Seminario 1

Introducción a Java

PROGRAMACIÓN 3

David Rizo, Pedro J. Ponce de León
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Contenidos

Contenidos

Características

Sintáxis básica

Programa principal

Salida básica

Compilación y ejecución

Tipos de datos básicos

Objetos

Excepciones

Cadenas

Arrays

Métodos

Control de flujo

Paquetes

Librerías Java

CLASSPATH

Documentación

Archivos JAR

ANT

Características principales de Java

- Lenguaje orientado a objetos: (casi) todo son objetos
- Ficheros fuente: extensión *.java*. Cada fichero contiene una clase. El nombre del fichero debe coincidir con el de la clase.
- Se compila a *bytecode*. Por cada fichero fuente se genera un fichero de *bytecode* con extensión *.class*
- Librerías: ficheros *.jar* que contienen ficheros *.class*
- *JDK, Java Development Kit* : incluye el compilador de Java (`javac`), las librerías estándar de Java y otras utilidades para desarrollar con código Java.
- Entornos de desarrollo integrados (IDE) principales: *Eclipse**, *Netbeans*, e *Intellij IDEA*

(*) En prácticas usaremos Eclipse.

David Rizo, Pedro J. Ponce de León

Características principales de Java

- El *bytecode* es un lenguaje intermedio que es interpretado y ejecutado por la *máquina virtual de Java*, que es multiplataforma.
- *JRE, Java Runtime Environment*: incluye la máquina virtual (`java`), librerías estándar y otras utilidades para la ejecución de *bytecode*.
- El JDK incluye al JRE. En una máquina donde desarrollemos aplicaciones Java instalaríamos el JDK.
- En una máquina donde queremos ejecutar aplicaciones Java, sólo es necesario instalar el JRE.

David Rizo, Pedro J. Ponce de León

Sintaxis básica

Las reglas de nombrado de identificadores son básicamente las mismas que se usan para C++. En la 'jerga' de Java, los campos de una clase se denominan '*atributos*' y las funciones de una clase se llaman '*métodos*'.

```
// Este fichero se debe guardar como Clase.java
// Generalmente, cada clase se situa en un fichero
public class Clase {
  /* Todos los atributos deben especificar la visibilidad */
  private int campo1;
  private float campo2; // los atributos se inicializan a 0

  /* El constructor no devuelve nada */
  public Clase() {
     campo1 = 0;
  }
  /* Todos los metodos se definen inline */
  public int getCampo1() {
    return campo1;
  }
}
```

Constantes, estáticos [1]

Constantes

Las constantes se definen usando la palabra reservada `final`

```
public final int KN=10;
```

Métodos y campos estáticos

Se definen usando la palabra reservada `static`

```
private static int contador=1;
public static final int KNN=10;
public static void incrementaContador () {
  contador++;
}
```

Programa principal

main

El punto de entrada a la aplicación es el método *main*, un método estático:

```
// esto es una clase normal
public class ClaseConMain {
  // que ademas tiene el metodo main
  // por lo que se puede invocar desde la maquina virtual
  public static void main(String[] args) {
    // el array 'args' contiene los argumentos que se pasan
    // al programa desde l nea de comando.
    // sin incluir (como en C++) el nombre del ejecutable
  }
}
```

Salida por consola

Salida

Para imprimir por la salida estándard usaremos

```
System.out.print("Cadena"); // sin salto de linea final
System.out.println(10+3); // con salto de linea final
```

Para imprimir por la salida de error

```
System.err.println("Ha ocurrido un error...");
```

Compilación y ejecución en línea de comando

Compilación

La compilación en línea de órdenes (terminal):

```
> javac ClaseConMain.java
```

Genera el fichero con *bytecode* `ClaseConMain.class`, por defecto en el directorio actual.

Ejecución

Con la orden `java` invocamos a la máquina virtual, indicando el nombre de la clase que contiene el método *main()*

```
> java ClaseConMain
```

Para esto el archivo *ClaseConMain.class* debe estar, por defecto, en el directorio actual (más adelante veremos que esto se puede cambiar).

Compilación y ejecución en línea de comando

ACTIVIDAD

Ahora estás listo para compilar y ejecutar tu primer programa Java. en la carpeta `codigo` que acompaña a este documento encontrarás la clase *ClaseConMain* lista para compilar y ejecutar.

- Abre el archivo *ClaseConMain.java* con un editor de texto plano o de código y échale un vistazo para hacerte una idea de lo que hace el programa.
- Luego compílalo y ejecútalo desde un terminal situándote en la carpeta *codigo*.

