lanza una excepción del tipo **MyException**
  - El control pasa al bloque ***catch*** que decide lo que hay que hacer
  - Finalmente se ejecuta el bloque ***finally***, que es opcional, pero que si está presente se ejecuta siempre, sea cual sea el tipo de error

15

Uso de variables. Funciones

- Ejemplos de ***variables***:

```
int a;          // Define una variable que aún no tiene valor.
int b = 3;      // Define una variable y la inicializa.
a = b;          // Asigna el valor de b a a.
int c = a+b;    // Se puede operar con ellas
```

- Una ***función*** o ***método*** es una porción de código que realiza una tarea específica, separable del resto. Suele tomar uno o más ***argumentos*** y devolver un ***valor de retorno***.

```
int sumar(int num1, int num2) {
    int num3;
    num3 = num1+num2;
    return num3;
}
```

- Así la 4ª línea se puede cambiar por unas llamada a la función ***sumar()***:

```
int c = sumar(a,b);
```

16

Concepto de "clase" en Java

- Una *clase* es una agrupación de datos (***variables***) y de funciones que operan sobre esos datos (***métodos***).

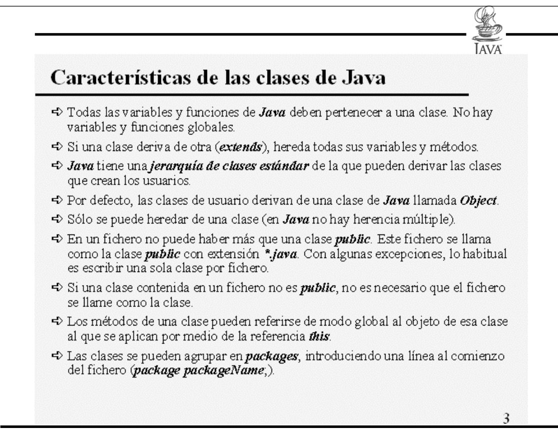
```
[public] class Classname {
    // definición de variables y métodos
    ...
}
```

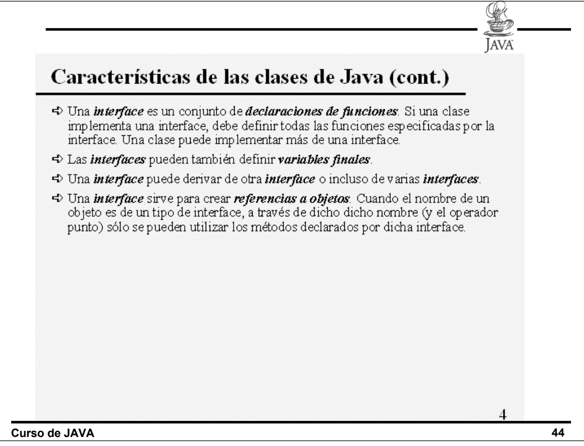
- Un ***objeto*** es un ejemplar concreto de una clase. Las ***clases*** son como tipos de variables, mientras que los ***objetos*** son como variables concretas de un tipo determinado.

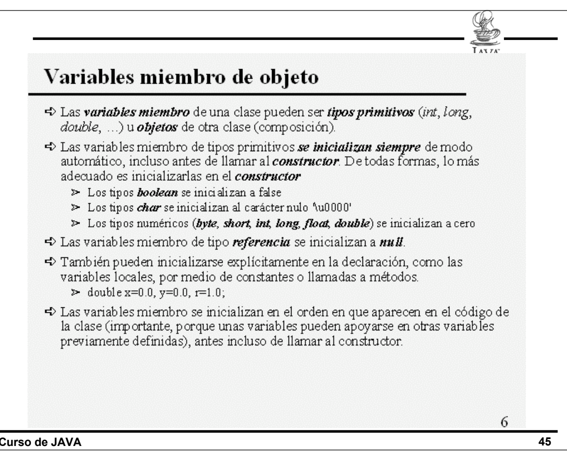
```
Classname unObjeto;
Classname otroObjeto;
```

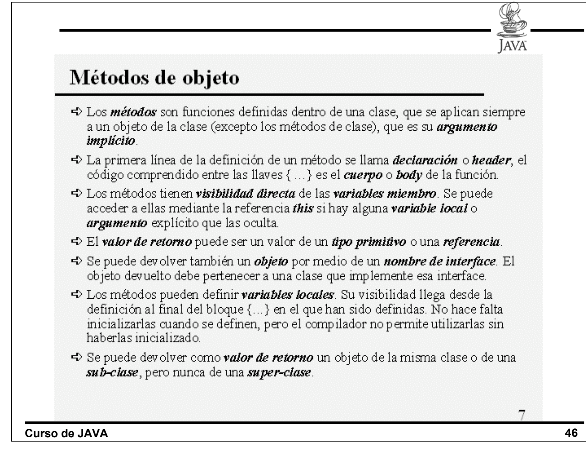
- Algunos **conceptos importantes** en POO:
  - **Encapsulación**. Las clases pueden ser declaradas como ***públicas*** y como ***package*** (visibles sólo para las clases del mismo ***package***). Las variables y métodos pueden ser declaradas ***public***, ***private***, ***protected*** y ***package***. De esta forma se puede controlar el acceso y evitar un uso inadecuado.
  - **Herencia**. Una clase puede derivar de otra (***extends***), y en ese caso hereda todas sus variables y métodos. Una clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados.
  - **Polimorfismo**. Los objetos de distintas clases pertenecientes a una misma jerarquía pueden tratarse de una forma general e individualizada, al mismo tiempo.

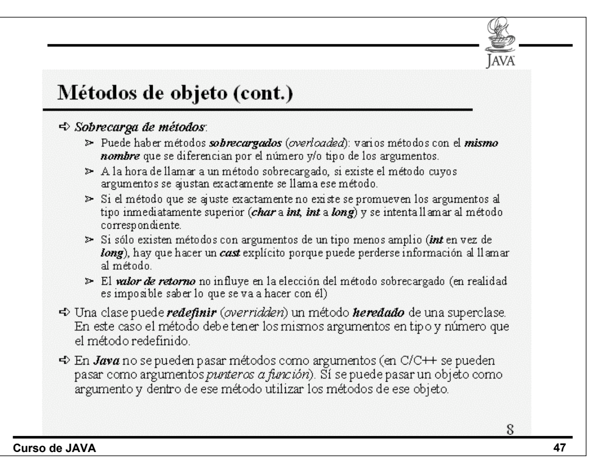
2

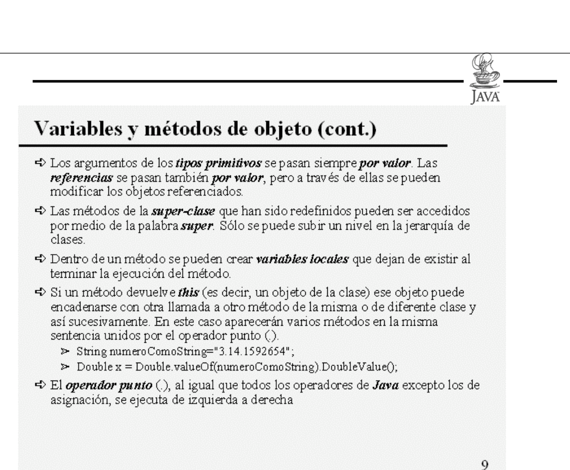

JAVA
Características de las clases de Java
Todas las variables y funciones de Java deben pertenecer a una clase. No hay variables y funciones globales.
Si una clase deriva de otra (extends), hereda todas sus variables y métodos.
Java tiene una jerarquía de clases estándar de la que pueden derivar las clases que crean los usuarios.
Por defecto, las clases de usuario derivan de una clase de Java llamada Object.
Sólo se puede heredar de una clase (en Java no hay herencia múltiple).
En un fichero no puede haber más que una clase public. Este fichero se llama como la clase public con extensión *.java. Con algunas excepciones, lo habitual es escribir una sola clase por fichero.
Si una clase contenida en un fichero no es public, no es necesario que el fichero se llame como la clase.
Los métodos de una clase pueden referirse de modo global al objeto de esa clase al que se aplican por medio de la referencia this.
Las clases se pueden agrupar en packages, introduciendo una línea al comienzo del fichero (package packageName;).
3
Curso de JAVA
43

