Complementos de Programación

Tema 1: Introducción a la Programación Dirigida a Objetos

Andrés Cano Utrera
Departamento de Ciencias de la Computación e I.A.

DECSAI

ugr Universidad de Granada

ETSIIT Universidad de Granada

Curso 2015-16

Índice I

Índice II

Contenido del tema

Origen de Java I

Lenguaje Java

Java es un lenguaje de programación potente y versátil para el desarrollo de software para dispositivos móviles, ordenadores de escritorio y servidores.

- Deriva su sintaxis de C y sus características orientadas a objetos las deriva casi todas de C++.
- La primera versión de Java (llamada Oak) fue desarrollada en Sun Microsystems Inc. en 1991 (por James Gosling, Patrick Naughton, Chris Warth, Ed Frank y Mike Sheridan).
- En 1995 el lenguaje se renombró como Java y se rediseñó para el desarrollo de aplicaciones Web.
- En 2010, Sun Microsystems fue comprada por Oracle.

Origen de Java II

- El impulso inicial de Java no fue Internet, sino la necesidad de un **lenguaje independiente de la plataforma** para crear software para dispositivos electrónicos (microondas, controles remotos, etc), lo cual con lenguajes como C y C++ era complicado de hacer (se necesitaba un compilador para cada CPU).
- Mientras se elaboraban los detalles de Java, apareció un segundo y más importante factor: La **World Wide Web**. La red también exigía programas portables. La WWW hizo que Java fuese impulsado al frente del diseño de los lenguajes de programación.

Importancia de Java para Internet

Importancia de Java para Internet

- Internet ha ayudado a Java a situarse como líder de los lenguajes de programación y por otro lado Java ha tenido un profundo efecto sobre Internet.
- La razón es que Java **amplía el universo de objetos** que pueden moverse libremente por el ciberespacio: programas dinámicos autoejecutables.

Tipos de programas Java

Tipos de programas Java

Java es un lenguaje de propósito general, con un conjunto completo de características, que puede ser usado para desarrollar aplicaciones muy robustas. Permite crear varios tipos de programas:

- **Aplicaciones de escritorio**
- **Applets**
- **Servlets**
- **Middlets** y aplicaciones para Android

Por ejemplo, fue usado por la NASA para programar un robot autónomo enviado a Marte
`http://java.sys-con.com/node/39220`

¿Por qué Java?: Ventajas de Java frente a otros

¿Por qué Java?: Ventajas de Java frente a otros

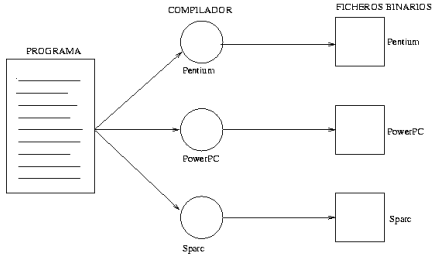
- Las posibilidades que ofrece para facilitar la ejecución de programas en **diferentes plataformas**
- Lenguaje orientado a objetos
- De uso extendido (aplicaciones relacionadas con web). De los más usados en el ámbito de las telecomunicaciones.
- Facilidad para programar con hebras
- Gran número de bibliotecas disponibles (interfaz gráfico, trabajo en red, clases de colección, etc)

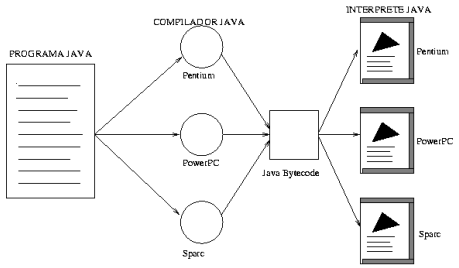
La máquina virtual de Java y el bytecode I

- El **compilador de Java** traduce código fuente Java en *bytecode*.

  *La salida del **compilador de Java** no es código ejecutable, sino un código binario intermedio (**bytecode**) que contiene un conjunto de instrucciones altamente optimizadas, y que luego podrán ejecutarse mediante una **máquina virtual** (el intérprete de Java).*

La máquina virtual de Java y el bytecode II





La máquina virtual de Java y el bytecode III

- El *bytecode* consiste en las intrucciones máquina de una *pseudo CPU*: CPU virtual cargada en memoria.
- La **Máquina Virtual de Java** (JVM) emula esta CPU virtual. Es esencialmente una máquina que es capaz de ejecutar un ejecutable Java.
  - Proporciona el entorno de ejecución para un programa Java ejecutable (bytecode).
  - Proporciona un intérprete de bytecode.
  - Proporciona un verificador de código que confirma la valided del bytecode antes de traducirlo y ejecutarlo
  - Contiene módulos para controlar la seguridad, memoria y manejo de hebras, etc