Tipos de datos escalares [2]

Java es un lenguaje fuertemente tipado. Cada expresión del lenguaje tiene un tipo asociado. Casi todos los tipos en Java son objetos, menos los tipos escalares básicos:

Tipos escalares (no objetos)

byte, short, int, long, float, double, char, boolean

Los literales se especifican así:

```
int x = -14;
float a = 100.3f;
double b = 100.3; // o 1.03e2
char c = 'a';
boolean d = true; // o false
```

Operadores

Disponemos de los mismos operadores que en C++

```
a+b*3; a++; if (a==1) b=2; a = (float)b; ...
```

Tipos de datos escalares [2]

ACTIVIDAD

En algunas páginas como ésta o la anterior, verás que el título tiene un número entre corchetes ([2]). Este número te servirá para comprobar el código que se presenta en esa página, ejecutando un programa que encontrarás en la carpeta `codigo` adjunta. El archivo de código fuente es `Clase.java`. Compílalo y ejecútalo pasándole como argumento el número indicado entre los corchetes:

```
> java Clase 2
```

te mostrará el resultado del código que aparece en la página. También puedes consultar el código fuente y buscar la cadena `"// [N]"`, sustituyendo 'N' por el número correspondiente. Te llevará a un método que contiene el código de la página actual.

Objetos

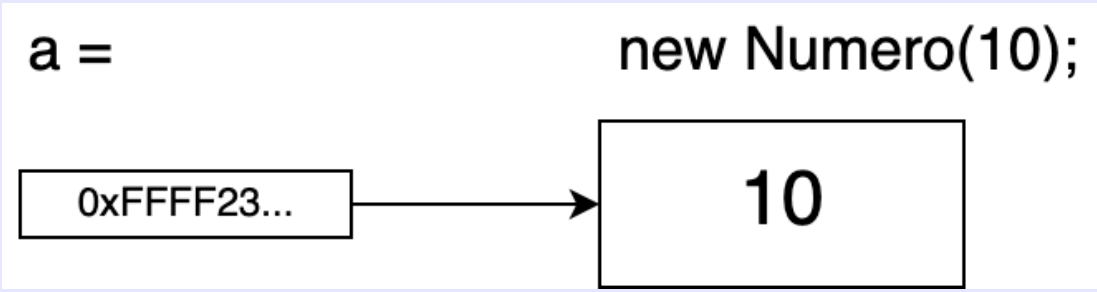
Objetos y referencias a objetos

- Los objetos se crean siempre en memoria dinámica con el operador `new`.
- Para guardar la dirección de memoria de un objeto utilizamos **referencias a objetos** (o simplemente *'referencias'*). Equivalen a los punteros en C++.
- las referencias tienen valor `null` (en minúsculas) por defecto.

En el código siguiente, 'Numero' es una clase ficticia y 'a' una referencia:

```
Numero a = new Numero(10);
```

que podemos representar gráficamente así:

