

## ■ Introducción a Java

- Conceptos básicos o “*Cómo aprender Java en 20 minutos sin pestañear*”



Tecnologías  Web

## ■ 1. La filosofía de Java

- La plataforma Java



Tecnologías  Web

## Ventajas de Java

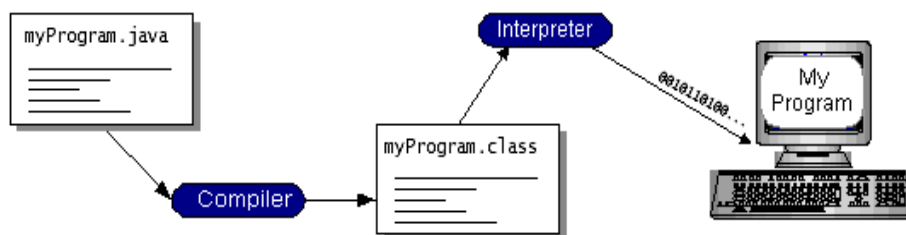
- Multiplataforma
- Orientado a objetos
- Integrado con la web (*Applets*)
- Extensa API
  - GUI, redes, threads, B.D., 3D, ...
- Util para programación distribuída



Tecnologías  Web

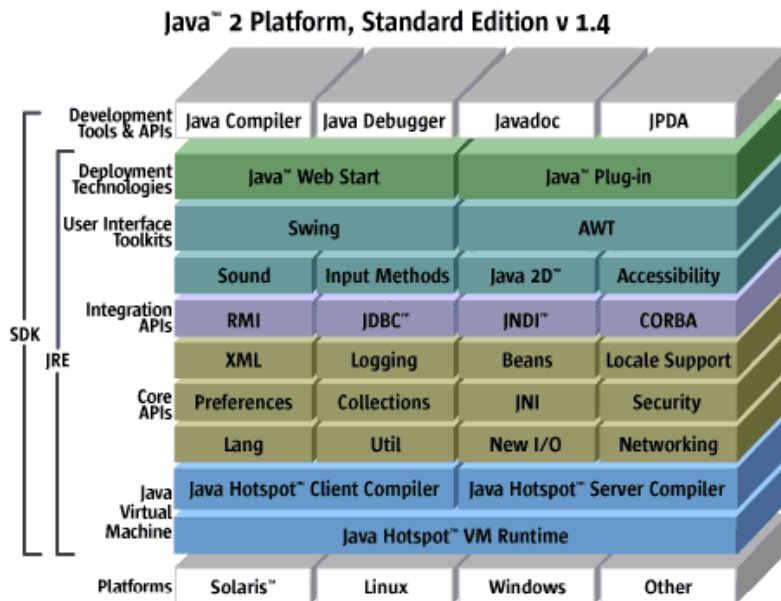
## La plataforma Java

- Compilador (.java  $\Rightarrow$  .class)
- Intérprete (máquina virtual Java)



Tecnologías  Web

# La plataforma Java (II)



Tecnologías Web

## Software de desarrollo

- Línea de comandos: J2SDK
- Visual: Eclipse

```
1package cines.beans;
2
3/**
4 * Clase abstracta que representa una entrada o grupo de entradas.
5 */
6public abstract class Entrada {
7
8    /**
9     * Identificador único para cada entrada (numerada) o grupo
10     * de entradas (sin numerar).
11     */
12    private int id;
13
14    /**
15     * Pase al que va asociada la entrada o grupo de entradas.
16     */
17    private Pase pase;
18
19    /**
20     * Código de compra que el usuario debe presentar
21     * para verificar que él ha hecho la reserva
22     */
23    private String codigo;
24
25    /**
26     * Obtiene el código de compra. Este código se genera de manera
27     * automática a partir de la fecha y hora del sistema.
28     * @return código de la entrada
29     */
30    public String getCodigo() {
31        // ...
32    }
33}
```



Tecnologías Web

## 2. El lenguaje Java

### Sentencias y tipos de datos



### Características generales

- El lenguaje es heredero directo de C++
  - Por ello comparte las sentencias de control y los operadores lógico/aritméticos, y gran parte de la filosofía OO
  
- Java = C- -
  - No se pueden definir variables o métodos fuera de las clases. Se elimina la herencia de C: **structs** y similares
  - No se permite herencia múltiple
  - No se pueden redefinir operadores
  - No existe la aritmética de punteros
  - No existe **delete**: los objetos los destruye un proceso denominado “recolector de basura” (*garbage collector*)
  - Hasta java 1.5 no existían
    - *enums*
    - *templates*