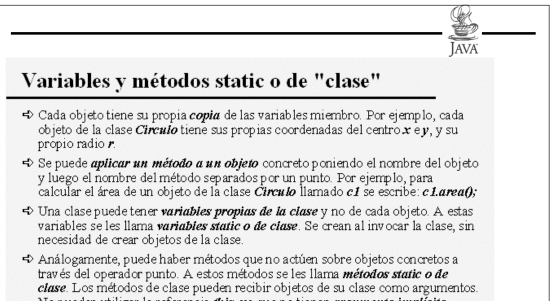

JAVA
Características de las clases de Java (cont.)
Una interface es un conjunto de declaraciones de funciones. Si una clase implementa una interface, debe definir todas las funciones especificadas por la interface. Una clase puede implementar más de una interface.
Las interfaces pueden también definir variables finales.
Una interface puede derivar de otra interface o incluso de varias interfaces.
Una interface sirve para crear referencias a objetos. Cuando el nombre de un objeto es de un tipo de interface, a través de dicho dicho nombre (y el operador punto) sólo se pueden utilizar los métodos declarados por dicha interface.
4
Curso de JAVA
44

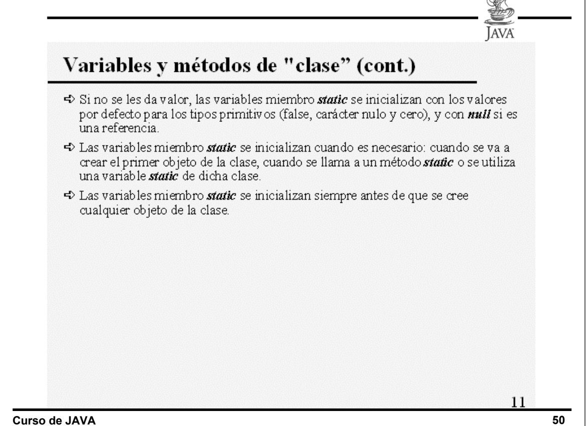

Variables miembro de objeto
Las variables miembro de una clase pueden ser tipos primitivos (int, long, double, …) u objetos de otra clase (composición).
Las variables miembro de tipos primitivos se inicializan siempre de modo automático, incluso antes de llamar al constructor. De todas formas, lo más adecuado es inicializarlas en el constructor
Los tipos boolean se inicializan a false
Los tipos char se inicializan al carácter nulo '\u0000'
Los tipos numéricos (byte, short, int, long, float, double) se inicializan a cero
Las variables miembro de tipo referencia se inicializan a null.
También pueden inicializarse explícitamente en la declaración, como las variables locales, por medio de constantes o llamadas a métodos.
double x=0.0, y=0.0, r=1.0;
Las variables miembro se inicializan en el orden en que aparecen en el código de la clase (importante, porque unas variables pueden apoyarse en otras variables previamente definidas), antes incluso de llamar al constructor.
6
Curso de JAVA
45

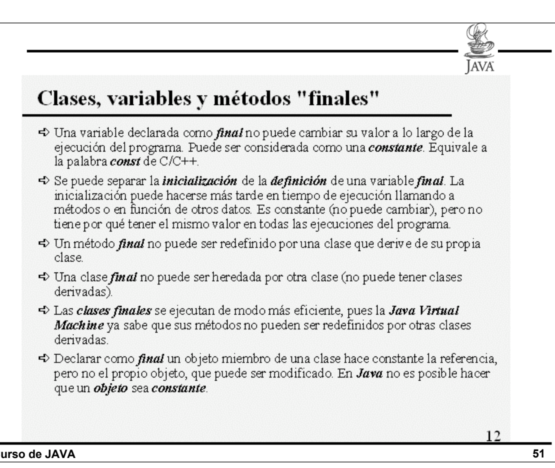

JAVA
Métodos de objeto
Los métodos son funciones definidas dentro de una clase, que se aplican siempre a un objeto de la clase (excepto los métodos de clase), que es su argumento implícito.
La primera línea de la definición de un método se llama declaración o header, el código comprendido entre las llaves {…} es el cuerpo o body de la función.
Los métodos tienen visibilidad directa de las variables miembro. Se puede acceder a ellas mediante la referencia this si hay alguna variable local o argumento explícito que las oculta.
El valor de retorno puede ser un valor de un tipo primitivo o una referencia.
Se puede devolver también un objeto por medio de un nombre de interface. El objeto devuelto debe pertenecer a una clase que implemente esa interface.
Los métodos pueden definir variables locales. Su visibilidad llega desde la definición al final del bloque {…} en el que han sido definidas. No hace falta inicializarlas cuando se definen, pero el compilador no permite utilizarlas sin haberlas inicializado.
Se puede devolver como valor de retorno un objeto de la misma clase o de una sub-clase, pero nunca de una super-clase.
7
Curso de JAVA
46

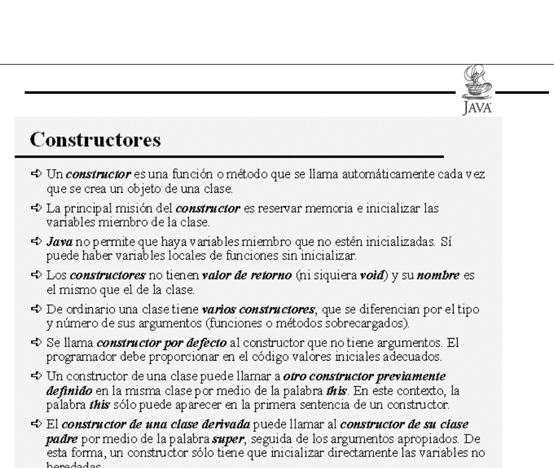

JAVA
Métodos de objeto (cont.)
Sobrecarga de métodos:
Puede haber métodos sobrecargados (overloaded): varios métodos con el mismo nombre que se diferencian por el número y/o tipo de los argumentos.
A la hora de llamar a un método sobrecargado, si existe el método cuyos argumentos se ajustan exactamente se llama ese método.
Si el método que se ajuste exactamente no existe se promueven los argumentos al tipo inmediatamente superior (char a int, int a long) y se intenta llamar al método correspondiente.
Si sólo existen métodos con argumentos de un tipo menos amplio (int en vez de long), hay que hacer un cast explícito porque puede perderse información al llamar al método.
El valor de retorno no influye en la elección del método sobrecargado (en realidad es imposible saber lo que se va a hacer con él)
Una clase puede redefinir (overridden) un método heredado de una superclase. En este caso el método debe tener los mismos argumentos en tipo y número que el método redefinido.
En Java no se pueden pasar métodos como argumentos (en C/C++ se pueden pasar como argumentos punteros a función). Sí se puede pasar un objeto como argumento y dentro de ese método utilizar los métodos de ese objeto.
8
Curso de JAVA
47