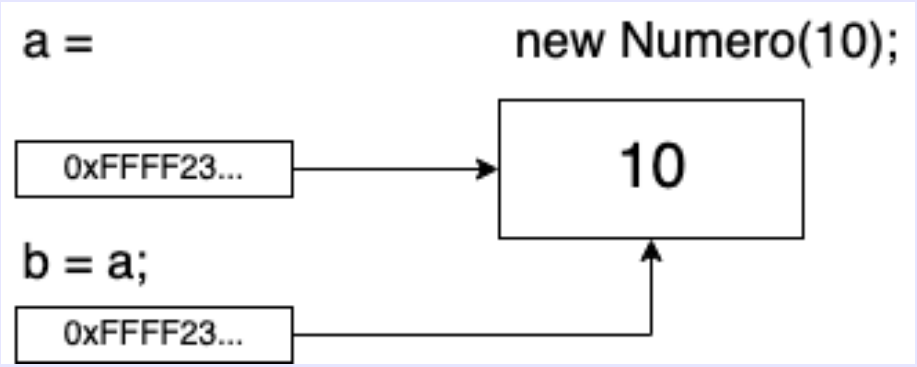


Objetos

Asignación de referencias

```
Numero a = new Numero(10);
Numero b = a;
```

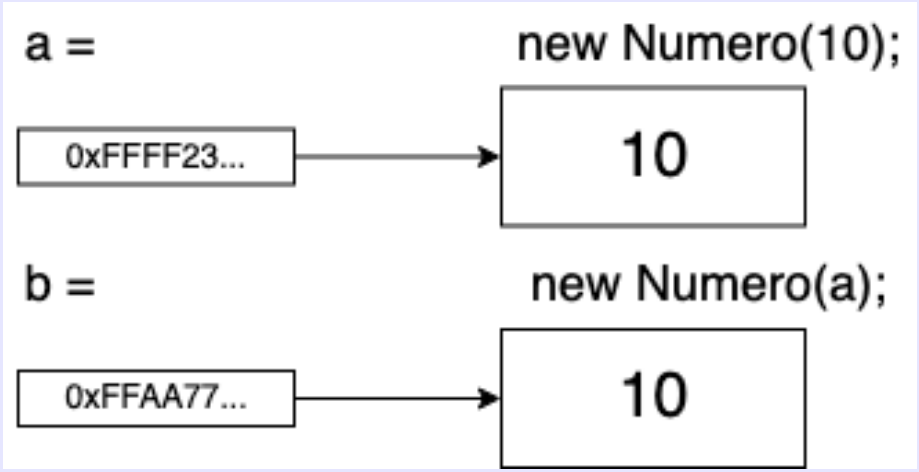
hace que ’b’ apunte a la misma instancia, es decir, el mismo objeto, la misma zona de memoria que ’a’. Tenemos dos referencias apuntando a un único objeto.



Copia de objetos

Para duplicarlo habría que crear un nuevo objeto con `new`, posiblemente invocando a un *constructor de copia*, si la clase lo tiene:

```
Numero a = new Numero(10);
Numero b = new Numero(a);
```



Objetos

La clase *Object*

La clase *Object* representa a *todos los objetos* de Java. Cualquier objeto de cualquier clase es también un objeto de la clase Object.

```
Object obj = new Numero(10); // Ok
obj = new Persona(); // Ok
```

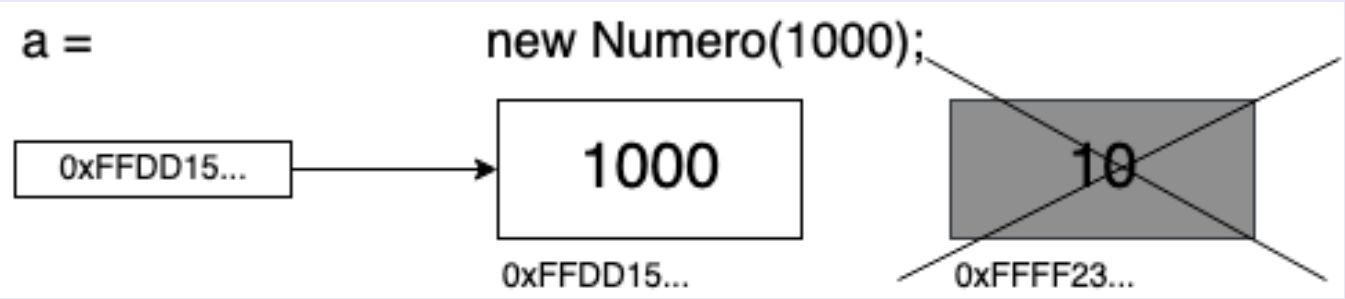
Así, podemos utilizar referencias de tipo *Object* para apuntar a cualquier tipo de objeto.

David Rizo, Pedro J. Ponce de León

El recolector de basura

En Java no es necesario liberar explícitamente la memoria de objetos que ya no necesitamos. Lo hace el *recolector de basura* o *garbage collector* cuando un objeto se queda sin referencias que apunten a él:

```
Numero a = new Numero(10);
a = new Numero(1000);
```



Objetos

operador instanceof

La expresión

```
objeto  instanceof Clase
```

devuelve cierto si 'objeto' apunta a un objeto de la clase 'Clase', y falso en caso contrario.

Casting (conversión)

Es similar a C++:

```
int x = 10;
float f = (float) x;
```

Dado un objeto cualquiera, también podemos usar el operador de conversión para asignarlo a una referencia de tipo conocido:

```
Object cualquiera;
MiClase obj = (MiClase) cualquiera;
```

Nota: para hacer la conversión sin riesgo, debemos estar seguros de que 'cualquiera' apunta a un objeto de tipo 'MiClase'.

Objetos

operador punto (’.’)