La máquina virtual de Java y el bytecode IV

- Gracias al uso de la JVM y el bytecode, se controlan los problemas de **seguridad** y **portabilidad** tan importantes en la red.
  - **Seguridad**: La JVM controla el código del bytecode, evitando que provoque efectos no deseados en el sistema.
    - Impedir infectarse con virus en computadora local.
    - Impedir el acceso a ficheros confidenciales.
    - Además Java no usa punteros evitando el acceso ilegal a memoria.
  - **Portabilidad**: Código ejecutable portable a diferentes plataformas.
    - Sólo se requiere una JVM para cada plataforma para ejecutar bytecode en cualquier sitio.
- A pesar de ser interpretado, la velocidad no es mucho peor que en un lenguaje compilado.
- Además nada impide que el programa sea compilado en código nativo.

Características de Java I

- *Pequeño*: El código ejecutable generado por Java es muy pequeño.

  Un programa Hello World ocupará unos pocos bytes. El entorno de ejecución necesario para ejecutar código compilado Java ocupa habitualmente menos de 1MB de memoria.

- *Simple*: Fácil de aprender y de utilizar de forma eficiente.

  Es parecido a C++, pero evita los conceptos más confusos de C++ (punteros, referencias, registros, typedef, macros, necesidad de liberar memoria), o bien los implementa de forma más clara y agradable. Es un lenguaje que podría introducirse como primer lenguaje de programación a un estudiante.

Características de Java II

- *Orientado a objetos*
  - Todo el código debe estar encapsulado en clases.
  - Es un lenguaje adaptado a la PDO ya que no permite variables globales, structs, unions como en C y C++.
  - Los tipos primitivos no se declaran como objetos para mantener la eficiencia, aunque existen clases Wrapper (envoltura) para estos tipos.
- *Interpretado y de alto rendimiento*: El bytecode fue diseñado con cuidado para ser sencillo de traducir a código máquina nativo para conseguir un rendimiento alto.
  - La mayoría de las JVMs implementan técnicas de optimización, como la ejecución *Just-in-Time* (JIT) (traducción a código máquina del bytecode, antes la ejecución del código).
- *Portable, Arquitectura neutral*: Independiente de la plataforma (máquina y sistema operativo).

Características de Java III

- *Robusto*: Los programas se comportarán de manera predecible bajo diversas condiciones. La robustez se consigue en Java con:
  - Se comprueba el código en tiempo de compilación (es un lenguaje fuertemente tipado), lo que permite encontrar pronto los errores de un programa.
  - La JVM verifica el bytecode en tiempo de ejecución, y mantiene la integridad del espacio de memoria que usa cada aplicación.
  - No se deja al programador la tarea de reservar y liberar memoria: Posee un **recolector automático de basura**.
  - Las situaciones excepcionales (división por 0, archivo no encontrado, etc) son manejadas con la **gestión de excepciones**.
- *Seguro*
- *Multihebra*: Permite escribir programas que hacen varias cosas a la vez.

Características de Java IV

- *Distribuido*: Diseñado para el entorno distribuido de Internet (trabaja con TCP/IP). Permite ejecutar métodos en máquinas remotas y acceder a archivos remotos (URLs).
- *Dinámico*: Java no conecta todos los módulos de una aplicación hasta el tiempo de ejecución.

Programación orientada a objetos I

Paradigmas de programación

- **Programas orientados a proceso**: Organizados conceptualmente en base al código (lo que está sucediendo). El código actúa sobre los datos.
- **Programas orientados a objetos**: Organizados en base a los datos y a un conjunto de interfaces bien definidos a esos datos. Los datos controlan el acceso al código.
  La PDO es la base de Java. Todos los programas en Java son orientados a objetos.

Programación orientada a objetos

Abstracción

- Es un elemento esencial de la PDO: Significa ignorar los detalles de cómo funcionan los objetos, y conocer sólo cómo se usan.
- Los objetos se pueden tratar como entidades que responden a mensajes que les dicen que hagan algo.

Programación orientada a objetos

Encapsulamiento

Es un mecanismo que permite **juntar el código y los datos** que el código manipula en una misma entidad.

- El encapsulamiento es como un **envoltorio protector** que evita que otro código que está fuera pueda acceder arbitrariamente al código o a los datos.
- El acceso al código y a los datos se hace de forma controlada a través de una **interfaz** bien definida.
- La base del encapsulamiento es la **clase**: Define la **estructura** (datos) y **comportamiento** que serán compartidos por un conjunto de objetos.
  Define una plantilla para los objetos de la clase.
- En la clase, los métodos y datos pueden definirse como **privados** o **públicos**.

Programación orientada a objetos

Herencia

Proceso mediante el cual un objeto adquiere las propiedades (datos y métodos) de otro.