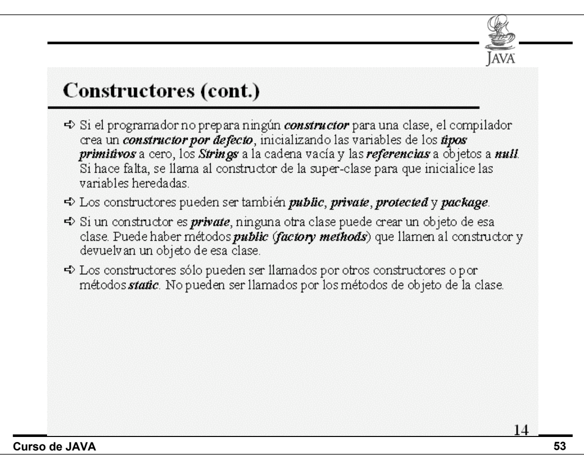

JAVA
Variables y métodos de objeto (cont.)
Los argumentos de los tipos primitivos se pasan siempre por valor. Las referencias se pasan también por valor, pero a través de ellas se pueden modificar los objetos referenciados.
Las métodos de la super-clase que han sido redefinidos pueden ser accedidos por medio de la palabra super. Sólo se puede subir un nivel en la jerarquía de clases.
Dentro de un método se pueden crear variables locales que dejan de existir al terminar la ejecución del método.
Si un método devuelve this (es decir, un objeto de la clase) ese objeto puede encadenarse con otra llamada a otro método de la misma o de diferente clase y así sucesivamente. En este caso aparecerán varios métodos en la misma sentencia unidos por el operador punto (.).
String numeroComoString="3.14.1592654";
Double x = Double.valueOf(numeroComoString).DoubleValue();
El operador punto (.), al igual que todos los operadores de Java excepto los de asignación, se ejecuta de izquierda a derecha
9
Curso de JAVA
48

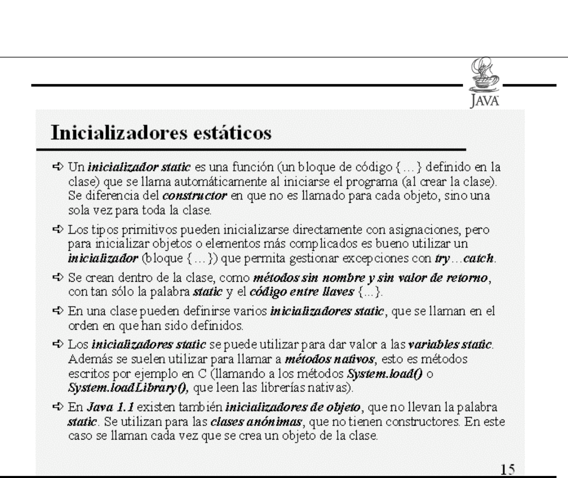

JAVA
Variables y métodos static o de "clase"
Cada objeto tiene su propia copia de las variables miembro. Por ejemplo, cada objeto de la clase Circulo tiene sus propias coordenadas del centro x e y, y su propio radio r.
Se puede aplicar un método a un objeto concreto poniendo el nombre del objeto y luego el nombre del método separados por un punto. Por ejemplo, para calcular el área de un objeto de la clase Circulo llamado c1 se escribe: c1.area();
Una clase puede tener variables propias de la clase y no de cada objeto. A estas variables se les llama variables static o de clase. Se crean al invocar la clase, sin necesidad de crear objetos de la clase.
Análogamente, puede haber métodos que no actúen sobre objetos concretos a través del operador punto. A estos métodos se les llama métodos static o de clase. Los métodos de clase pueden recibir objetos de su clase como argumentos. No pueden utilizar la referencia this, ya que no tienen argumento implícito.
Los métodos y las variables de clase son lo más parecido que Java tiene a las funciones y variables globales de C/C++.
Los métodos y variables de clase se crean anteponiendo la palabra static. Para llamarlos se suele utilizar el nombre de la clase (no es imprescindible), en vez del nombre de un objeto de la clase (por ejemplo, Circulo.numCirculos puede ser una variable de clase que cuente el número de círculos creados).
10
Curso de JAVA
49

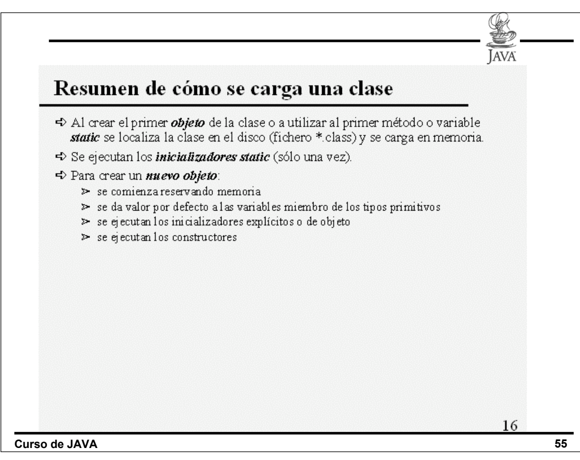

JAVA
Variables y métodos de "clase" (cont.)
Si no se les da valor, las variables miembro static se inicializan con los valores por defecto para los tipos primitivos (false, carácter nulo y cero), y con null si es una referencia.
Las variables miembro static se inicializan cuando es necesario: cuando se va a crear el primer objeto de la clase, cuando se llama a un método static o se utiliza una variable static de dicha clase.
Las variables miembro static se inicializan siempre antes de que se cree cualquier objeto de la clase.
11
Curso de JAVA
50

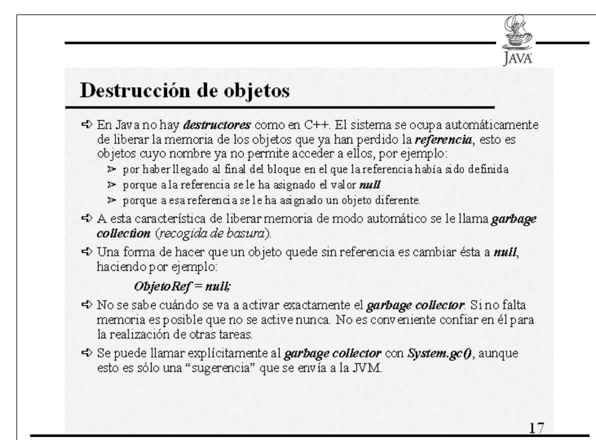

JAVA
Clases, variables y métodos "finales"
Una variable declarada como final no puede cambiar su valor a lo largo de la ejecución del programa. Puede ser considerada como una constante. Equivale a la palabra const de C/C++.
Se puede separar la inicialización de la definición de una variable final. La inicialización puede hacerse más tarde en tiempo de ejecución llamando a métodos o en función de otros datos. Es constante (no puede cambiar), pero no tiene por qué tener el mismo valor en todas las ejecuciones del programa.
Un método final no puede ser redefinido por una clase que derive de su propia clase.
Una clase final no puede ser heredada por otra clase (no puede tener clases derivadas).
Las clases finales se ejecutan de modo más eficiente, pues la Java Virtual Machine ya sabe que sus métodos no pueden ser redefinidos por otras clases derivadas.
Declarar como final un objeto miembro de una clase hace constante la referencia, pero no el propio objeto, que puede ser modificado. En Java no es posible hacer que un objeto sea constante.
12
Curso de JAVA
51