Para acceder a los atributos o invocar a los métodos de una clase usando una referencia a objeto, usamos el operador ’.’:

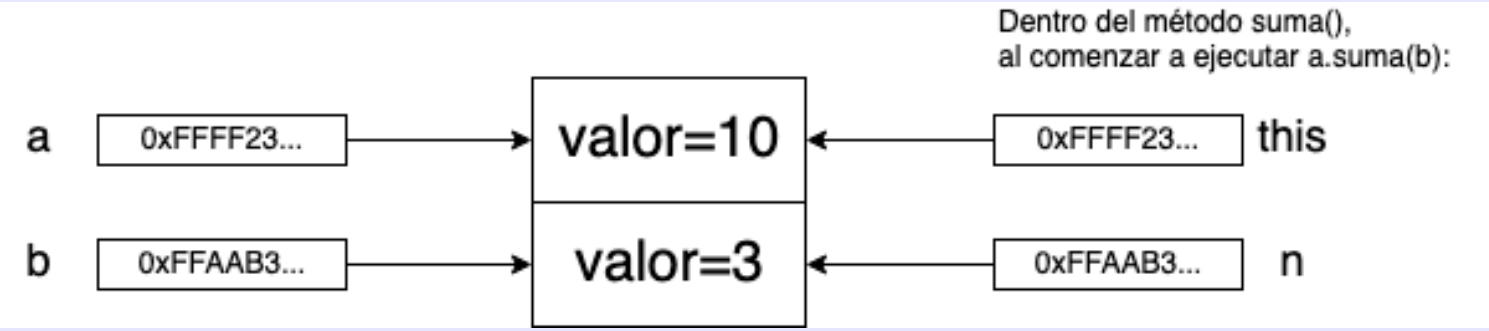
```
Numero a = new Numero(10);
Numero b = new Numero(100);

a.valor = 3;
b.valor = 4;
a.suma(b);
a.incrementa(3);
```

Objetos

La referencia `this` [3]

Igual que en C++, dentro de un método no estático podemos hacer referencia al objeto mediante el cual invocamos al método con la referencia `this`.

```
public class Numero {
  private int valor;
  public Numero(int valor) { this.valor = valor; }
  public void suma(Numero n) { this.valor += n.valor; }
}
...
Numero a = new Numero(10);
Numero b = new Numero(3);
a.suma(b);
```



Objetos

ACTIVIDAD

Ejecuta

```
> java Clase 3
```

para explorar el resultado del código en la página anterior (no te lo recordaremos más, recuerda hacerlo cuando veas en una página el número entre corchetes).

Objetos

Comparación

La expresión

```
a==b
```

está comparando referencias, es decir, direcciones de memoria. Para comparar el contenido de dos objetos debemos hacer:

```
a.equals(b)
```

El método 'equals'

Si queremos compara objetos de una clase creada por nosotros, debemos implementar el método 'equals'.

```
public boolean equals(Object obj)
```

El argumento de equals es una referencia a objeto de clase 'Object'. Esto implica que al método equals se le puede pasar un objeto de cualquier clase (aunque normalmente será uno del mismo tipo del objeto con que queremos compararlo).

Implementación de 'equals' [4]

Para implementar el método 'equals' hay que tener en cuenta que la operación de igualdad debe cumplir las propiedades reflexiva, simétrica y transitiva y asegurarnos de que

```
x.equals(null) == false // para cualquier x no nulo
```

Además, para poder comparar los atributos del argumento con los del objeto `this`, deberemos convertir el argumento a una referencia de nuestra clase. Por tanto, toda implementación del método equals debe realizar estas comprobaciones:

```
public boolean equals(Object obj) {
  if (obj == this) return true; // las dos referencias
      // apuntan al mismo objeto
  if (obj == null) return false;
  if (!(obj instanceof MiClase)) return false;
  MiClase elotro = (MiClase) obj;
  // a partir de aqui comparar los atributos de ambos
  // objetos ('this' y 'elotro') para determinar si
  // son iguales o no.
  // !OJO! si los atributos son referencias a objetos,
  // hay que usar 'equals' para compararlos.
}
```

Objetos

Wrappers (objetos)

Cada tipo escalar tiene una clase equivalente:

`Byte`, `Integer`, `Float`, `Double`, `Char`, `Boolean`

que se inicializan

```
Integer a = null; // es nulo por defecto
a = new Integer(29);
int x = a.intValue(); // x es 29
```

Java simula la compatibildad entre los tipos escalares y sus correspondientes clases 'wrapper' mediante asignación directa entre ellos. Es lo que se conoce como 'boxing' y 'unboxing' (sig. página).

