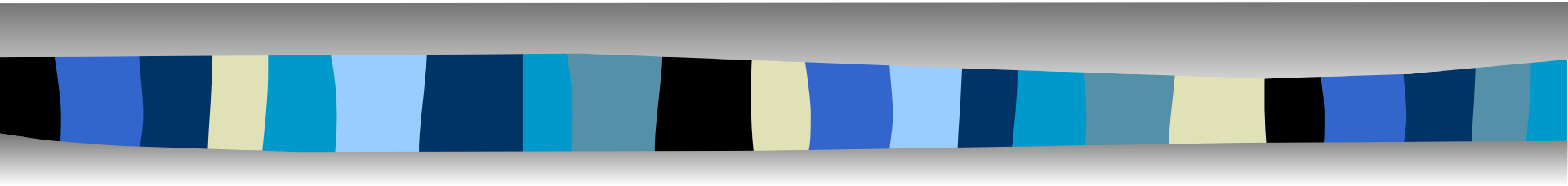
# JAVA: Applets



## Diseño de aplicaciones web

mperez@fi.upm.es

# Características de Java (I)

- Simple
  - El programador no tiene que gestionar la memoria!
- Orientado a Objetos
  - Encapsulación de datos y reutilización.
- Distribuido
  - Comunicación a través de la red.
- Robusto
- Arquitectura Neutral
- Seguro
  - La máquina virtual java proporciona un contexto de ejecución seguro.

# Características de Java (II)

- Transportable
  - Independiente de la plataforma *(Write once, run anywhere).*
- Interpretado
  - Más lento. Hay opción de ejecutar código compilado para una arquitectura determinada.
- Multitarea
  - Procesamiento paralelo.
- Dinámico
  - Late Binding.
- Integración con páginas web
  - Applets o servlets.

# Java es distribuido

- Capacidades de interconexión TCP/IP.
  - Proporciona librerías y herramientas para que los programas construidos puedan ser distribuidos:
    - Librerías de rutinas para acceder e interactuar con protocolos como http y ftp.

# Java es seguro

- Las aplicaciones Java no acceden a zonas “delicadas” de memoria o del sistema.
  - La seguridad se integra en el momento de la compilación.

# Máquina virtual Java

- Permite proporcionar portabilidad.
  - Independencia del computador.
  - Independencia del sistema operativo.
- Compilación del código Java a un código independiente de máquina (*Byte code*).
- El *byte code* se puede ejecutar en una máquina virtual simulada por software: JVM (*Java Virtual Machine*).
  - Una JVM es necesaria para cada computador concreto.

# Máquina virtual Java

- Enlace a bibliotecas en tiempo de ejecución y cuando se hace referencia a las mismas (enlace dinámico):
  - El *byte code* de Java se transmite más rápido a través de la red.
  - Portabilidad respecto a las bibliotecas.
  - Facilidad de mantenimiento y actualización de software. Se utiliza la última versión de la biblioteca.
- Es posible utilizar código nativo (código compilado para una plataforma concreta):
  - Más rápido.
  - No portable.

# Aplicaciones Java

- Dependiendo en que entorno se vaya a ejecutar, los programas se denominan:
  - Stand-Alone: Programas tanto gráficos como de línea de comandos.
  - Applet: Programas embebidos en el “navegador web”.
  - Servlets: Programas embebidos en el “servidor web”.

# Una aplicación en Java

- Cada programa de la aplicación es una clase.
- Cada clase tiene un método llamado *main*.
- *main* es un método estático, accesible sólo a la clase:
  - No es necesario crear una instancia de la clase.
  - No obstante, se puede crear una instancia de la clase!
- Un *applet* no se rige por las mismas normas.