JAVA
Constructores
Un constructor es una función o método que se llama automáticamente cada vez que se crea un objeto de una clase.
La principal misión del constructor es reservar memoria e inicializar las variables miembro de la clase.
Java no permite que haya variables miembro que no estén inicializadas. Sí puede haber variables locales de funciones sin inicializar.
Los constructores no tienen valor de retorno (ni siquiera void) y su nombre es el mismo que el de la clase.
De ordinario una clase tiene varios constructores, que se diferencian por el tipo y número de sus argumentos (funciones o métodos sobrecargados).
Se llama constructor por defecto al constructor que no tiene argumentos. El programador debe proporcionar en el código valores iniciales adecuados.
Un constructor de una clase puede llamar a otro constructor previamente definido en la misma clase por medio de la palabra this. En este contexto, la palabra this sólo puede aparecer en la primera sentencia de un constructor.
El constructor de una clase derivada puede llamar al constructor de su clase padre por medio de la palabra super, seguida de los argumentos apropiados. De esta forma, un constructor sólo tiene que inicializar directamente las variables no heredadas.
13
Curso de JAVA
52

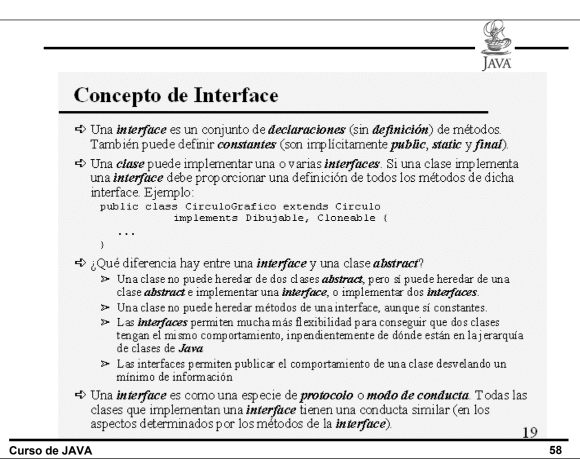

JAVA
Constructores (cont.)
Si el programador no prepara ningún constructor para una clase, el compilador crea un constructor por defecto, inicializando las variables de los tipos primitivos a cero, los Strings a la cadena vacía y las referencias a objetos a null. Si hace falta, se llama al constructor de la super-clase para que inicialice las variables heredadas.
Los constructores pueden ser también public, private, protected y package.
Si un constructor es private, ninguna otra clase puede crear un objeto de esa clase. Puede haber métodos public (factory methods) que llamen al constructor y devuelvan un objeto de esa clase.
Los constructores sólo pueden ser llamados por otros constructores o por métodos static. No pueden ser llamados por los métodos de objeto de la clase.
14
Curso de JAVA
53

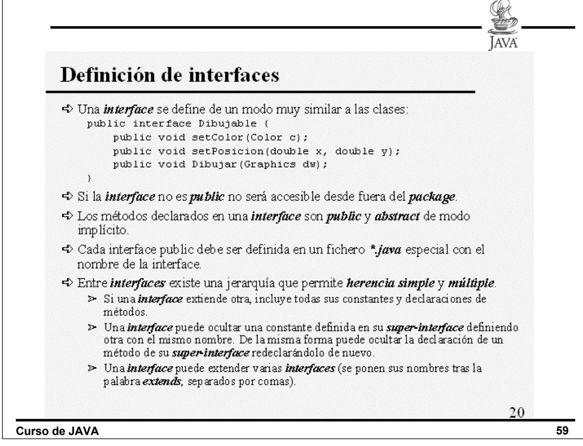

JAVA
Inicializadores estáticos
Un inicializador static es una función (un bloque de código { ... } definido en la clase) que se llama automáticamente al iniciarse el programa (al crear la clase). Se diferencia del constructor en que no es llamado para cada objeto, sino una sola vez para toda la clase.
Los tipos primitivos pueden inicializarse directamente con asignaciones, pero para inicializar objetos o elementos más complicados es bueno utilizar un inicializador (bloque { ... }) que permita gestionar excepciones con try…catch.
Se crean dentro de la clase, como métodos sin nombre y sin valor de retorno, con tan sólo la palabra static y el código entre llaves {...}.
En una clase pueden definirse varios inicializadores static, que se llaman en el orden en que han sido definidos.
Los inicializadores static se puede utilizar para dar valor a las variables static. Además se suelen utilizar para llamar a métodos nativos, esto es métodos escritos por ejemplo en C (llamando a los métodos System.load() o System.loadLibrary(), que leen las librerías nativas).
En Java 1.1 existen también inicializadores de objeto, que no llevan la palabra static. Se utilizan para las clases anónimas, que no tienen constructores. En este caso se llaman cada vez que se crea un objeto de la clase.
15
Curso de JAVA
54

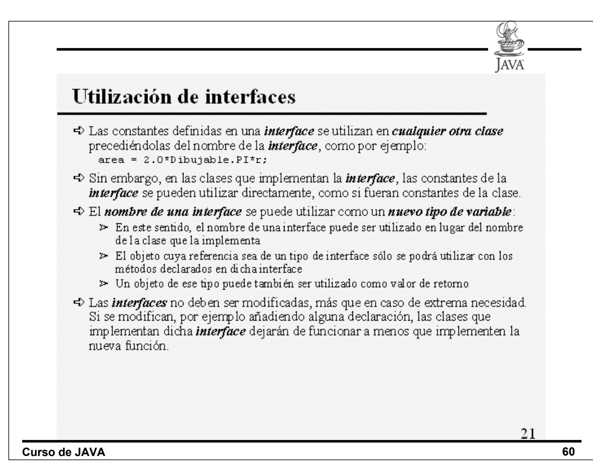

Java
Resumen de cómo se carga una clase
Al crear el primer objeto de la clase o a utilizar al primer método o variable static se localiza la clase en el disco (fichero *.class) y se carga en memoria.
Se ejecutan los inicializadores static (sólo una vez).
Para crear un nuevo objeto:
se comienza reservando memoria
se da valor por defecto a las variables miembro de los tipos primitivos
se ejecutan los inicializadores explícitos o de objeto
se ejecutan los constructores
16
Curso de JAVA
55