Objetos

Boxing [5]

Cuando hacemos

```
Integer b = 3;
```

internamente se está haciendo

```
Integer b = new Integer(3);
```

Unboxing [5]

y al contrario, al escribir

```
Integer b = new Integer(100);
int x = b;
```

internamente se está haciendo

```
int x = b.intValue();
```

Excepciones

Concepto

- Una excepción es un mecanismo diseñado para manejar situaciones de error alterando el flujo normal de ejecución de un programa.
- Ejemplo de excepciones son el acceso a una dirección de memoria inválida, la división por cero, o la referencia a una posición negativa en un array.
- En su forma más básica, cuando se produce la excepción el método invocado aborta su ejecución y devuelve el control al método que lo invoca, operación que se repite hasta llegar al programa principal el cual para la ejecución de la aplicación.
- Las excepciones son objetos de clases cuyo nombre suele tener la forma 'BlablaException'.

Excepciones [6]

Las dos excepciones con las que es más probable que nos encontremos son:

NullPointerException

Se lanza cuando estamos accediendo a una posición de memoria sin inicializar (para la que no se ha hecho un new). Por ejemplo:

```
Integer a, b;
if (a.equals(b)) {
// este if lanza la excepcion NullPointerException
} .....
```

ArrayIndexOutOfBoundsException

Lanzada cuando se accede a una posición inválida de un array. Por ejemplo:

```
int [] v = new int[10];
v[20] = 3;
// esto lanza la excepcion ArrayIndexOutOfBoundsException
```

Cadenas [7]

String

Java dispone de una clase para trabajar con cadenas

```
String s = new String("Hola");
```

Recordar la comparación

```
s == "Hola" // mal
s.equals("Hola") // bien
```

toString()

Todas las clases suelen tener definido el método `toString()`. Java utiliza este método para convertir un objeto de cualquier clase a cadena.

```
Float f = new Float(20);
String s = f.toString();
```

¡OJO! 'String' comienza con mayúscula. Los nombres de clases en Java se escriben normalmente con mayúscula inicial.

David Rizo, Pedro J. Ponce de León

Cadenas

Concatenación [8]

Las cadenas se pueden concatenar con el operador +, si mezclamos otros tipos éstos se pasan a cadena automáticamente usando el método `toString()`.

```
int i=100;
"El valor de i es = " + i;
```

Este código internamente crea 4 objetos, internamente hace

```
String s1 = new String("El valor de i es = ");
String s2 = new Integer(i).toString();
String s3 = s1.concat(s2); // que crea un objeto nuevo
```

StringBuilder

Para evitar la creación de tantos objetos podemos usar `StringBuilder`

```
StringBuilder sb = new StringBuilder();
sb.append("El valor de i es = ");
sb.append(i);
sb.toString(); // devuelve un objeto String
```

Arrays [9]

Los arrays se definen como los arrays dinámicos de C++

```
Integer [] v; // v es un puntero a null
```

Los arrays son objetos. Se inicializa así:

```
v = new Integer[100];
```

Ahora los contenidos de `v`, es decir `v[0], v[1],` , etc... son `null`, se deben inicializar

```
// v.length es la longitud reservada para el array
for (int i=0; i<v.length; i++) {
  v[i] = new Integer(0);
  //  v[i] = 0 (equivalente por el boxing)
}
```

Se pueden crear literales array reservando también memoria

```
int [] v = new int []{1,2,3,4,5}; // contiene 5 enteros
```

y se pueden copiar manualmente usando un bucle o con el método estático `arraycopy` de la clase `System`

```
int [] origen = new int []{1,2,3,4,5};
int [] destino = new int[origen.length];
System.arraycopy(origen, 0, destino, 0, origen.length);
```

Métodos, paso por valor

Métodos

A las funciones miembro de una clase se les llama métodos.

Parámetros

Todos los parámetros se pasan por valor

```
void F(int a, String x, int [] v) {
  a=10; // este cambio no afecta al valor original
  x = "Hola"; // no afecta al original
  v[2] = 7000;
  // lo que se ha pasado por valor es
  // el puntero a v, v[2] se cambia en el original
}
```

Métodos, paso por valor

ACTIVIDAD [10]