- La mayor parte del conocimiento se puede organizar en clasificaciones jerárquicas.
- Usando una jerarquía de objetos, un objeto sólo necesita definir aquellas cualidades que lo hacen único dentro de su clase.
- Las subclases heredan todos los atributos de cada uno de sus antecesores en la jerarquía de clases.

Programación orientada a objetos

Polimorfismo

Significa que se usa un **mismo interfaz**, pero **varios métodos** distintos.
Permite usar una misma interfaz para una clase general de objetos.

- Pila de números enteros, flotantes y char: En un lenguaje no orientado a objetos hay que crear tres conjuntos de rutinas diferentes y con nombres diferentes.
- En Java se puede especificar un conjunto de rutinas para las pilas con el mismo nombre para todos los tipos.
- El compilador es el que selecciona la acción específica (método) que se debe aplicar en cada situación.

Primer programa en Java

Primer programa en Java

```
/*
  Este es un primer programa de prueba.
  Este archivo se llama "Example.java"
*/
class Example {
  // El programa comienza con una llamada a main()
  public static void main(String args[]) {
    System.out.println("Hola mundo.");
  }
}
```


Primer programa en Java

Cuestiones sobre nomenclatura

- En Java, un fichero fuente contiene una o más definiciones de clase.
- El nombre de un fichero fuente suele ser el mismo de la clase que contenga.
- Los ficheros de programas en Java se guardarán con la extensión .java
- Debemos asegurarnos que coinciden mayúsculas y minúsculas en nombre de fichero y clase.

Primer programa en Java

Compilación del programa con jdk

- Se ejecutará la orden:

  `javac Example.java`

- El compilador **javac** crea un archivo llamado `Example.class` que contiene el bytecode compilado del programa.

Ejecución del programa con jdk

- Se ejecutará la orden:

  `java Example`

- La salida del programa será:

  `Hola mundo`

Análisis del primer programa de prueba I

- Comentarios de varias líneas

```
/*
  Este es un primer programa de prueba.
  Este archivo se llama "Example.java"
*/
```

- Definición de la clase

```
class Example {
```

  La definición de una clase, incluyendo todos sus miembros, estará entre la llave de apertura ({) y la de cierre (}).

- Comentario de una línea

```
// El programa comienza con una llamada a main()
```


Análisis del primer programa de prueba II

- Cabecera del método main

```
public static void main(String args[]) {
```

  - Todas las aplicaciones Java comienzan su ejecución llamando a main.
  - La palabra **public** es un *especificador de acceso* (puede usarse fuera de la clase en la que se declara).
  - Otro especificador de acceso es el **private** (puede usarse sólo en métodos de la misma clase).
  - La palabra **static** permite que main sea llamado sin tener que crear un objeto de esta clase (Example).
  - La palabra **void** indica que main no devuelve ningún valor.
  - El parámetro **String args[]** declara una matriz de instancias de la clase **String** que corresponde a los argumentos de la línea de ordenes.

Análisis del primer programa de prueba III

- Siguiente línea de código:

```
System.out.println("Hola mundo.");
```

  Esta línea visualiza la cadena `"Hola mundo"` en la pantalla.

Contenido del tema

Tipos de datos, variables

- Java es un lenguaje fuertemente tipado:
  - Cada variable tiene un tipo, cada expresión tiene un tipo y cada tipo está definido estrictamente.
  - En todas las asignaciones (directas o a través de parámetros) se comprueba la compatibilidad de los tipos.
- Java es más estricto que C en asignaciones y paso de parámetros.

Tipos de datos simples I

Tipos de datos simples

- No son orientados a objetos y son análogos a los de C (por razones de eficiencia).
- Todos los tipos de datos tienen un **rango definido estrictamente** a diferencia de C (por razones de portabilidad).

Tipos de datos simples II

Datos enteros

Todos los tipos enteros son con signo.

- **byte**: 8 bits. $[-128; 127]$
- **short**: 16 bits. $[-32.768; 32.767]$
- **int**: 32 bits. $[-2.147.483.648; 2.147.483.647]$
- **long**: 64 bits. $[9.223.372.036.854.775.808; 9.223.372.036.854.775.807]$

Tipos de datos en coma flotante

- **float**: 32 bits. $[3,4 \cdot 10^{-38}; 3,4 \cdot 10^{38}]$
- **double**: 64 bits. $[1,7 \cdot 10^{-308}; 1,7 \cdot 10^{308}]$

Tipos de datos simples III

Datos carácter

- Java utiliza código **Unicode** para representar caracteres.
- Es un conjunto de caracteres completamente internacional con todos los caracteres de todas las lenguas del mundo.
- **char**: 16 bits ($[0; 65536]$)
- Los caracteres ASCII van del $[0; 127]$ y el ISO-Latin-1 (caracteres extendidos) del $[0; 255]$

Datos booleanos

boolean: Puede tomar los valores **true** o **false**

Literales I

Literales enteros

- Base decimal: 1, 2, 3, etc
- Base octal: 07
- Hexadecimal: Se antepone $0x$ o $0X$.
- Long: Se añade **L**. Ej: $101L$

Literales en coma flotante

Son de doble precisión por defecto (double).