# Hola Mundo!!

```
public class HolaMundoApp
{
  public static void main(String args[ ])
  {
  System.out.println ("Hola mundo");
  }
}
```

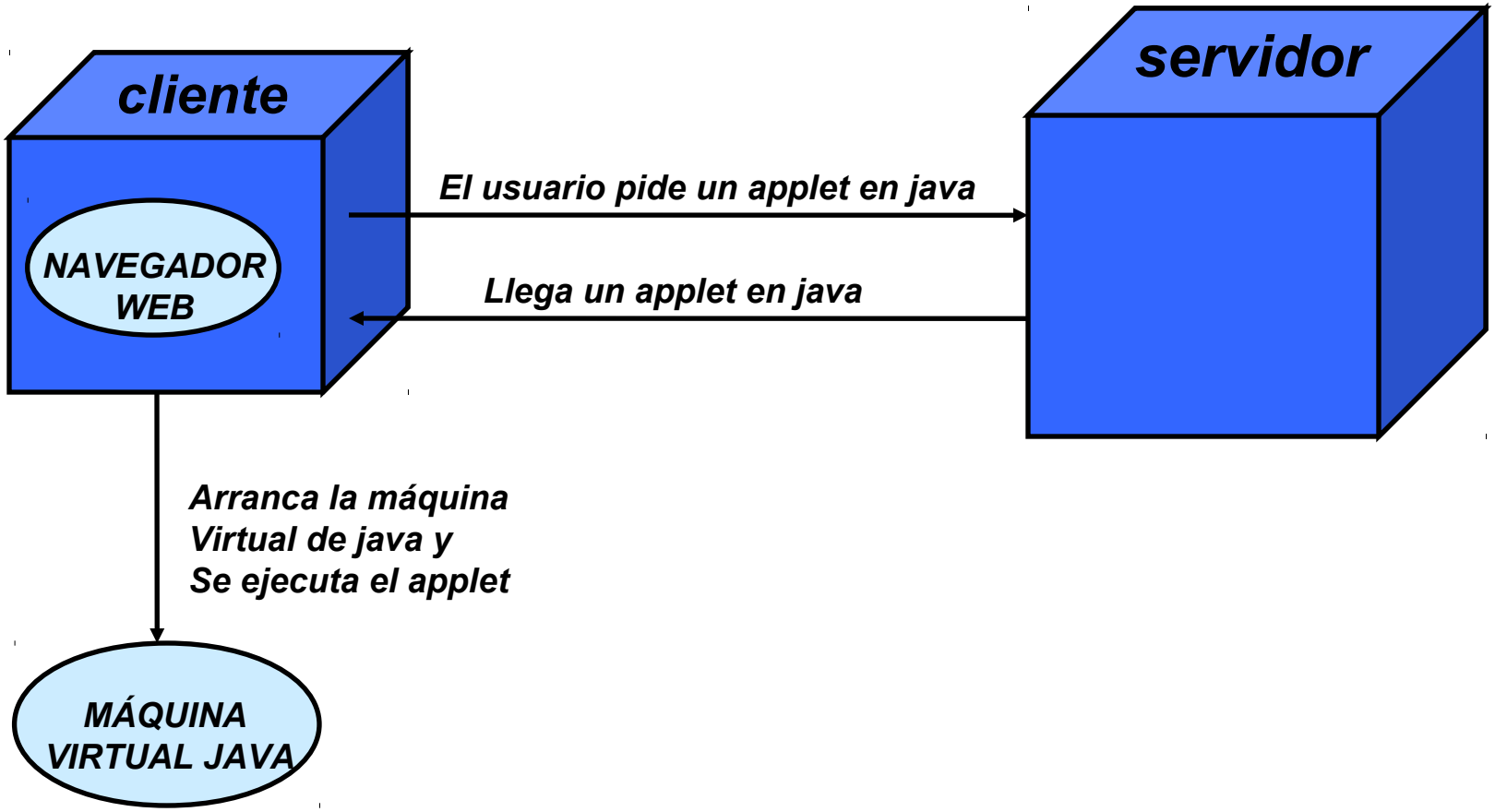
# Compilación y ejecución de HolaMundo

- El código se guarda en un fichero llamado HolaMundoApp.java.
  - El nombre de la clase debe ser igual al del fichero que contiene el código fuente.
- Compilación del programa:
  - javac HolaMundoApp.java

