Primera Escuela de la Red Temática SVO.
Madrid, 27-28 Noviembre, 2006

JAVA BÁSICO

Raúl Gutiérrez Sánchez
LAEFF - INTA
raul@laeff.inta.es

¿Qué es Java?

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems al inicio de la década de 1990. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es ejecutado (usando normalmente un compilador JIT), por una máquina virtual Java.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos mucho más simple y elimina herramientas de bajo nivel como punteros.

¿Cómo es Java?

- "Simple". Características de lenguaje potente, pero sin los elementos menos usados y más confusos de C y C++.
- Orientado a objetos.
- Distribuido. Extensas capacidades de interconexión TCP/IP.
- Robusto. No hay punteros. Verificaciones en busca de problemas en tiempo de compilación y de ejecución. Comprobación de tipos. Declaración explícita de métodos. Manejo de memoria.
- Arquitectura neutral. Se ejecuta sobre una máquina virtual, de la que hay implementaciones para múltiples arquitecturas y sistemas operativos.
- Seguro.
- Multitarea.

¿Por qué Java?

- Lenguaje de propósito general.
- Portable: mayor rendimiento del tiempo de desarrollo.
- Muy extendido: gran cantidad de librerías.
- Facilidad web: jsp, applets, ...
- VO: la mayoría del software VO realizado en JAVA.

¿Orientado a Objetos?

“Empty your mind”

¿Orientado a Objetos?

- Todo es un Objeto: variable “chula”; almacena datos pero puedes pedirle que haga operaciones sobre sí misma. Puede elegirse cualquier componente conceptual del problema (perros, edificios, servicios...) y representarlos como objetos.
- Un programa es un conjunto de objetos diciéndose mediante mensajes qué hacer: puede pensarse en un mensaje como una llamada a un método de un determinado objeto.
- Cada objeto tiene su propia memoria creada a partir de otros objetos: una nueva clase de objetos se crea a partir de otros objetos ya existentes.
- Todo objeto es de un tipo: cada objeto es una “instancia de una clase” donde “clase” es sinónimo de “tipo”.
- Todos los objetos de un tipo pueden recibir los mismos mensajes.

Alan Kay, creador de Smalltalk

Ejemplo

Bombilla

lum

encender()
apagar()
subirLum()
bajarLum()
fijarLum(int)

```
Bombilla bomb1 = new
Bombilla();

bomb1.encender();
bomb1.subirLum();
bomb1.fijarLum(9);
bomb1.apagar();
```

```
class Bombilla{
    private int lum = 0;

    public void encender(){
        lum = 5;
    }
    public void apagar(){
        lum = 0;
    }
    public void subirLum(){
        lum++;
    }
    public void bajarLum(){
        lum--;
    }
    public void fijarLum(int nuevaLum){
        lum = nuevaLum;
    }
}
```

campo (field)

métodos

public: accesible por todo el mundo.

private: sólo accesible desde métodos de la propia clase.

protected: igual que private, pero accesible desde clases que heredan de esta.

Reutilización

```
class Coordenadas{
    Coordenadas( double ra, double de ){
        this.ra = ra;
        this.de = de;
    };

    private double ra;
    private double de;

    public double getRa(){
        return ra;
    }
    public double getDe(){
        return de;
    }
    public double dist(Coordenadas coord2){
        return dist( this.ra,
                     this.de,
                     coord2.getRa(),
                     coord2.getDe());
    }
    public static double dist(double ra1,
                              double de1,
                              double ra2,
                              double de2){

        ......
        return resultado;
    }
}
```

```
class ObjetoCeleste{
    ObjetoCeleste(){};

    private String nombre;
    private double magV;
    private Coordenadas coords;

    public void setNombre(String n){
        coords = c;
    }
    public void setCoords(Coordenadas c){
        coords = c;
    }
    public double setMagV(double m){
        magV = m;
    }
    public Coordenadas getCoords(){
        return coords;
    }
        ......

    public double dist(ObjetoCeleste o2){
        return coords.dist(o2.getCoords());
    }

}
```

“Static”

```
class Coordenadas{
    Coordenadas( double ra, double de ){
        this.ra = ra;
        this.de = de;
    };

    private double ra;
    private double de;

    public double getRa(){
        return ra;
    }
    public double getDe(){
        return de;
    }
    public double dist(Coordenadas coord2){
        return dist( this.ra,
                     this.de,
                     coord2.getRa(),
                     coord2.getDe());
    }
    public static double dist(double ra1,
                              double de1,
                              double ra2,
                              double de2){
        ......
        return resultado;
    }
}
```

No estático:

Hace falta tener una referencia de un objeto ya creado.

```
Coordenadas c1 = new Coordenadas(10,10);
Coordenadas c2 = new Coordenadas(90,90);
double d = c1.dist(c2);
```