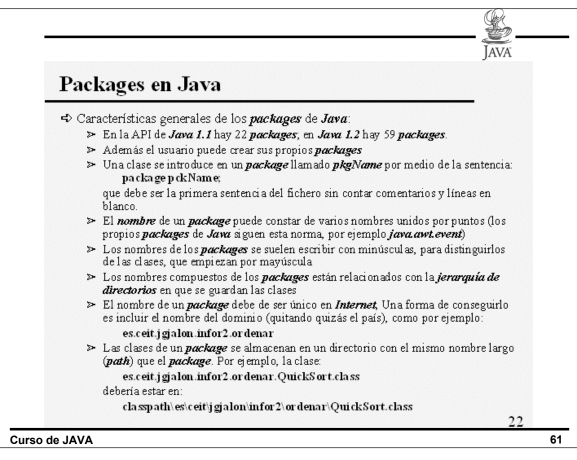

Java
Destrucción de objetos
En Java no hay destructores como en C++. El sistema se ocupa automáticamente de liberar la memoria de los objetos que ya han perdido la referencia, esto es objetos cuyo nombre ya no permite acceder a ellos, por ejemplo:
por haber llegado al final del bloque en el que la referencia había sido definida
porque a la referencia se le ha asignado el valor null
porque a esa referencia se le ha asignado un objeto diferente.
A esta característica de liberar memoria de modo automático se le llama garbage collection (recogida de basura).
Una forma de hacer que un objeto quede sin referencia es cambiar ésta a null, haciendo por ejemplo:
ObjetoRef = null;
No se sabe cuándo se va a activar exactamente el garbage collector. Si no falta memoria es posible que no se active nunca. No es conveniente confiar en él para la realización de otras tareas.
Se puede llamar explícitamente al garbage collector con System.gc(), aunque esto es sólo una "sugerencia" que se envía a la JVM.
17
Curso de JAVA
56

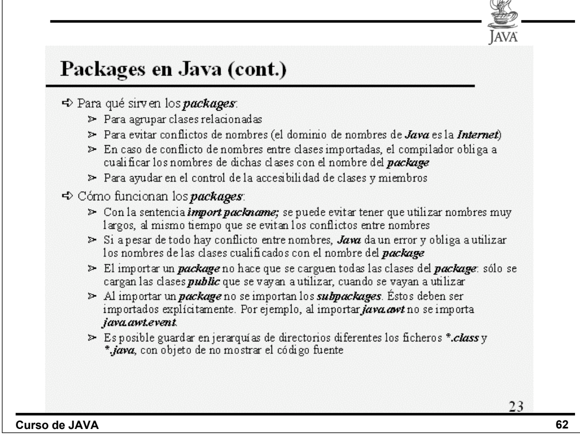
Java
Finalizadores
Un finalizador es un método que se llama automáticamente al destruir un objeto (antes de que la memoria sea liberada de modo automático por el sistema).
Se utilizan para ciertas operaciones de terminación distintas de liberar memoria (cerrar ficheros, cerrar conexiones, liberar memoria reservada por funciones nativas, etc.)
Un finalizador es un método de objeto (no static), sin valor de retorno (void), sin argumentos y que siempre se llama finalize().
Los finalizadores se llaman de modo automático siempre que estén definidos. Un finalizador debería terminar llamando al finalizador de su superclase.
No se puede saber el momento preciso en que los finalizadores van a ser llamados. En muchas ocasiones será conveniente que el programador realice de modo explícito esas operaciones de finalización.
El garbage collector sólo libera la memoria reservada con new. Si por ejemplo se ha reservado memoria con funciones nativas en C (utilizando malloc()) esta memoria hay que liberarla explícitamente con el método finalize().
El método System.runFinalization() "sugiere" a la JVM que ejecute los finalizadores de los objetos pendientes (que han perdido la referencia).
Parece ser que hay que llamar primero a gc() y luego a runFinalization()
18
Curso de JAVA
57

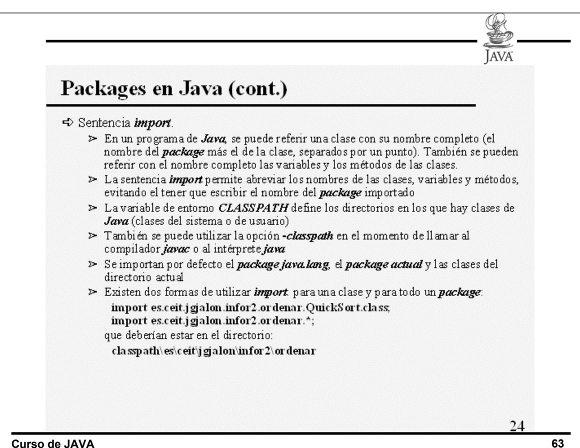

Java
Concepto de Interface
Una interface es un conjunto de declaraciones (sin definición) de métodos. También puede definir constantes (son implícitamente public, static y final).
Una clase puede implementar una o varias interfaces. Si una clase implementa una interface debe proporcionar una definición de todos los métodos de dicha interface. Ejemplo:
public class CirculoGrafico extends Circulo
implements Dibujable, Cloneable {
...
}
¿Qué diferencia hay entre una interface y una clase abstract?
Una clase no puede heredar de dos clases abstract, pero sí puede heredar de una clase abstract e implementar una interface, o implementar dos interfaces.
Una clase no puede heredar métodos de una interface, aunque sí constantes.
Las interfaces permiten mucha más flexibilidad para conseguir que dos clases tengan el mismo comportamiento, inpendientemente de dónde están en la jerarquía de clases de Java
Las interfaces permiten publicar el comportamiento de una clase desvelando un mínimo de información
Una interface es como una especie de protocolo o modo de conducta. Todas las clases que implementan una interface tienen una conducta similar (en los aspectos determinados por los métodos de la interface).
19
Curso de JAVA
58


Java
Definición de interfaces
Una interface se define de un modo muy similar a las clases:
public interface Dibujable {
public void setColor(Color c);
public void setPosicion(double x, double y);
public void Dibujar(Graphics dw);
}
Si la interface no es public no será accesible desde fuera del package.
Los métodos declarados en una interface son public y abstract de modo implícito.
Cada interface public debe ser definida en un fichero *.java especial con el nombre de la interface.
Entre interfaces existe una jerarquía que permite herencia simple y múltiple.
Si una interface extiende otra, incluye todas sus constantes y declaraciones de métodos.
Una interface puede ocultar una constante definida en su super-interface definiendo otra con el mismo nombre. De la misma forma puede ocultar la declaración de un método de su super-interface redeclarándolo de nuevo.
Una interface puede extender varias interfaces (se ponen sus nombres tras la palabra extends, separados por comas).
20
Curso de JAVA
59

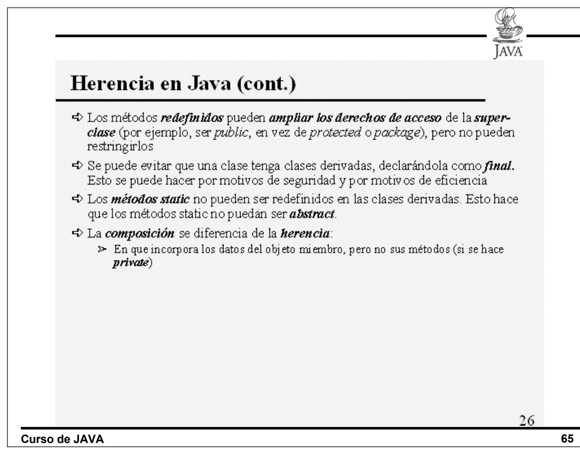

Java
Utilización de interfaces
Las constantes definidas en una interface se utilizan en cualquier otra clase precediéndolas del nombre de la interface, como por ejemplo:
area = 2.0*Dibujable.PI*r;
Sin embargo, en las clases que implementan la interface, las constantes de la interface se pueden utilizar directamente, como si fueran constantes de la clase.
El nombre de una interface se puede utilizar como un nuevo tipo de variable:
En este sentido, el nombre de una interface puede ser utilizado en lugar del nombre de la clase que la implementa
El objeto cuya referencia sea de un tipo de interface sólo se podrá utilizar con los métodos declarados en dicha interface
Un objeto de ese tipo puede también ser utilizado como valor de retorno
Las interfaces no deben ser modificadas, más que en caso de extrema necesidad. Si se modifican, por ejemplo añadiendo alguna declaración, las clases que implementan dicha interface dejarán de funcionar a menos que implementen la nueva función.
21
Curso de JAVA
60