    ⇓

    HolaMundoApp.class
- Ejecución del programa:
  - java HolaMundoApp

# Applets


cliente
NAVEGADOR WEB
servidor
El usuario pide un applet en java
Llega un applet en java
Arranca la máquina
Virtual de java y
Se ejecuta el applet
MÁQUINA VIRTUAL JAVA

# Applets

- Integración página web/ código Java:

```
<title> Página web con Applet </title>
<applet code= "Codigo.class"
width=300 height=200 > </applet>
```

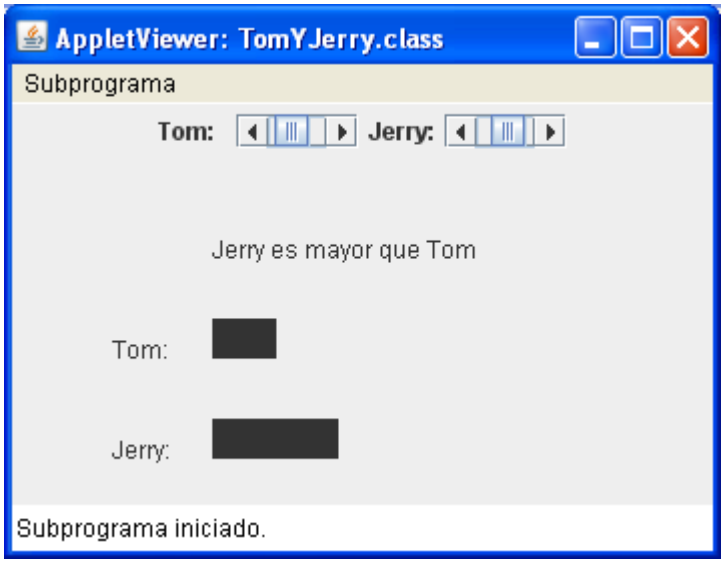
- Método *init();*
- Método *paint();*

# Applet Hola Mundo!!

```
import java.awt.*;
import java.applet.Applet;
public class HolaMundoApplet extends Applet {
{
   public void paint(Graphics g)
   {
   g.drawString("Hola mundo”, 50, 50);
   }
}
```

# Applets

- Visor de applets:
  - appletviewer



# Actividades de un applet

- Iniciación de un applet

```
public void init() {
    ...
}
```

- *Comienzo de un applet*

```
public void start() {
    ...
}
```

# Actividades de un applet

- Parada de un applet

```
public void stop() {
    ...
}
```

- *Destrucción de un applet*

```
public void destroy() {
    ...
}
```

# Actividades de un applet

- Actualización de un applet

```
public void update(Graphics g) {
    ...
}
```

- *Pintado de un applet*

```
public void paint(Graphics g) {
    ...
}
```

# Paso de parámetros a un applet

- `String getParameter(String name);`
  - `Name`: Nombre del parámetro
  - Devuelve el valor que toma el parámetro
  - Pares (Nombre, valor)
- Uso de los métodos de los “wrappers” de los tipos básicos para interpretar el valor del parámetro
  - Ejemplo: Integer.parseInt(getParameter(“Numero”);
- Etiqueta PARAM

<APPLET CODE ….>
<PARAM NAME = nombrePar1 VALUE = “val1”>
<PARAM NAME = nombrePar2 VALUE = “val2”>
</APPLET>