## Sentencias y operadores

- Las mismas sentencias y operadores lógicos/aritméticos que C/C++
- Los operadores **no se pueden sobrecargar**. No obstante, algunos ya están sobrecargados por defecto

“hola “ + “ amigos “



Tecnologías  Web

## Tipos de datos

### ■ Primitivos

- boolean
- char (16 bits)
- byte
- short (16 bits)
- int (32 bits)
- long (64 bits)
- float (32 bits)
- double (64 bits)

### ■ Objetos

- Java tiene una amplísima librería estándar

- String

```
String saludo = "hola";
```

- Conversión: cast igual que en C++

```
double d;
```

```
float f = (float) d
```



Tecnologías  Web

## Paso por valor vs. por referencia

- Los tipos de datos primitivos se pasan por valor, los objetos por referencia

```
Date ahora, fecha2;  
ahora = new Date();  
fecha2 = ahora;  
ahora.setYear(2004);  
System.out.print(fecha2.getYear()) //2004!!
```



## Arrays

- Son objetos (paso por referencia)
- Declaración e inicialización

```
double lecturas[]; //declaración estilo C  
double[] lecturas; //esto es equivalente  
//el tamaño del array no se puede poner en tiempo de  
//compilación. Se reserva espacio con new.  
lecturas = new double[5];  
int largo = lecturas.length //largo == 5  
//cuando es un array de objetos, estos no se inicializan  
Date fechas[] = new Date[5];  
System.out.print(fechas[0]) //null!!!!  
For(int i=0;i<5; i++)  
    fechas[i] = new Date();
```



## 3. Clases y objetos

### Constructores y métodos



### Definir una clase

```
public class Circulo {  
    Var. miembro {  
        private double centroX, centroY;  
        private double radio;  
    }  
    Constructores {  
        public Circulo() {}  
        public Circulo(double cx, double cy, double r)  
        { centroX = cx; centroY = cy; radio = r; }  
    }  
    Método {  
        public double area() {  
            return 3.141592*radio*radio;  
        }  
    }  
}
```



## Trabajar con objetos

### ■ Creación y acceso a métodos

//Los objetos en realidad son punteros a objetos. La variable circ

//en C++ la declararíamos como Circulo \*circ;

```
Circulo circ;
```

```
circ = new Circulo(0,0,1);
```

//pero donde en C++ pondríamos -> aquí ponemos .

```
float area = circ.area();
```

//Esto en C++ se puede hacer, pero no en Java.

```
circ++;
```

//Tampoco esto

```
Circulo circ2 = &circ;
```

### ■ Destrucción: recolector de basura

- No hay que hacer delete, cuando nos salimos del ámbito de la variable o la ponemos a null se borrará automáticamente.

Puede que esto no sea instantáneo, el recolector de basura actuará cuando sea necesario



## Datos y métodos de clase (*static*)

### ■ Compartidos entre todos los objetos de la misma clase. Se llaman precedidos del nombre de la clase, no del objeto.

```
public class Circulo {  
    //final significa que es una cte., no se puede modificar  
    static final float PI=3.141592;  
    static anchoLinea = 1;  
    static ponAnchoLinea(int ancho) {  
        anchoLinea = ancho;  
    }  
}
```

```
System.out.println("PI vale" + Circulo.PI);
```

```
Circulo.ponAnchoLinea(3);
```



## Comprobar si dos objetos son iguales

- ¡Cuidado!: el operador `==` aplicado sobre dos objetos significa **misma referencia** (lógico, si se piensa que los objetos en Java son punteros)

```
Circulo c1 = new Circulo(0,0,10);  
Circulo c2 = new Circulo(0,0,10);  
if (c1 == c2)           /* false */  
...  
...
```

- Para comprobar que el contenido de dos objetos es igual, se utiliza `equals`

```
String cad1 = "hola";  
String cad2 = "hola";  
if (cad1.equals(cad2)) /* true */
```

- La mayor parte de objetos de las librerías estándar de Java implementan `equals`. Si son nuestras clases, habría que implementarlo



## 4. Organización del código

### Distribución de las clases en ficheros



## Fichero Circulo.java

```
public class Circulo {
    private double centroX, centroY;
    private double radio;
    public Circulo() {}
    public Circulo(double cx, double cy, double r) {
        centroX = cx; centroY = cy;
        radio = r; }

    public double area() {
        return 3.141592*radio*radio;
    }
}
```