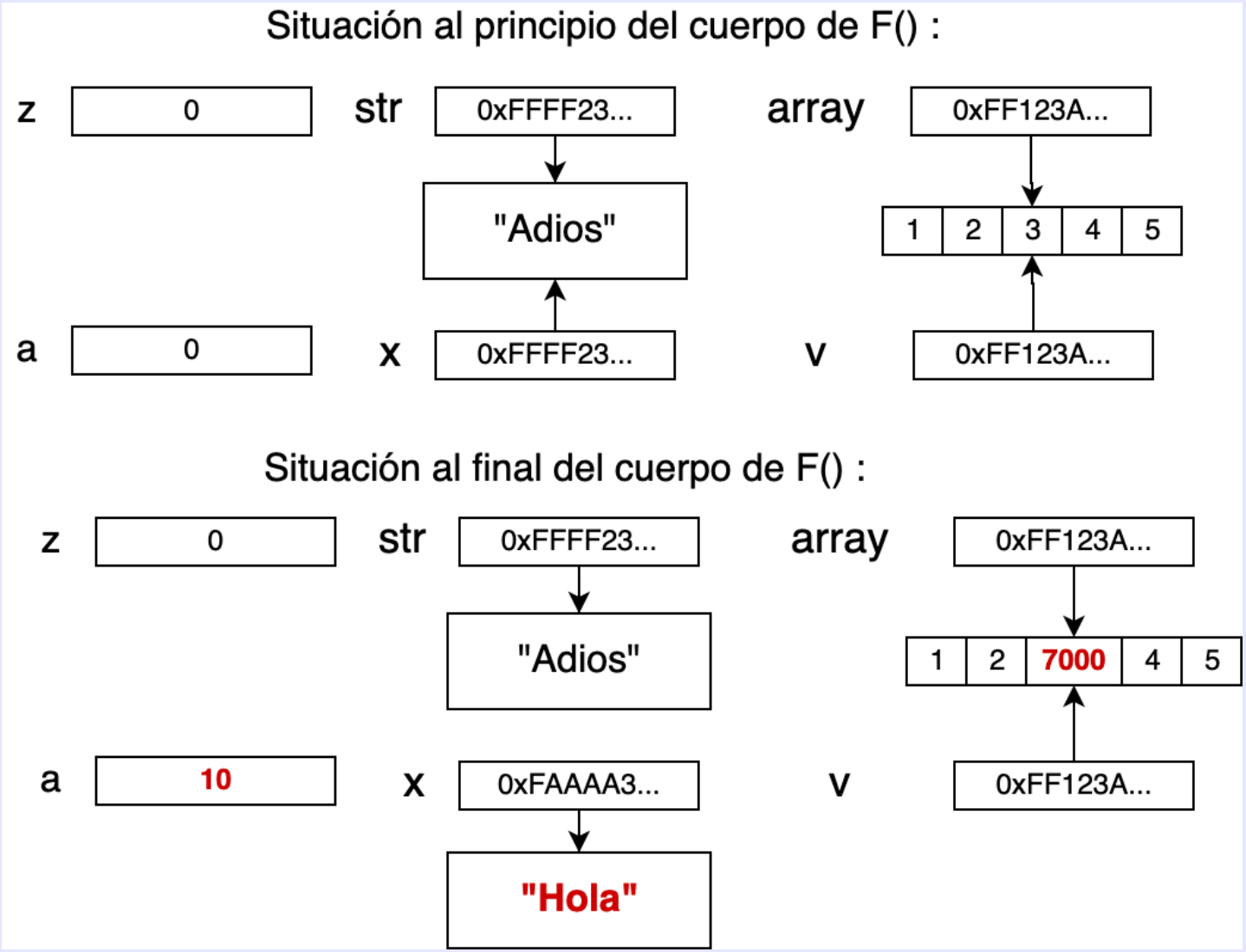
Dada la función F() definida en la página anterior y el siguiente fragmento de código

```
int z = 0;
String str = "Adios";
int [] array = new int []{1,2,3,4,5};
F(z,str,array);
```

- Intenta predecir el valor de 'z', 'str' y 'array[2]' tras la llamada a F().
- Ejecuta la actividad mediante la orden `java Clase 10` en la carpeta `codigo` y comprueba si acertaste.
- Consulta el esquema de la siguiente página para comprender lo sucedido.

Métodos, paso por valor

ACTIVIDAD [10]



Control de flujo [11]

En general no cambia respecto a C++. A partir de la versión 1.7 de Java existe la posibilidad de usar cadenas en los `switch`.

Bucles

Para recorrer arrays, por ejemplo, usaremos:

```
String v[] = new String[] {"Azul", "Verde", "Rojo"};

for (int i=0; i<v.length; i++) {
  System.out.println(v[i]);
}

for (String color: v) {
  System.out.println(color); // imprime un color por linea
}
```

Las instrucciones 'if', 'while' y 'do-while' se usan igual que en C++.

Paquetes

package

Las clases se distribuyen físicamente en directorios. Éstos constituyen lo que se denomina `package`

Para que una clase esté en un paquete hay que:

- Guardar el fichero de la clase en el directorio del paquete
- Declarar al inicio del fichero el `package` al que pertenece, separando directorios (paquetes) con puntos

```
package prog3.ejemplos;
class Ejemplo {
}
```

El fichero `Ejemplo.java` se debe guardar en el directorio `prog3/ejemplos`.