- Notación estándar: 2.0, 3.14, .66
- Notación científica: $6.022E23$
- Añadiendo al final **F** se considera float, y añadiendo **D** double.

Literales II

Literales binarios (Java 7)

Los *tipos integrales* (`byte`, `short`, `int` y `long`) pueden expresarse también usando el sistema binario

```
// Valor 'byte' de 8 bits:
byte aByte = (byte)0b00100001;

// Valor 'short' de 16 bits:
short aShort = (short)0b1010000101000101;

// Algunos valores 'int' de 32 bits:
int anInt1 = 0b10100001010001011010000101000101;
int anInt2 = 0b101;
int anInt3 = 0B101; // La B puede ir en mayuscula o minuscula.

// Valor 'long' de 64 bits (notese que debemos usar el sufijo "L"):
long aLong = 0b1010000101000101101000010100010110100001010001011010000101000101L;
```


Literales III

Caracter de subrayado en literales numéricos (Java 7)

Podemos usar cualquier número de caracteres _ entre los dígitos de un literal numérico

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =  3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Literales booleanos

true y **false**

Literales IV

Literales de carácter

- Se encierran entre comillas simples.
- Se pueden usar secuencias de escape:
  - `\141` (3 dígitos): Código en octal de la letra **a**
  - `\u0061` (4 dígitos): Código en hexadecimal (carácter Unicode) de la letra **a**
  - `\'`: Comilla simple
  - `\\`: Barra invertida
  - `\r`: Retorno de carro
  - `\t`: Tabulador `\b`: Retroceso

Literales cadena

Entre comillas dobles

Variables I

Declaración de variables

La forma básica de declaración de variables es:

```
tipo identificador [=valor][,identificador[=valor]...];
```

- `tipo` es un tipo básico, nombre de una clase o de un interfaz.
- Los inicializadores pueden ser dinámicos:

```
double a=3.0, b=4.0;
double c=Math.sqrt(a*a + b*b);
```


Variables II

- Java permite declarar variables dentro de cualquier bloque.
- Java tiene dos **tipos de ámbito**
  - De clase
  - De método
- Los ámbitos se pueden anidar aunque las variables de un bloque interior no pueden tener el mismo nombre de alguna de un bloque exterior.
- Dentro de un bloque, las variables pueden declararse en cualquier momento pero sólo pueden usarse después de declararse.

Conversión de tipos I

Conversión automática de Java

En asignaciones y paso de parámetros a métodos, puede hacerse una **conversión automática de tipos** si:

- Los dos tipos son compatibles. Ejemplo: se puede asignar **int** a un **long**
- El tipo destino es más grande que el tipo origen

Los tipos **char** y **boolean** no son compatibles con el resto.

Conversión de tipos II

Castings: Conversión de tipos incompatibles

Un ***casting*** (conversión explícita) es necesario cuando asignamos un valor a una variable de un tipo más pequeño.

```
int a;
byte b;
// ...
b = (byte) a;  // Convierte el valor de a en un byte
System.out.println((int)1.7); // Imprime 1
System.out.println((double)1 / 2);  // Imprime 0.5
```


Conversión de tipos III

Promoción de tipo automática en expresiones

Además de las asignaciones, también se pueden producir ciertas conversiones automáticas de tipo en la evaluación de expresiones.

- **short** y **byte** promocionan a **int** en expresiones para hacer los cálculos.
- Si un operador es **long** todo se promociona a **long**
- Si un operador es **float** todo se promociona a **float**
- Si un operador es **double** todo se promociona a **double**

Ejemplo de código con error de compilación:

```
byte b=50;
b = b*2; //Error, no se puede asignar un int a un byte
```


Contenido del tema

Operadores I

La mayoría de los operadores de Java funcionan igual que los de C/C++, salvo algunas excepciones (los operadores a nivel de bits de desplazamiento a la derecha).