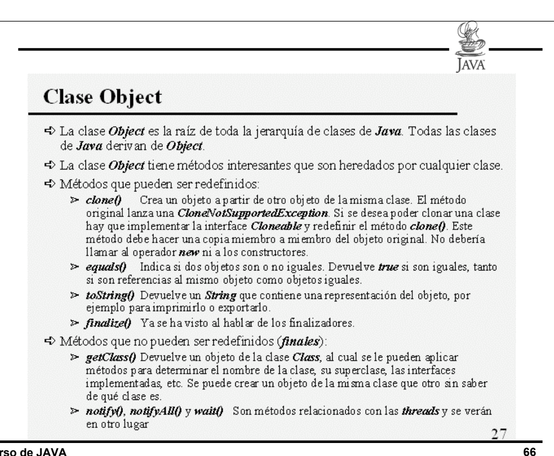

JAVA
Packages en Java
Características generales de los packages de Java:
En la API de Java 1.1 hay 22 packages; en Java 1.2 hay 59 packages.
Además el usuario puede crear sus propios packages
Una clase se introduce en un package llamado pkgName por medio de la sentencia:
package pckName;
que debe ser la primera sentencia del fichero sin contar comentarios y líneas en blanco.
El nombre de un package puede constar de varios nombres unidos por puntos (los propios packages de Java siguen esta norma, por ejemplo java.awt.event)
Los nombres de los packages se suelen escribir con minúsculas, para distinguirlos de las clases, que empiezan por mayúscula
Los nombres compuestos de los packages están relacionados con la jerarquía de directorios en que se guardan las clases
El nombre de un package debe de ser único en Internet, Una forma de conseguirlo es incluir el nombre del dominio (quitando quizás el país), como por ejemplo:
es.ceit.jgjalon.infor2.ordenar
Las clases de un package se almacenan en un directorio con el mismo nombre largo (path) que el package. Por ejemplo, la clase:
es.ceit.jgjalon.infor2.ordenar.QuickSort.class
debería estar en:
classpath\es\ceit\jgjalon\infor2\ordenar\QuickSort.class
22
Curso de JAVA
61

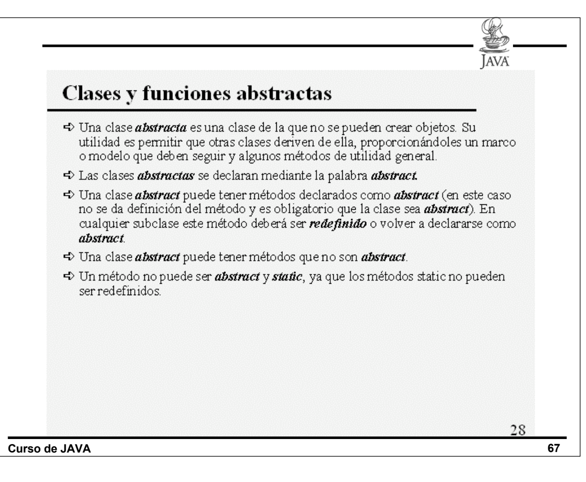

JAVA
Packages en Java (cont.)
Para qué sirven los packages:
Para agrupar clases relacionadas
Para evitar conflictos de nombres (el dominio de nombres de Java es la Internet)
En caso de conflicto de nombres entre clases importadas, el compilador obliga a cualificar los nombres de dichas clases con el nombre del package
Para ayudar en el control de la accesibilidad de clases y miembros
Cómo funcionan los packages:
Con la sentencia import packname; se puede evitar tener que utilizar nombres muy largos, al mismo tiempo que se evitan los conflictos entre nombres
Si a pesar de todo hay conflicto entre nombres, Java da un error y obliga a utilizar los nombres de las clases cualificados con el nombre del package
El importar un package no hace que se carguen todas las clases del package: sólo se cargan las clases public que se vayan a utilizar, cuando se vayan a utilizar
Al importar un package no se importan los subpackages. Éstos deben ser importados explícitamente. Por ejemplo, al importar java.awt no se importa java.awt.event.
Es posible guardar en jerarquías de directorios diferentes los ficheros *.class y *.java, con objeto de no mostrar el código fuente
23
Curso de JAVA
62

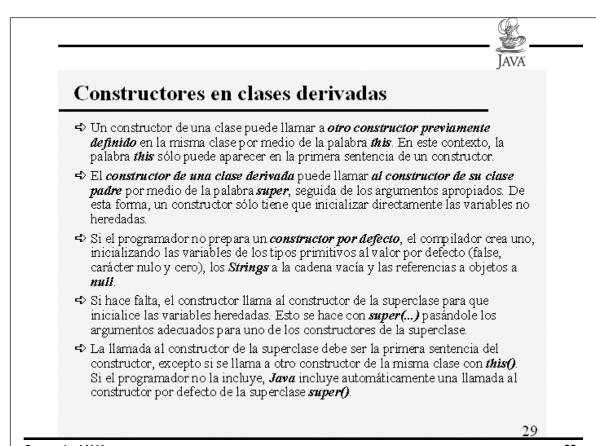

JAVA
Packages en Java (cont.)
Sentencia import.
En un programa de Java, se puede referir una clase con su nombre completo (el nombre del package más el de la clase, separados por un punto). También se pueden referir con el nombre completo las variables y los métodos de las clases.
La sentencia import permite abreviar los nombres de las clases, variables y métodos, evitando el tener que escribir el nombre del package importado
La variable de entorno CLASSPATH define los directorios en los que hay clases de Java (clases del sistema o de usuario)
También se puede utilizar la opción -classpath en el momento de llamar al compilador javac o al intérprete java
Se importan por defecto el package java.lang, el package actual y las clases del directorio actual
Existen dos formas de utilizar import: para una clase y para todo un package:
import es.ceit.jgjalon.infor2.ordenar.QuickSort.class;
import es.ceit.jgjalon.infor2.ordenar.*;
que deberían estar en el directorio:
classpath\es\ceit\jgjalon\infor2\ordenar
24
Curso de JAVA
63