Paquetes

Modularización

No es obligatorio usar paquetes, pero es recomendable. Si queremos usar una clase de otro paquete debemos incluirla, tanto si es nuestra o de una librería

```
package prog3.ejemplos;
// clase de libreria de Java
import java.util.ArrayList;

// clase nuestra de otro paquete
import prog3.otrosejemplos.Clase;

// Lo siguiente incluye todas las clases de prog3.practicas.
// Por trazabilidad, es mejor no usar el *
import prog3.practicas.*;
```

Sólo se incluyen por defecto todas las clases de `java.lang` y por tanto no es necesario incluirlas explícitamente

```
// Lo siguiente no es necesario, todos las clases
// de java.lang estan incluidas por defecto
import java.lang.String;
```

Paquetes

ACTIVIDAD

- Localiza en la carpeta `codigo` el paquete `prog3` y dentro de el las clases `Ejemplo` y `Clase` que aparecen en las páginas anteriores.
- Revisa su código fuente y en particular las instrucciones `package` e `import`.
- Desde la carpeta `codigo`, compila la clase *Ejemplo* : `javac prog3/ejemplos/Ejemplo.java`
- ¿Dónde se ha guardado el archivo `Ejemplo.class`?
- El compilador también ha compilado el archivo `prog3/otrosejemplos/Clase.java` sin tú pedírselo. ¿A qué crees que se debe?

API de Java

API (Application Programming Interface)

Java dispone de una extensa librería de clases que se puede consultar en `http://download.oracle.com/javase/8/docs/api/overview-summary.html`

- Esta estructurada en paquetes.
- El paquete `java.lang` contiene las clases básicas del lenguaje
- El paquete `java.util` contiene clases para crear colecciones de objetos: vectores dinámicos (listas), conjuntos, mapas, etc.

David Rizo, Pedro J. Ponce de León

lsi

API de Java

Vectores dinámicos [12]

Como medio de almacenamiento lineal dinámico usaremos la clase `ArrayList`.

```
import java.util.ArrayList; // necesario
.....
ArrayList<Integer> v = new ArrayList<Integer>();
v.add(87); // esto internamente hace v.add(new Integer(87))
v.add(22); // hace mas grande el vector

ArrayList<String> sv = new ArrayList<String>();
sv.add("PROG3");
sv.add("JAVA");

// get() devuelve un objeto del tipo especificado
Integer a = v.get(0);
String s = sv.get(1); // "JAVA"

v.get(100);      // lanza una excepcion (error de ejecucion)
System.out.println(v.size()); // size() devuelve el tamanyo
```

CLASSPATH

ClassNotFoundException

Esta excepción aparece a veces al intentar iniciar un programa Java. Antes de comenzar a ejecutar el programa principal, la máquina virtual debe cargar todos los archivos .class necesarios. Si no encuentra alguno, lanza esta excepción.

Ejemplo

```
mihome$ java Clase
Error: Could not find or load main class Clase
Caused by: java.lang.ClassNotFoundException: Clase
```

Pero ¿ dónde deben estar esos archivos? En el *classpath*.

CLASSPATH

classpath

El *classpath* es la lista de directorios donde Java busca los archivos .class necesarios para ejecutar una aplicación.
Por defecto son

- el directorio actual
- ruta de las librerías del JRE (Java Runtime Environment), donde se encuentran los archivos .class de la API de Java.

CLASSPATH

Supongamos que nuestro programa principal está compilado en un archivo llamado `Clase.class` que reside en `/home/mihome/codigo`.

Caso 1

Todas nuestras clases están en un mismo directorio. No usamos `package`. Desde ese directorio,

```
/home/mihome/codigo> java Clase
```

(Ya has comprobado que esto funciona si has realizado alguna de las actividades)

CLASSPATH

Caso 2

Ejecutamos `java` desde un directorio distinto al que contiene nuestros `.class`. Hay que definir el *classpath*:

Opcion 1

Definir la variable de entorno **CLASSPATH** con la lista de directorios donde están los .class (mejor usar rutas absolutas)

```
.../otrodirectorio> export CLASSPATH=/home/mihome/codigo

.../otrodirectorio> java Clase
```

Opcion 2

Usar la opcion **-cp** o **-classpath** de `java`:

```
.../otrodirectorio> java -cp /home/mihome/codigo Clase
```

CLASSPATH

Caso 3

Los .class están repartidos en varios directorios.

```
> export CLASSPATH=/home/mihome/milibjava:/home/mihome/codigo
> java Clase
```

o bien usar **-cp**. OJO: la opcion '-cp' anula a CLASSPATH. Se debe usar una u otra, pero no ambas a la vez.