- Los tipos enteros se representan mediante codificación en *complemento a dos*: los números negativos se representan invirtiendo sus bits y sumando 1.
- Desplazamiento a la derecha: `valor >> num`
  Ejemplo 1

  ```
  int a=35;
  a = a >> 2; // ahora a contiene el valor 8
  ```

  Si vemos las operaciones en binario:

  ```
  00100011      35
  >>2
  00001000      8
  ```

Operadores II

Este operador >> rellena el nuevo bit superior con el contenido de su valor previo:
Ejemplo 2

```
11111000       -8
>>2
11111100       -4
```

- Desplazamiento a la derecha sin signo: `valor >>> num`
  Rellena el bit superior con cero.
  Ejemplo

```
int a=-1;
a = a >>> 24;
```

Si vemos las operaciones en binario:

```
11111111 11111111 11111111 11111111   -1 en binario como entero
>>>24
00000000 00000000 00000000 11111111   255 en binario como entero
```


Tabla de precedencia de operadores I

Tabla de precedencia de operadores

```
.      []     ()
++     --     !       ~
*      /      % (se puede aplicar a flotantes)
+      -
<<     >>     >>>
<      >      <=      >=
==     !=
&
^
|
&&
||
?:
=      op=    *=    /=     %=     +=     -=     etc.)
```


Contenido del tema

Sentencias de selección I

De nuevo son prácticamente idénticas a las de C/C++ salvo en las sentencias `break` y `continue`

Sentencia de selección if/else

```
if( Boolean ) {
   sentencias;
}
else {
   sentencias;
}
```

Sentencias de selección II

Sentencia de selección switch

```
switch( expresion ) {
   case valor1:
      sentencias;
      break;
   case valor2:
      sentencias;
      break;
   default:
      sentencias;
      break;
}
```


Sentencias de selección III

Strings en sentencias switch (Java 7)

Es posible usar un objeto `String` en una sentencia `switch`

```
public String getTypeOfDayWithSwitchStatement(String dayOfWeekArg) {
     String typeOfDay;
     switch (dayOfWeekArg) {
         case "Monday":
             typeOfDay = "Start of work week";
             break;
         case "Tuesday":
         case "Wednesday":
         case "Thursday":
             typeOfDay = "Midweek";
             break;
         case "Friday":
             typeOfDay = "End of work week";
             break;
         case "Saturday":
         case "Sunday":
             typeOfDay = "Weekend";
             break;
         default:
             throw new IllegalArgumentException("Invalid day of the week: " + dayOfWeekArg);
     }
     return typeOfDay;
}
```


Sentencias de iteración I

Bucle for

```
for( expr1 inicio; expr2 test; expr3 incremento ) {
   sentencias;
}
```

Bucle while

```
while( Expresion booleana ) {
   sentencias;
}
```

Bucle do/while

```
do {
   sentencias;
}while( Expresion booleana );
```


Tratamiento de excepciones I

Tratamiento de excepciones: try-catch-throw-throws-finally

```
try {
   sentencias;
}
catch( Exception ) {
   sentencias;
}
finally{
   sentencias;
}
```

Sentencias de salto I

Las sentencias `break` y `continue` se recomienda no usarlas, pues rompen con la filosofía de la programación estructurada:

Sentencia de salto break

Tiene tres usos:

- Para terminar una secuencia de sentencias en un **switch**
- Para salir de un bucle
- Como una forma de goto: `break etiqueta`

Sentencias de salto II

Sentencia de salto continue

Tiene dos usos:

- Salir anticipadamente de una iteración de un bucle (sin procesar el resto del código de esa iteración).
- Igual que antes pero se especifica una etiqueta para describir el bucle que lo engloba al que se aplica.

Ejemplo

```
uno: for( ) {
   dos: for( ){
      continue;      // sigue en el bucle interno
      continue uno; // sigue en el bucle principal
      break uno;     // sale del bucle principal
   }
}
```


Sentencias de salto III

Sentencia return

Tiene el mismo uso de C/C++

Ejemplo

```
int metodo(){
   if( a == 0 )
      return 1;
   return 0; // es imprescindible
}
```


Contenido del tema

Arrays: introducción I

Hay algunas diferencias en el funcionamiento de los arrays y matrices respecto a C y C++

Declaración de un array

```
tipo nombre_array[];
```
o bien
```
tipo[] nombre_array; // Forma preferida
```

Esto sólo declara `nombre_array` como array de `tipo`, y le asigna **null**.

Reserva de la memoria para los elementos

```
nombre_array=new tipo[tamaño];
```
Esto hace que se inicialicen a 0 todos los elementos.

Arrays: introducción II

Declaración y reserva al mismo tiempo

```
int[] array_int=new int[12];
```

Inicialización con literales

```
int[] array_int={3,2,7};
```

El intérprete de Java comprueba siempre que no nos salimos de los índices del array.
Se genera una excepción **ArrayIndexOutOfBounds** si lo hacemos.