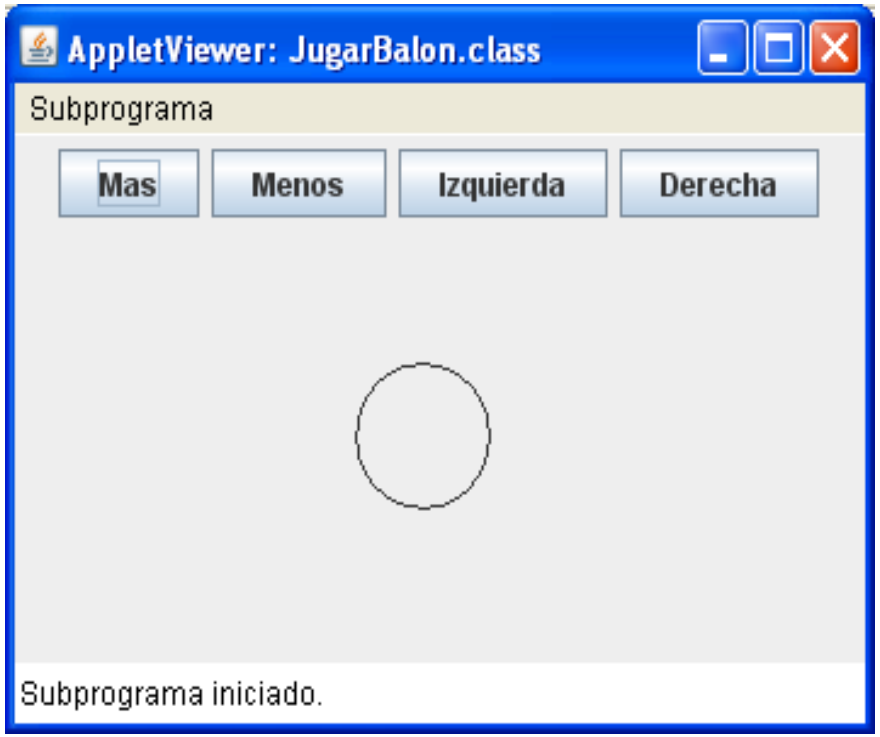
# Un ejemplo más completo

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class JugarBalon extends Applet
    implements ActionListener {

    private Button mas, menos, izq, der;
    private Balon miBalon;

    public void init() {
    mas = new Button("Mas");
    add(mas);
    mas.addActionListener(this);
    menos = new Button("Menos");
    add(menos);
    menos.addActionListener(this);
    izq = new Button("Izquierda");
    add(izq);
    izq.addActionListener(this);
    der = new Button("Derecha");
    add(der);
    der.addActionListener(this);