Paquetes y *classpath*

Cuando nuestras clases están organizadas en paquetes.
Supongamos que tenemos lo siguiente:

Estructura del proyecto

modelo/MiClase.java:

```
package modelo;
public class MiClase {...}
```

mains/Main.java:

```
package mains;
public class Main {...}
```

modelo/m2/OtraClase.java:

```
package modelo.m2;
public class OtraClase {...}
```

classpath deberá contener **el directorio padre** de la estructura de paquetes.

Paquetes y *classpath*

Supongamos que ese directorio del proyecto es `/home/mihome/codigo`. Si quiero usar **OtraClase** en Main.java:

```
import modelo.m2.OtraClase;
```

Al ejecutar

```
.../otrodir>java -cp /home/mihome/code mains.Main
```

para poder ejecutar la clase Main, 'java' buscará en los directorios del *classpath* un directorio `mains` y dentro de éste el archivo `Main.class`, y un directorio `modelo/m2` y dentro de éste el archivo `OtraClase.class`.

Paquetes y *classpath*

ACTIVIDAD

En la carpeta `codigo` encontrarás los paquetes `modelo` y `mains` indicados en las páginas anteriores. Tu objetivo va a ser ejecutar el programa principal desde otro directorio.

- Echale un vistazo al código de las tres clases que contienen los paquetes.
- Cambia a otro directorio.
- Asigna el *classpath* de manera que apunte a la carpeta `codigo` (usa rutas absolutas). Puedes usar la variable de entorno CLASSPATH o la opción -cp de la máquina virtual.
- Ejecuta el programa principal que se encuentra en la clase *Main* como se indica en la página anterior.

Documentación I

Javadoc

En java se utiliza un formato basado en anotaciones embebido en comentarios. Éstos se inician con `/**` y los tipos de anotaciones comienzan por `@`:

```
package paquete;
/**
 * Clase de ejemplo: documentamos brevemente el cometido
 * de la clase
 * @author drizo
 * @version 1.8.2011
 */
 public class Ejemplo {

 /**
  * Esto es un campo que vale para ...
  */
  private int x;

  private int y; // esto no sale en el javadoc
```

Documentación II

```
  /**
   * Constructor: hace esta operacion....
   * @param ax Es el radio de ...
   * @param ab Si es cierto pasa ...
   * '@param' documenta argumentos de un metodo.
   * Su formato es '@param <id> <descripcion>'
   * <id> debe coincidir con el nombre de alguno de los argumentos
   */
    public Ejemplo(int ax, boolean ab) {
        ....
    }

  /**
   * Getter.
   * @return un valor siempre mayor que cero...
   */
    public double getX() {
        return x;
    }
}
```

Documentación III

Generación

La documentación en html se genera mediante la orden

```
javadoc -d doc paquete otropaquete
```

genera un directorio `doc` con la documentación de las clases en los paquetes `paquete` y `otropaquete`.

ACTIVIDAD

- Prueba a generar la documentación del paquete `mains` que se encuentra en la carpeta `codigo` que acompaña a este documento.
- Para ver el resultado abre el archivo generado `doc/index.html` en tu navegador.

Archivos JAR

jar es una utilidad de Java (similar a **tar**) para empaquetar en un único fichero con extensión **.jar** una estructura de directorios. Se suele usar para archivos .class.

JAR

Para empaquetar, desde el directorio de trabajo:

```
/home/mihome/code> jar cvf MisClase.jar mains modelo
```

Ahora podemos llevarnos MisClases.jar donde queramos (p. ej. /home/mihome/libs) y, desde cualquier lugar:

```
> java -cp /home/mihome/libs/MisClases.jar mains.Main
```

Para ver el contenido de un archivo .jar:

```
> jar tvf MisClases.jar
```

ANT

ant

ant es una herramienta para automatizar las diversas tareas relativas a la compilacion, generación de documentación, archivos jar, etc. Es similar a 'make'. En *Programación 3* lo usaremos como parte del script de corrección de prácticas. No es necesario que conozcas su funcionamiento, pero si tienes curiosidad...

Tutorial de 'ant'

En el enlace siguiente tienes un breve tutorial en castellano:
`http://www.chuidiang.com/java/herramientas/ant.php`