Estático:

Puede usarse sin haber creado ningún objeto. También reciben el nombre de campos y métodos “de clase”.

```
double d = Coordenadas.dist(10,10,
                            90,90);
```

Herencia

```
class ObjetoCeleste{

    private String nombre;
    private double magV;
    private Coordenadas coords;

    public void setNombre(String n){
        coords = c;
    }
    public void setCoords(Coordenadas c){
        coords = c;
    }
    public double setMagV(double m){
        magV = m;
    }
    public Coordenadas getCoords(){
        return coords;
    }
        ......

    public double dist(ObjetoCeleste o2){
        return coords.dist(o2.getCoords());
    }

}
```

```
class Estrella extends ObjetoCeleste{

    private double teff;

    public void setTeff(double t){
        teff = t;
    }

    public double getTeff(){
        return teff;
    }
}
```

```
Estrella st1 = new Estrella();

st1.setNombre("vega");
st1.setMagV();
st1.setCoords(new Coordenadas(279.23,38.78));
st1.setTeff(9434);

....
```

Tipos Primitivos

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	Boolean
char	16-bit	Unicode 0	Unicode 2^{16}-1	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15}	$+2^{15}-1$	Short
int	32-bit	-2^{31}	$+2^{31}-1$	Integer
long	64-bit	-2^{63}	$+2^{63}-1$	Long
float	32-bit	IEEE754	IEEE754	Float
double	64-bit	IEEE754	IEEE754	Double
void	—	—	—	Void

- El tamaño no cambia con la arquitectura: mayor portabilidad.
- Todos los tipos numéricos son con signo, no los hay sin signo.

Sobrecarga de métodos

```
class Coordenadas{
    Coordenadas( double ra, double de ){
        this.ra = ra;
        this.de = de;
    };

    private double ra;
    private double de;

    public double getRa(){
        return ra;
    }
    public double getDe(){
        return de;
    }
    public double dist(Coordenadas coord2){
        return dist( this.ra,
                     this.de,
                     coord2.getRa(),
                     coord2.getDe());
    }
    public static double dist(double ra1,
                              double de1,
                              double ra2,
                              double de2){
        ......
        return resultado;
    }
}
```

Pueden definirse métodos que se llamen igual simplemente cambiando el número, tipo u orden de los argumentos (esto último no es recomendable).

Paquetes: la base de las librerías

Coordenadas.java

```
package astronomia;

class Coordenadas{
    .....
}
```

ObjetoCeleste.java

```
package astronomia;

class ObjetoCeleste{
    .....
}
```

Programa1.java

```
import astronomia.Coordenadas;

class Programa1{
    .....
}
```

Programa2.java

```
import astronomia.*;

class Programa2{
    .....
}
```

- Permiten establecer espacios de nombres independientes: puede haber clases y métodos con el mismo nombre siempre que estén en paquetes diferentes.
- Los nombres de los paquetes se mapean sobre directorios:

astronomia.estrellas → [src]/astronomia/estrellas

Excepciones

```
class Coordenadas{
    Coordenadas( double ra, double de ) throws Exception{
        if(ra < 0 || ra >=360 ){
            throw new Exception("RA out of bounds.");
        }
        if(de<-90 || de>90){
            throw new Exception("Dec. out of bounds.");
        }
        this.ra = ra;
        this.de = de;
    }
    ....
}
```

```
public class Programa1 {
    public static void main(String args[]){
        Coordenadas c1;
        try{
            c1 = new Coordenadas(450,10);
        }catch(Exception e){
            System.out.println("Error en coordenadas"+e.getMessage());
        }
    }
}
```

Excepciones

```
class Coordenadas{
    Coordenadas( double ra, double de ) throws Exception{
        if(ra < 0 || ra >=360 ){
            throw new Exception("RA out of bounds.");
        }
        if(de<-90 || de>90){
            throw new Exception("Dec. out of bounds.");
        }
        this.ra = ra;
        this.de = de;
    }
    ....
}
```

```
public class Programa1 {
    public static void main(String args[]) throws Exception{
        Coordenadas c1 = new Coordenadas(450,10);
    }
}
```

```
Exception in thread "main" java.lang.Exception: RA out of bounds.
        at Coordenadas.<init>(Coordenadas.java:32)
        at Programa1.main(Programa1.java:33)
```

¡Hola mundo!

```
// Aplicacion HolaMundo
class HolaMundo{
    public static void main( String args[] ){
        System.out.println("¡Hola Mundo!");
    }
}
```

Compilación (genera HolaMundo.class):
`%javac HolaMundo.java`

Ejecución:
`%java HolaMundo`

Comentario

Uso de clases externas.

Método main.