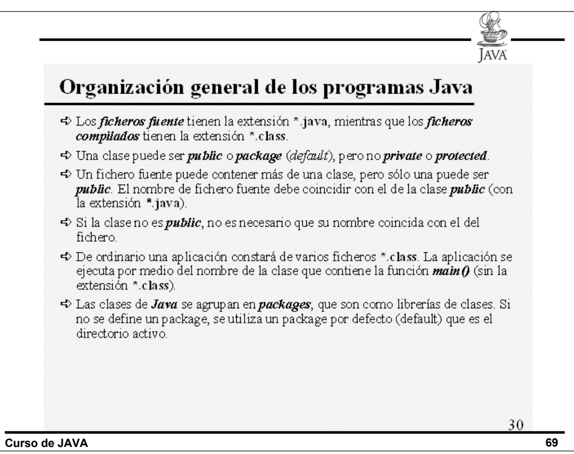
JAVA
Herencia en Java
Una clase puede construirse a partir de otra mediante el mecanismo de la herencia. Se indica que una clase deriva de otra mediante la palabra extends, como por ejemplo:
class CirculoGrafico extends Circulo {...}
Cuando una clase deriva de otra, hereda todas sus variables y métodos. Estas funciones y variables miembro pueden ser redefinidas (overridden) en la clase derivada, que puede también definir o añadir nuevas variables y métodos.
Es como si la sub-clase "contuviera" un objeto de la super-clase.
Java permite múltiples niveles de herencia, pero no permite que una clase derive de varias (no es posible la herencia múltiple).
En ocasiones se utiliza la terminología superclase para la clase padre y subclase para la clase derivada.
Todas las clases de Java creadas por el programador tienen una superclase. Si no se indica explícitamente con la palabra extends, las clases derivan de Object, que es la clase raíz de toda la jerarquía de clases de Java. Como consecuencia, todas las clases heredan algunos métodos de Object.
Se pueden crear tantas clases derivadas de una misma clase como se quiera, siempre y cuando una clase no herede de más de una clase.
25
Curso de JAVA
64

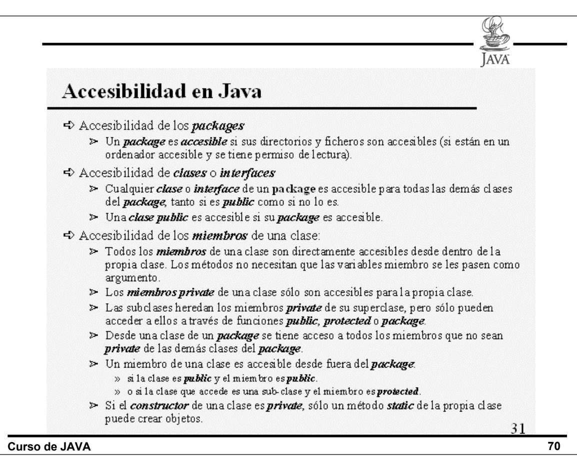

JAVA
Herencia en Java (cont.)
Los métodos redefinidos pueden ampliar los derechos de acceso de la super-clase (por ejemplo, ser public, en vez de protected o package), pero no pueden restringirlos
Se puede evitar que una clase tenga clases derivadas, declarándola como final. Esto se puede hacer por motivos de seguridad y por motivos de eficiencia
Los métodos static no pueden ser redefinidos en las clases derivadas. Esto hace que los métodos static no puedan ser abstract.
La composición se diferencia de la herencia:
En que incorpora los datos del objeto miembro, pero no sus métodos (si se hace private)
26
Curso de JAVA
65

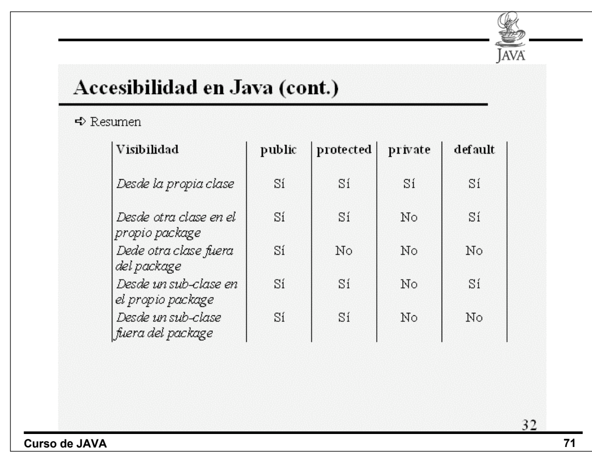

JAVA
Clase Object
La clase Object es la raíz de toda la jerarquía de clases de Java. Todas las clases de Java derivan de Object.
La clase Object tiene métodos interesantes que son heredados por cualquier clase.
Métodos que pueden ser redefinidos:
clone() Crea un objeto a partir de otro objeto de la misma clase. El método original lanza una CloneNotSupportedException. Si se desea poder clonar una clase hay que implementar la interface Cloneable y redefinir el método clone(). Este método debe hacer una copia miembro a miembro del objeto original. No debería llamar al operador new ni a los constructores.
equals() Indica si dos objetos son o no iguales. Devuelve true si son iguales, tanto si son referencias al mismo objeto como objetos iguales.
toString() Devuelve un String que contiene una representación del objeto, por ejemplo para imprimirlo o exportarlo.
finalize() Ya se ha visto al hablar de los finalizadores.
Métodos que no pueden ser redefinidos (finales):
getClass() Devuelve un objeto de la clase Class, al cual se le pueden aplicar métodos para determinar el nombre de la clase, su superclase, las interfaces implementadas, etc. Se puede crear un objeto de la misma clase que otro sin saber de qué clase es.
notify(), notifyAll() y wait() Son métodos relacionados con las threads y se verán en otro lugar
27
Curso de JAVA
66



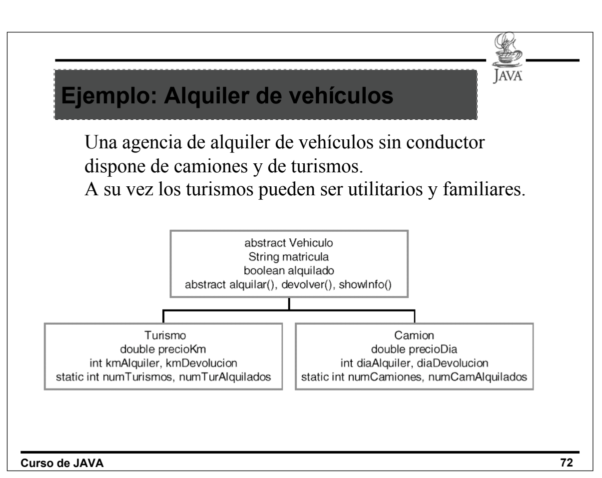








Visibilidad	public	protected	private	default
Desde la propia clase	Sí	Sí	Sí	Sí
Desde otra clase en el propio package	Sí	Sí	No	Sí
Dede otra clase fuera del package	Sí	No	No	No
Desde un sub-clase en el propio package	Sí	Sí	No	Sí
Desde un sub-clase fuera del package	Sí	Sí	No	No



Ejemplo: Vehiculo.java

```
// fichero Vehiculo.java

public abstract class Vehiculo {
        protected String matricula;
        protected boolean alquilado;

        public Vehiculo(String mat, boolean alq) {
                matricula=mat; alquilado=alq;
        }

        public abstract void alquilar(int dia);
        public abstract void devolver(int dia);

        public abstract void showInfo();

} // fin de clase Vehiculo
```


Ejemplo: Turismo.java

```
// fichero Turismo.java
public class Turismo extends Vehiculo {
        protected double precioKm;
        protected int kmAlquiler, kmDevolucion;
        protected static int numTurismos, numTurAlquilados;

        public Turismo(String mat, boolean alq, double prKm) {
                // llamada constructor de la superclase. Debe ser
                //    la primera sentencia del constructor
                super(mat, alq);
                precioKm = prKm;
                kmAlquiler = 0;
                kmDevolucion = 0;
                numTurismos++;
        }

        public void alquilar(int kmAlq) {
                if (alquilado == true)
                        System.out.println("El turismo " + matricula + " esta ya alquilado.");
                else {
                        kmAlquiler = kmAlq;
                        System.out.println("Se alquila un turismo con " + kmAlquiler + "km.");
                        numTurAlquilados++;
                        alquilado = true;
                }
        }
```