Ejemplo de uso arrays

Ejemplo de uso de arrays

```
import java.io.*;

public class TestScoreAverage {
    public static void main(String[] args) {
        final int NUMERO_ESTUDIANTES = 5;
        int[] notas = new int[NUMERO_ESTUDIANTES];
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                    System.in));
            for (int i = 0; i < NUMERO_ESTUDIANTES; i++) {
                System.out.print("Introduce notas para estudiante #" + (i + 1) + ": ");
                String str = reader.readLine();
                notas[i] = Integer.parseInt(str);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        int total = 0;
        for (int i = 0; i < NUMERO_ESTUDIANTES; i++) {
            total += notas[i];
        }
        System.out.println("Nota media " + (float) total / NUMERO_ESTUDIANTES);
    }
}
```


Ejemplo de uso arrays

Salida del programa:

```
Introduce notas para estudiante #1: 7
Introduce notas para estudiante #2: 8
Introduce notas para estudiante #3: 9
Introduce notas para estudiante #4: 6
Introduce notas para estudiante #5: 5
Nota media 7.0
```

Cálculo de la longitud de un array

El campo (público) `length` de un array especifica la longitud del array.

Ejemplo de uso del campo length de un array

```
public class ArrayLengthApp {
   public static void main(String[] args) {
      final int TAMANO = 5;
      int[] intArray = new int[TAMANO];
      float[] floatArray = {5.0f, 3.0f, 2.0f, 1.5f};
      String[] diasSemana = {"Domingo", "Lunes", "Martes",
          "Miercoles", "Jueves", "Viernes", "Sabado"};

      System.out.println("Longitud intArray: " + intArray.length);
      System.out.println("Longitud floatArray: " + floatArray.length);
      System.out.println("Numero de dias en una semana: " +
                         diasSemana.length);
   }
}
```

Salida del programa

```
Longitud intArray: 5
Longitud floatArray: 4
Numero de dias en una semana: 7
```


Copia de arrays

Para copiar el contenido de un array en otro, debemos copiar los elementos del primero de uno en uno. No sería correcto hacer lo siguiente:

```
array2 = array1;
```

Hay tres formas para copiar arrays:

- Usar un bucle para copiar los elementos uno a uno.
- Usar el método estático `arraycopy` de la clase `System`.
- Usar el método `clone`

Copiando un array con un bucle

Copia de un array con un bucle

Consiste en copiar uno a uno los elementos del array original en el destino utilizando un bucle

```
int[] arrayFuente = {2, 3, 1, 5, 10};
int[] arrayDestino = new int[arrayFuente.length];
for (int i = 0; i < arrayFuente.length; i++) {
   targetArray[i] = arrayFuente[i];
}
```


Copiando un array con arraycopy

Copiando un array con arraycopy

Consiste en usar el método `arraycopy` de la clase `java.lang.System`.

```
arraycopy(arrayFuente, posFuente, arrayDestino, posDestino, longitud);
```

- `arrayFuente`: array original
- `posFuente`: posición del array original a partir de la cual copiamos
- `arrayDestino`: array destino de la copia.
- `posDestino`: posición en el array destino donde empiezan a copiarse los elementos
- `longitud`: número de elementos a copiar

El método `arraycopy` no reserva memoria para el array destino. Éste debe haber sido creado previamente.

Por ejemplo, el array anterior podríamos copiarlo con:

```
System.arraycopy(arrayFuente, 0, arrayDestino, 0, arrayFuente.length);
```

Copiando un array con clone I

Copiando un array con clone

El método `clone()` devuelve una copia (un clon) de un array. En este caso, el método sí reserva memoria para el array destino.

Ejemplo: Clonado un array

```
import java.util.Arrays;
public class ArrayCopyApp {
    public static void main(String[] args) {
        float[] floatArray = {5.0f, 3.0f, 2.0f, 1.5f};
        float[] copiaFloatArray = floatArray.clone();
        System.out.println(Arrays.toString(floatArray) + " - Original");
        System.out.println(Arrays.toString(copiaFloatArray) + " - Copia");
        System.out.println();
        System.out.println("Modificamos el segundo elemento del array original");
        floatArray[1] = 20;
        System.out.println(Arrays.toString(floatArray) +
                " - Original despues de la modificacion");
        System.out.println(Arrays.toString(copiaFloatArray) + " - Copia");
        System.out.println();
        System.out.println("Modificamos el tercer elemento del array copia");
        copiaFloatArray[2] = 30;
        System.out.println(Arrays.toString(floatArray) + " - Original");
        System.out.println(Arrays.toString(copiaFloatArray) +
                " - Array copia despues de la modificacion");
```


Copiando un array con clone II

```
    }
}
```


Copiando un array con clone III

```
[5.0, 3.0, 2.0, 1.5] - Original
[5.0, 3.0, 2.0, 1.5] - Copia

Modificamos el segundo elemento del array original
[5.0, 20.0, 2.0, 1.5] - Original despues de la modificacion
[5.0, 3.0, 2.0, 1.5] - Copia

Modificamos el tercer elemento del array copia
[5.0, 20.0, 2.0, 1.5] - Original
[5.0, 3.0, 30.0, 1.5] - Array copia despues de la modificacion
```


Paso de un array como parámetro de un método I

Paso de un array como parámetro de un método

Al pasar un array a un método, se pasa una referencia al array (y no una copia del array), lo que permite modificar el contenido del array dentro del método.