Tecnologías  Web

## Fichero MiPrograma.java

```
public class MiPrograma {
    public static void main(String[] args) {
        Circulo c;

        c = new Circulo(0,0,1);
        System.out.println("El área es "+c.area());
    }
}
```



Tecnologías  Web

## Puntos importantes

- ▮ Cada **.java** debe definir solo una clase pública
- ▮ El nombre de la clase debe ser **igual** que el del fichero (incluyendo mayúsculas/minúsculas)
- ▮ La clase “principal” debe definir un método **main**
  - En aplicaciones web no hay clase principal, ya que nuestro código lo ejecuta el servidor web
- ▮ Las clases referenciadas deben estar en el **CLASSPATH**
  - En aplicaciones web, el CLASSPATH no hay que gestionarlo, el servidor web lo hace por nosotros



## Paquetes

- ▮ **Package** (librería): conjunto de ficheros *.class* que están en el mismo directorio, o fichero *.JAR*
- ▮ Todos los ficheros *.java* del *package* deben comenzar por

```
package nombrePaquete;
```

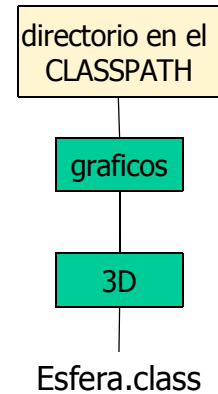


## Nombres de paquetes

- Estilo DNS, deben reflejar la estructura de directorios en la que residen los `.class`

```
/*fichero Esfera.java */  
package graficos.3D;  
    public class Esfera {  
        ...  
    }
```

- La convención habitual es utilizar el nombre de dominio de la empresa en los paquetes que ésta desarrolla
  - `es.ua.dccia.tw.blog`



Tecnología Web

## Referenciar un paquete

- `import nombre_paquete.clases_importadas`**

```
import graficos.3D.*;           //Todo el paquete  
import graficos.3D.esfera;     //Solo una clase
```

- Si no se pone el **import**, hay que poner el nombre completo del objeto y método

```
graficos.3D.esfera.dibujar();  
– Necesario cuando hay clases que se llaman igual en distintos  
paquetes  
fecha1 = new java.util.Date();  
fecha2 = new java.sql.Date();
```



Tecnología Web

## 5. Herencia

### Extendiendo clases



### Ejemplo

#### Supongamos la siguiente clase

```
public class Circulo {
    protected double x, y, r;
    public Circulo(double x, double y, double r) {
        //this es lo mismo que en C++
        this.x = x; this.y = y; this.r = r;
    }
    public void mover (double a, double b) {
        x=a; y=b;
    }
    public void escalar (double r) {
        this.r = r;
    }
    public double area () {return Math.PI*r*r;}
}
```



## Definir una subclase: *extends*

```
public class Elipse extends Circulo {
    //si necesitamos llamar a la clase "madre" podemos emplear "super"
    public Elipse(double x, double y, double r, double r2) {
        super(x, y, r);
        this.r2 = r2;
    }
    //por supuesto, se heredan los métodos y datos anteriores, y podemos crear nuevos
    private double r2;
    public void escalar(double factor) {
        super.escalar(factor);
        r2 *= factor;
    }
    //y podemos sobrescribir lo que ya había, salvo que estuviera marcado como final
    public double area() {return Math.PI*r*r2;}
}
```



Tecnologías  Web

## Polimorfismo

- Todos los métodos son implícitamente **virtual**

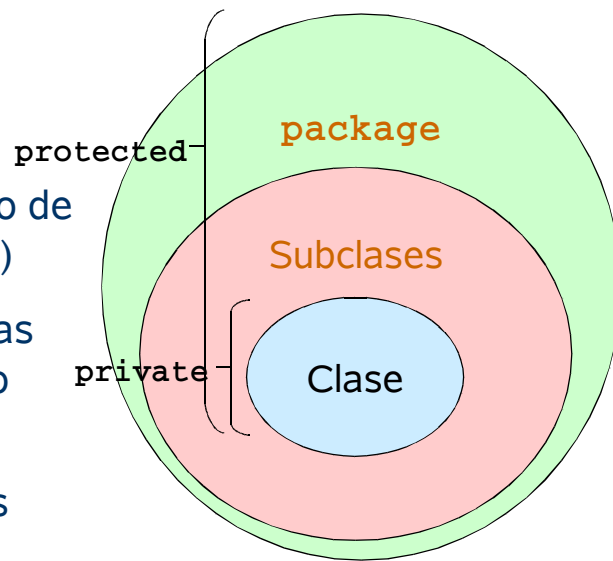
```
Elipse e = new Elipse(0,0,10,20);
Circulo c = (Circulo) e;
//Se llamará al método de Elipse, no de Circulo
System.out.println(c.area()); //628.31
```