Paso de valores en línea de comandos.

```
// Aplicacion HolaMundo extendida
import Coordenadas;

class HolaMundo{
    public static void main( String args[] ){
        Coordenadas coords = new Coordenadas(270,18);
        System.out.println("¡Hola Mundo!");
        System.out.println("Tus coordenadas son: "
                           +coords.getRa()+","+coords.getDe());
    }
}
```

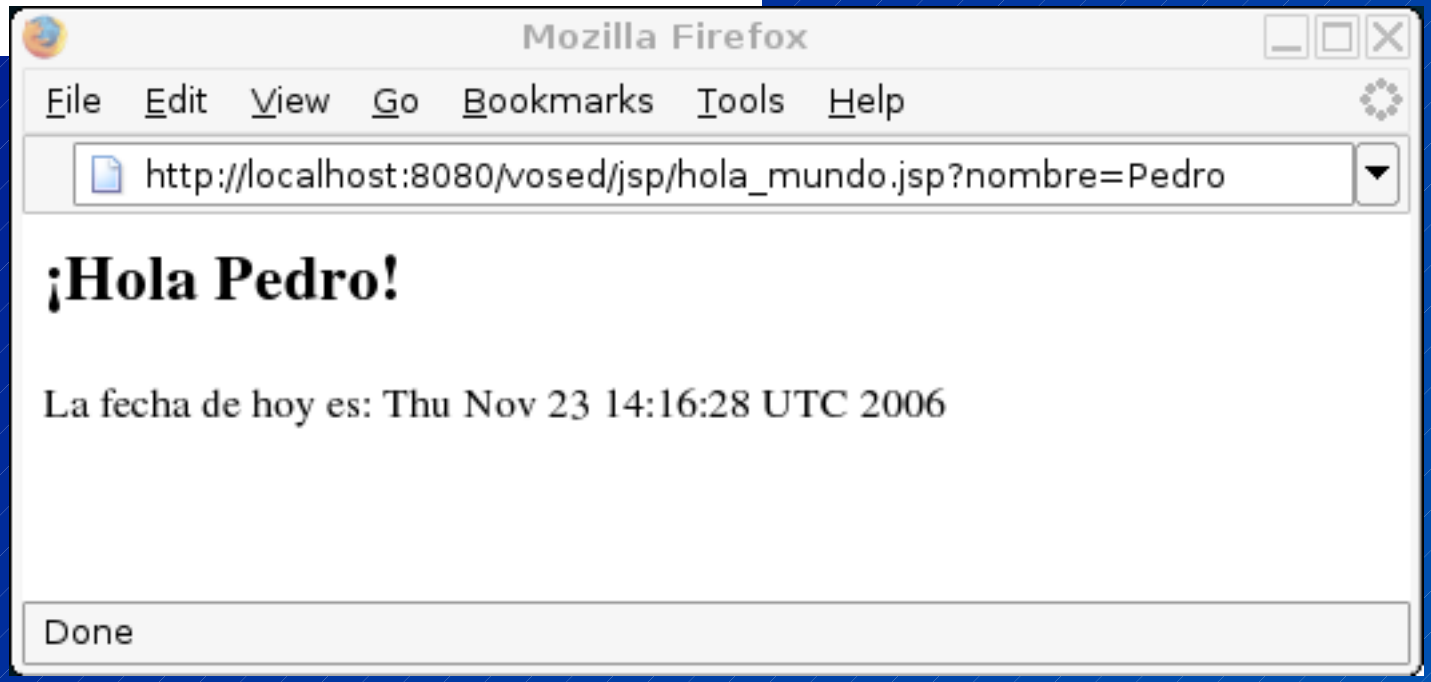
Java en la web

- **Servlets:** elementos escritos en Java capaces de interactuar con los navegadores para generar contenido web dinámicamente. Para su ejecución necesitan encontrarse dentro de un contenedor de servlets (p.e. Tomcat), capaz de ejecutar su código.
- **JSP:** (Java Server Pages). Lenguaje de “scripting” que simplifica la creación de servlets java. Generalmente consisten en código HTML con partes de código java incrustadas, pudiendo utilizar todos los recursos de Java, como librerías.
- **Applets:** aplicaciones java que pueden ser incrustadas en páginas HTML. El navegador descarga el software correspondiente desde el servidor web y lo ejecuta en la máquina cliente, con ciertas restricciones sobre el acceso a los recursos de esa máquina.

...

JSP

```
<%@ page import="java.util.Date" %>
<%
    String nombre = request.getParameter("nombre");
    Date fecha = new Date();
%>
<html>
<body>
<h2>¡Hola <%=nombre%>!</h2>
La fecha de hoy es: <%=fecha%>
</body>
</html>
```



¿Donde buscar?

Thinking in Java 3ª edición.

http://www.mindview.net/Books/TIJ/

Documentación Sun.

http://java.sun.com/docs/