    miBalon = new Balon();
    }
    public void actionPerformed (ActionEvent
    event) {
    if (event.getSource() == mas)
    miBalon.crecer();
    if (event.getSource() == menos)
    miBalon.decrecer();
    if (event.getSource() == izq)
    miBalon.irIzq();
    if (event.getSource() == der)
    miBalon.irDer();
    repaint();
    }
    public void paint(Graphics g) {
    miBalon.mostrar(g);
    }
}
```

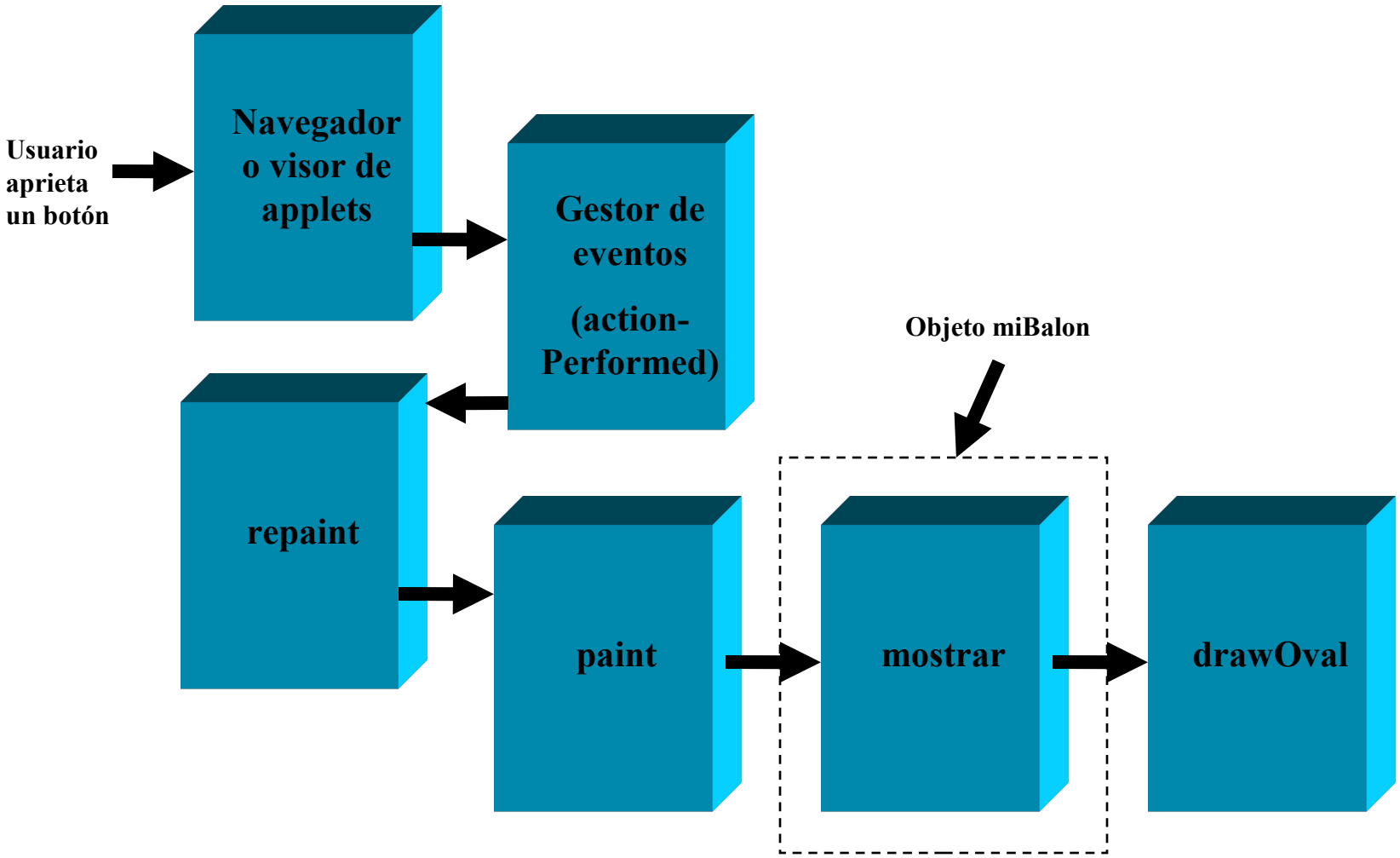
# Un ejemplo más completo

```
import java.awt.*;
class Balon {
    private int diametro;
    private int coordX, coordY;

    public Balon() {
    diametro = 10;
    coordX = 20;
    coordY = 50;
    }
    public void mostrar (Graphics g) {
    g.drawOval(coordX,coordY,diametro,diametro);
    }
    public void irIzq() {
    coordX = coordX - 10;
    }
    public void irDer() {
    coordX = coordX + 10;
    }
    public void crecer() {
    diametro = diametro + 5;
    }
    public void decrecer() {
    diametro = diametro - 5;
    }
}
```

# Un ejemplo más completo



# Un ejemplo más completo



# JAVA vs CGI

| CGI | JAVA |
| --- | --- |
| Ejecución en el servidor | Ejecución en el cliente |
| Sin tiempo de espera de carga | Tiempo de espera de carga |
| Esperas continuas para lograr la interactividad | No requiere esperas posteriores a la carga inicial |
| Precisa de un servidor | No requiere un servidor: se puede almacenar |
| Uso de diferentes lenguajes | Necesidad de aprender un nuevo lenguaje |
| Nivel de interactividad bajo | Nivel de interactividad alto |
| Funciona con cualquier navegador web | Requiere un cliente especial |

# Ventaja sobre CGIs

- Interactividad
  - Uso de MVJ independiente de hardware capaz de ejecutar *applets*.
  - Los *applets* se ejecutan en el cliente como una aplicación más.
    - Responden al teclado y al ratón.
    - Gestionan imagen y sonido.