Tecnologías  Web

## Modificadores de acceso (public, private,...)

- Basados en los de C++, pero muy simplificados
- **private**: solo dentro de la clase (no heredable)
- **protected**: todas las que están en el mismo **package**
- **public**: en todos los ámbitos



## 6. Interfaces

- Clases que comparten comportamiento



## Motivación

- ▮ Puede ser que dos clases no sean lo mismo. Simplemente tienen que tener los mismos métodos (aunque cada uno los implemente a su manera)
- ▮ Ejemplo: quizá hemos forzado un poco las clases Circulo y Elipse. Una elipse no es un círculo, solo que también se puede
  - mover
  - escalar
  - calcular su área,...
- ▮ **Interfaz:** conjunto de métodos que debe implementar una clase. El compilador comprueba que esto sea cierto, si no, no compilará



## Declaración y uso de una interfaz

//declaración: fichero Figura.java

```
public interface Figura {  
    public void escalar(double factor);  
    public double area();  
}
```

//en el fichero Circulo.java, ahora:

```
public class Circulo implements Figura {  
    ...
```

//cuidado, como se nos olvide implementar un método escalar y otro area, el  
//compilador “se negará” a compilar



## Variables con el tipo de la interfaz

```
Figura f;
```

```
//Esto es válido, ya que un círculo cumple el interfaz de Figura
```

```
f = new Circulo(0,0,10);
```

```
//pero esto no tiene sentido, ya que Figura no es una clase
```

```
f = new Figura();
```



Tecnologías  Web

## 7. Tratamiento de errores

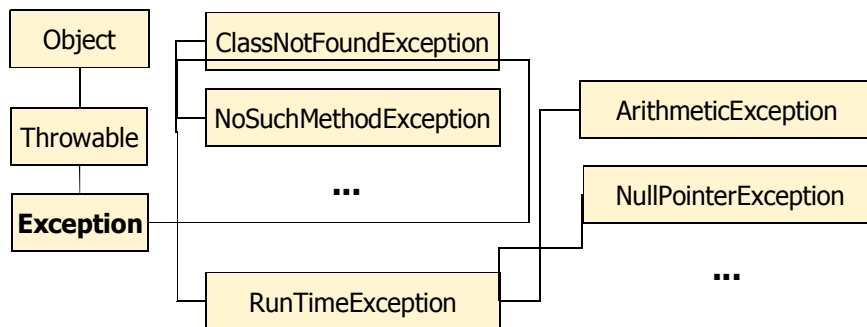
### Excepciones



Tecnologías  Web

# Excepciones

- Eventos que interrumpen el flujo de ejecución
- Están en la jerarquía de clases estándar de Java



Tecnologías  Web

## Try/catch

- Try: bloque en el que se puede producir un error
- Catch: manejador de un tipo de excepción

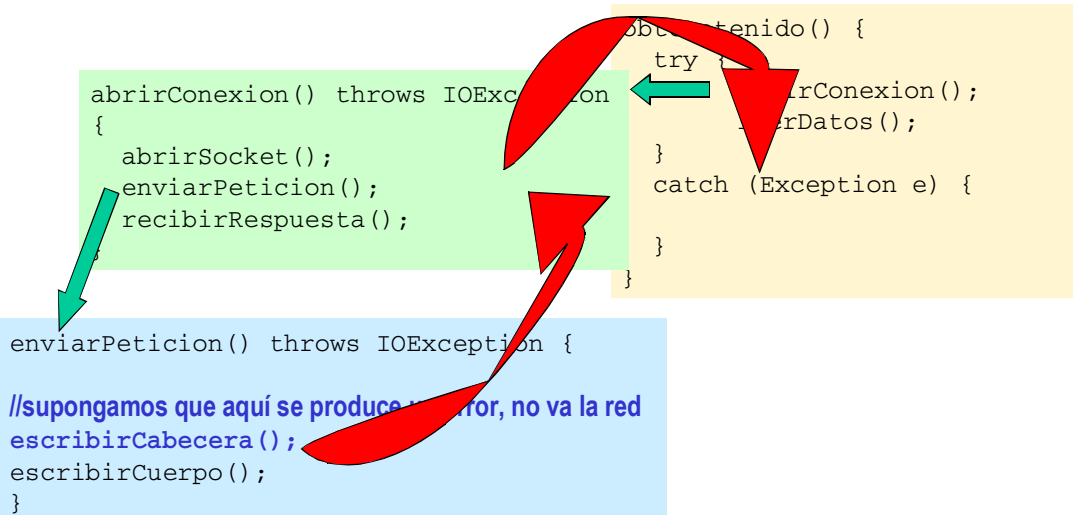
```
try {  
    ...  
    z = x/y;  
    //si lo anterior ha generado una excepción esto ya ni se hace  
    system.out.println(z);  
}  
catch (ArithmeticException e) {  
    System.out.println("error: "+ e);  
}
```