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Paso de un array como parámetro de un método II

Ejemplo para mostrar la diferencia del paso de parámetros de tipo primitivo y arrays

```
public class TestPassArray {
  public static void main(String[] args) {
    int[] a = {1, 2};

    // Intercambiar elementos usando el metodo intercambia
    System.out.println("Antes de llamar a intercambia");
    System.out.println("array es {" + a[0] + ", " + a[1] + "}");
    intercambia(a[0], a[1]);
    System.out.println("Despues de llamar a intercambia");
    System.out.println("array es {" + a[0] + ", " + a[1] + "}");

    // Intercambiar elementos usando el metodo intercambiaDosPrimeros
    System.out.println("Antes de llamar a intercambiaDosPrimeros");
    System.out.println("array es {" + a[0] + ", " + a[1] + "}");
    intercambiaDosPrimeros(a);
    System.out.println("Despues de llamar a intercambiaDosPrimeros");
    System.out.println("array es {" + a[0] + ", " + a[1] + "}");
  }

  /** Intercambia dos variables */
  public static void intercambia(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }
```


Paso de un array como parámetro de un método III

```
  /** Intercambia los dos primeros elementos en el array */
  public static void intercambiaDosPrimeros(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
  }
}
```

Salida del programa

```
Antes de llamar a intercambia
array es {1, 2}
Despues de llamar a intercambia
array es {1, 2}
Antes de llamar a intercambiaDosPrimeros
array es {1, 2}
Despues de llamar a intercambiaDosPrimeros
array es {2, 1}
```


Devolución de un array desde un método

Devolución de un array desde un método

A diferencia del lenguaje C, un método puede devolver un array.

Ejemplo: método que devuelve un array con los elementos invertidos de otro array

```
public static int[] invierte(int[] array) {
  int[] resultado = new int[array.length];
  for (int i = 0, j = resultado.length - 1;
       i < array.length; i++, j--) {
    resultado[j] = array[i];
  }
  return resultado;
}
```


Listas de argumentos de longitud variable I

Listas de argumentos de longitud variable

Un método puede tener una lista de argumentos de longitud variable del mismo tipo.

```
Tipo... nombreParametro
```

- Solo puede usarse un parámetro de longitud variable en un método, y deber ser el último de todos.
- Java trata los parámetros de longitud variable como un array.
- Al llamar al método, podremos usar un array o un número de argumentos variable. En el segundo caso, Java crea un array donde almacena los argumentos.

Listas de argumentos de longitud variable II

```
public class VarArgsDemo {
  public static void main(String args[]) {
    printMax(34, 3, 3, 2, 56.5);
    printMax(new double[]{1, 2, 3});
  }

  public static void printMax(double... numeros) {
    if (numeros.length == 0) {
      System.out.println("No se pasaron argumentos");
      return;
    }
    double resultado = numeros[0];
    for (int i = 1; i < numeros.length; i++)
      if (numeros[i] > resultado)
        resultado = numeros[i];
    System.out.println("El valor maximo es " + resultado);
  }
}
```

Salida del programa

```
El valor maximo es 56.5
El valor maximo es 3.0
```


Contenido del tema

Matrices multidimensionales I

Matrices multidimensionales

Al igual que en C/C++ podemos definir matrices de más de una dimensión. En Java las matrices son consideradas matrices de matrices.

- **Declaración**:

```
int matriz2D[][];
```

o bien

```
int[][] matriz2D; // Preferida
```

- **Declaración y reserva de memoria**:

```
int[][] matriz2D=new int[4][5];
```


Matrices multidimensionales II

- **Inicialización con literales**:

```
int[][] matriz2D= {
    {1, 98},
    {2, 58},
    {3, 78},
    {4, 89}
};
```

Matrices multidimensionales III

Ejemplo de matriz 2D: Sarang/ch2/MultiDimArrayApp.java

```
public class MultiDimArrayApp {

    public static void main(String[] args) {
        final int MAX_ESTUDIANTES = 50, MAX_ASIGNATURAS = 3;
        int[][] notas = new int[MAX_ESTUDIANTES][MAX_ASIGNATURAS];
        // Introducimos datos en la matriz
        for (int id = 0; id < MAX_ESTUDIANTES; id++) {
            for (int asignatura = 0; asignatura < MAX_ASIGNATURAS; asignatura++) {
                notas[id][asignatura] = (int) (Math.random() * 100);
            }
        }
        // Imprimimos la matriz
        System.out.print("Estudiante\t");
        for (int asignatura = 0; asignatura < MAX_ASIGNATURAS; asignatura++) {
            System.out.print("\t" + "Asignatura " + asignatura);
        }
        System.out.println();
        for (int id = 0; id < MAX_ESTUDIANTES; id++) {
            System.out.print("Estudiante " + (id + 1) + '\t');
            for (int asignatura = 0; asignatura < MAX_ASIGNATURAS; asignatura++) {
                System.out.print("\t" + notas[id][asignatura] + "\t");
            }
            System.out.println();
        }
    }
}
```