Ejemplo: Turismo.java II

```
        public void devolver(int kmDev) {
                if (alquilado == true) {
                        kmDevolucion = kmDev;
                        System.out.print("Se devuelve un turismo con " + kmDevolucion + "km. ");
                        System.out.println("Precio a abonar: " +
                                (kmDevolucion-kmAlquiler)*precioKm + " Euros");
                        numTurAlquilados--;
                        alquilado = false;
                } else
                        System.out.println("El turismo " + matricula + " no esta alquilado.");
        }

        public void showInfo() {
                System.out.println("\nDatos de un Turismo:");
                System.out.println("Matricula: " + matricula);
                System.out.println("Alquilado: " + ((alquilado==true)? "Si": "No"));
                System.out.println("Precio por km: " + precioKm + " Euros");
                System.out.println("Numero total de turismos: " + numTurismos);
                System.out.println("Numero de turismos alquilados: " + numTurAlquilados);
        }

} // fin de clase Turismo
```


Ejemplo: Camion.java

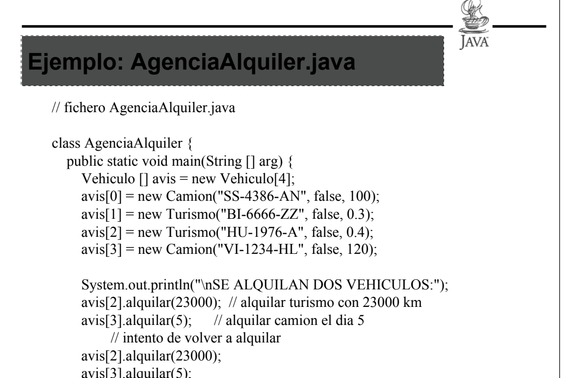
```
// fichero Camion.java

public class Camion extends Vehiculo {
        protected double precioDia;
        protected int diaAlquiler, diaDevolucion;
        protected static int numCamiones, numCamAlquilados;

        public Camion(String mat, boolean alq, double prD) {
                // llamada constructor de la superclase. Debe ser
                //    la primera sentencia del constructor
                super(mat, alq);
                precioDia = prD;
                diaAlquiler = 0;
                diaDevolucion = 0;
                numCamiones++;
        }
```


Ejemplo: Camion.java II

```
public void alquilar(int diaAlq) {
        if (alquilado==true) {
                System.out.println("El camion " + matricula + " esta ya alquilado.");
        } else {
                diaAlquiler = diaAlq;
                System.out.println("Se alquila un camion el dia " + diaAlquiler + ".");
                numCamAlquilados++;
                alquilado = true;
        }
}

public void devolver(int diaDev) {
        if (alquilado==true) {
                diaDevolucion = diaDev;
                System.out.print("Se devuelve un camion el dia " + diaDevolucion + ". ");
                System.out.println("Precio a abonar: " +
                        (diaDevolucion-diaAlquiler)*precioDia + " Euros");
                numCamAlquilados--;
                alquilado = false;
        } else
                System.out.println("El camion " + matricula + " no esta alquilado.");
}
```

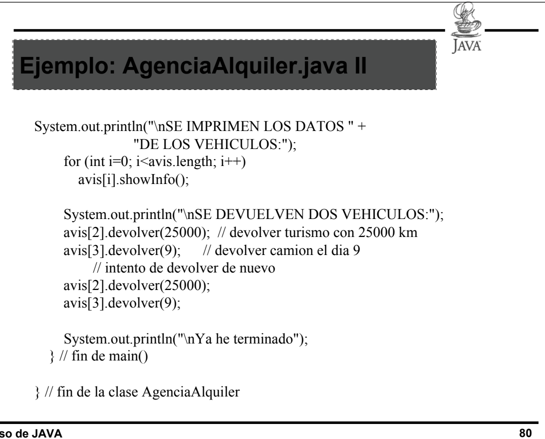

Ejemplo: Camion.java III

```
        public void showInfo() {
                System.out.println("\nDatos de un Camion:");
                System.out.println("Matricula: " + matricula);
                System.out.println("Alquilado: " + ((alquilado==true)? "Si": "No"));
                System.out.println("Precio por dia: " + precioDia + " Euros");
                System.out.println("Numero total de camiones: " + numCamiones);
                System.out.println("Numero de camiones alquilados: " + numCamAlquilados);
        }

} // fin de clase Camion
```



Ejemplo: AgenciaAlquiler.java

```
// fichero AgenciaAlquiler.java

class AgenciaAlquiler {
   public static void main(String [] arg) {
      Vehiculo [] avis = new Vehiculo[4];
      avis[0] = new Camion("SS-4386-AN", false, 100);
      avis[1] = new Turismo("BI-6666-ZZ", false, 0.3);
      avis[2] = new Turismo("HU-1976-A", false, 0.4);
      avis[3] = new Camion("VI-1234-HL", false, 120);

      System.out.println("\nSE ALQUILAN DOS VEHICULOS:");
      avis[2].alquilar(23000);  // alquilar turismo con 23000 km
      avis[3].alquilar(5);      // alquilar camion el dia 5
            // intento de volver a alquilar
      avis[2].alquilar(23000);
      avis[3].alquilar(5);
```

Ejemplo: AgenciaAlquiler.java II

```
System.out.println("\nSE IMPRIMEN LOS DATOS " +
                   "DE LOS VEHICULOS:");
      for (int i=0; i<avis.length; i++)
         avis[i].showInfo();

      System.out.println("\nSE DEVUELVEN DOS VEHICULOS:");
      avis[2].devolver(25000);  // devolver turismo con 25000 km
      avis[3].devolver(9);      // devolver camion el dia 9
            // intento de devolver de nuevo
      avis[2].devolver(25000);
      avis[3].devolver(9);

      System.out.println("\nYa he terminado");
   } // fin de main()

} // fin de la clase AgenciaAlquiler
```


Resultado Ejemplo 2

SE ALQUILAN DOS VEHICULOS:
Se alquila un turismo con 23000km.
Se alquila un camion el dia 5.
El turismo HU-1976-A esta ya alquilado.
El camion VI-1234-HL esta ya alquilado.
SE IMPRIMEN LOS DATOS DE LOS VEHICULOS:
Datos de un Camion:
Matricula: SS-4386-AN
Alquilado: No
Precio por dia: 100.0 Euros
Numero total de camiones: 2
Numero de camiones alquilados: 1
Datos de un Turismo:
Matricula: BI-6666-ZZ
Alquilado: No
Precio por km: 0.3 Euros
Numero total de turismos: 2
Numero de turismos alquilados: 1
Datos de un Turismo:
Matricula: HU-1976-A
Alquilado: Si
Precio por km: 0.4 Euros
Numero total de turismos: 2
Numero de turismos alquilados: 1
Datos de un Camion:
Matricula: VI-1234-HL
Alquilado: Si
Precio por dia: 120.0 Euros
Numero total de camiones: 2
Numero de camiones alquilados: 1
SE DEVUELVEN DOS VEHICULOS:
Se devuelve un turismo con 25000km. Precio a abonar: 800.0 Euros
Se devuelve un camion el dia 9. Precio a abonar: 480.0 Euros
El turismo HU-1976-A no esta alquilado.
El camion VI-1234-HL no esta alquilado.
Ya he terminado