Tecnologías  Web

# Propagación de excepciones

## “Pasando la pelota”



## ¿Qué hacer con una excepción?

### Posibilidades

- Ignorarla: por defecto se propagará al nivel superior
- Capturarla (**catch**)
- Pasarla al nivel superior (**throw**)

### El tratamiento difiere según el tipo

- Exc. *comprobadas*: **catch** o **throw**: el programador debe asumir que puede pasar un error y pensar un curso de acción dado el caso
- Exc. *sin comprobar*: cualquier posibilidad. Podemos hacer como si todo fuera a funcionar bien



## Excepciones comprobadas

- Son fallos puntuales, ante los que el programador puede intentar hacer algo (aunque sea imprimir un mensaje de error)
- Cuando llamamos a un método que puede generar una excepción comprobada hay que capturarla o lanzarla

```
//esto no compilará, ya que llamo a un constructor que puede
//generar una excepción FileNotFoundException, pero no le
//hago caso
import java.io.*;
miMetodoQueNoCompila() {
    FileReader f=new FileReader("datos.txt");
}
```

- Para especificar que la excepción se lanza se pone en la cabecera throws. Ahora sí compilará, pero quien llame a miMetodo se encontrará en la misma disyuntiva

```
miMetodo() throws FileNotFoundException
```



Tecnologías  Web

## Excepciones sin comprobar

- Errores graves en tiempo de ejecución
  - Salirse de un array
  - Referencia nula
  - Dividir por 0
  - ...
- Sería muy tedioso estar siempre pendiente de ellas, ya que se pueden dar en casi cualquier instrucción
- Si se ignoran Java muestra un mensaje de error apropiado y el programa aborta, aunque se pueden capturar



Tecnologías  Web

## Excepciones propias

```
public class MiExcepcion extends Exception {  
    MiExcepcion(String mens) {  
        super(mens);  
    }  
}  
  
...  
throw new MiExcepcion("La cosa está muy malita");
```



Tecnologías  Web

## 8. Colecciones

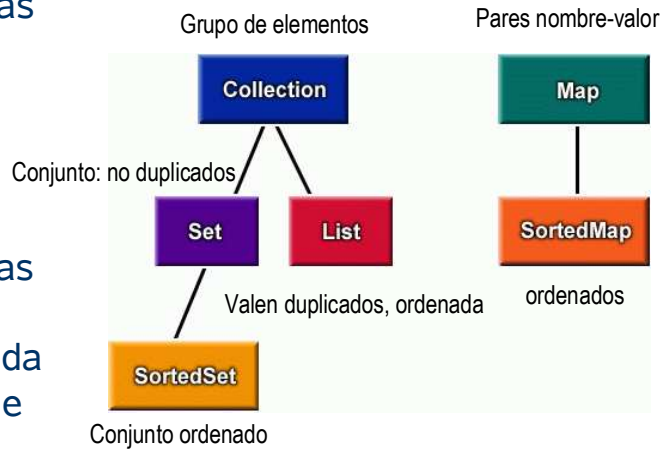


Tecnologías  Web

# Colecciones

- Conjunto de clases que modelan listas, tablas hash, conjuntos, árboles,...

- Lo de la figura son interfaces. Hay varias implementaciones alternativas para cada interfaz (más las que podríamos hacer nosotros)



# Listas

- Aunque hay varias implementaciones de listas, la más típica es ArrayList

```
//todo esto está en java.util
import java.util.*;
//esta lista solo puede contener String
ArrayList<String> lista;
lista = new ArrayList<String>();
//añadir objetos
lista.add("Hola");
lista.add("que tal");
//El siguiente bucle se lee como "for e in lista"
for (String e : lista)
    System.out.println(e);
//sacar cuántos elementos hay en la lista
int largo = lista.size();
```