Matrices multidimensionales IV

Salida del programa:

```
Estudiante      Asignatura 0    Asignatura 1    Asignatura 2
Estudiante 1    21              34              60
Estudiante 2    21              0               53
Estudiante 3    38              24              50
Estudiante 4    24              93              28
Estudiante 5    61              47              31
Estudiante 6    88              27              40
.
.
.
```


Obtención de la longitud de matrices I

Obtención de la longitud de matrices

Una matriz 2D `matriz` es realmente un array en el que cada elemento es un array unidimensional.

- Por tanto, el número de filas se obtendrá con `matriz.length`.
- El número de elementos en cada fila se obtendrá con `matriz[0].length`, `matriz[1].length`, ..., `matriz[matriz.length-1].length`.

Matrices no rectangulares I

Matrices no rectangulares

En una matriz, cada fila puede tener un número distinto de elementos (*ragged array*).

```
int[][] matriz2D=new int[4][];
    matriz2D[0]=new int[1];
    matriz2D[1]=new int[2];
    matriz2D[2]=new int[3];
    matriz2D[3]=new int[2];
```

Contenido del tema

El bucle for-each I

El bucle for-each

Un bucle for-each permite recorrer una colección de objetos, tal como un array, de forma secuencial, de principio a fin.
La forma de un bucle for-each en Java es:

```
for(tipo var-iter : objetoIterable)
   sentencias;
```

`objetoIterable` debe implementar el interfaz **Iterable**.
El bucle se repite hasta que se obtienen todos los elementos de la colección.

El bucle for-each II

Ejemplo de bucle tradicional

```
int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = 0;
for(int i=0; i < 10; i++)
   sum += nums[i];
```

Versión usando bucle for-each

```
int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = 0;
for(int x: nums)
   sum += x;
```

- El nuevo bucle automatiza algunos aspectos del bucle tradicional:
  - Elimina la necesidad de establecer un contador del bucle.
  - De especificar un valor de comienzo y final.
  - De indexar manualmente el array.

El bucle for-each III

Ejemplo de bucle for-each: P92/ForEach.java

```
class ForEach {
  public static void main(String args[]) {
    int numeros[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int suma = 0;

    // usa un bucle for-each para mostrar y sumar los valores
    for(int x : numeros) {
      System.out.println("Valor: " + x);
      suma += x;
    }
    System.out.println("Suma: " + suma);
  }
}
```

El bucle for-each IV

Salida del programa:

```
Valor: 1
Valor: 2
Valor: 3
Valor: 4
Valor: 5
Valor: 6
Valor: 7
Valor: 8
Valor: 9
Valor: 10
Suma: 55
```


El bucle for-each V

Finalización anticipada del bucle

Es posible finalizar el bucle antes de llegar al final usando **break**.

Ejemplo de uso de break en bucle for-each: P93/ForEach2.java

```
class ForEach2 {
  public static void main(String args[]) {
    int suma = 0;
    int numeros[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    for(int x : numeros) {
      System.out.println("Valor: " + x);
      suma += x;
      if(x == 5) break; // parar el bucle cuando se obtiene 5
    }
    System.out.println("Suma de los 5 primeros elementos: " + suma);
  }
}
```


El bucle for-each VI

Salida del programa:

```
Valor: 1
Valor: 2
Valor: 3
Valor: 4
Valor: 5
Suma de los 5 primeros elementos: 15
```


El bucle for-each VII

Iterando sobre arrays multidimensionales

- En Java los arrays multidimensionales son realmente *arrays de arrays*.
- En un array multidimensional, el bucle for-each obtiene un array de una dimensión menor en cada iteración.

El bucle for-each VIII

Ejemplo en array multidimensional: P94/ForEach3.java

```
class ForEach3 {
  public static void main(String args[]) {
    int suma = 0;
    int numeros[][] = new int[3][5];

    // Inicializamos la matriz
    for(int i = 0; i < 3; i++)
      for(int j=0; j < 5; j++)
        numeros[i][j] = (i+1)*(j+1);

    //Calculamos la suma
    for(int x[] : numeros) {
      for(int y : x) {
        System.out.println("Valor: " + y);
        suma += y;
      }
    }
    System.out.println("Suma: " + suma);
  }
}
```


El bucle for-each IX

Salida del programa:

```
Valor: 1
Valor: 2
Valor: 3
Valor: 4
Valor: 5
Valor: 2
Valor: 4
Valor: 6
Valor: 8
Valor: 10
Valor: 3
Valor: 6
Valor: 9
Valor: 12
Valor: 15
Suma: 90
```