Tipos de datos simples		
Tipo de dato	**Formato**	**Descripción Rango**
`int`	32 bits ca2	[–2.147.483.648 a 2.147.483.647]
`byte`	8 bits ca2	[–128 a 127]
`char`	16 bits	
`short`	16 bits	[–32768 a 32767] (Big endian)
`long`	64 bits	[-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807]
`double`	64 bits	[-1.797 E308 a –4.9E-324] negativos [4.9E-324 a 1.797E308] positivos
`float`	16 bits	[-3.402E38 a –1.4E-45] negativos [1.401E-45 a 3.4E38] positivos
`boolean`	`true/false`	

Estructuras de control

Selección	Iteración	Salto
`if(condición){ sentencia1; }else{ sentencia2; }`	`do{ //cuerpo del bucle }while(condición);`	`break` (implica código no estructurado salvo en switch)
`switch (expresión){ case valor1: //sentencias break; case valor2: //sentencias break; case valorN: //sentencias break; default: // sentencias }`	`for(inicialización; condición; iteración) { //cuerpo del bucle }`	`continue` (implica código no estructurado)
	`while(condición) { //cuerpo del bucle }`	`return[expr.]`

Operadores y su prioridad		
Operador postfijo	`() [] . exp++ expr--`	+ Precedencia -
Operador unarios	`++exp --expr +exp -expr~`	
Creación y cast	`new (tipo)expr`	
Multiplicativos	`* / %`	
Aditivas	`+ -`	
Shift (desplazam.)	`<<>>>>`	
Relacional	`<><= >= instanceof`	
Igualdad	`== !=`	
Bit AND	`&`	
Bit OR exclus.	`^`	
Bit OR inclus.	`|`	
AND lógico	`&&`	
OR lógico	`||`	
Condicional	`?:`	
Asignación	`= += -= *= /= %= &= ^= <<= >>= >>>=`	

II PLAN PROPIO DE DOCENCIA.
Universidad de Sevilla

Referencia Rápida de los contenidos Java de Fundamentos de Programación II

Escuela Técnica Superior de Ingeniería.
Departamento de Ingeniería Telemática.
Grado en Ingeniería de las Tecnologías de Telecomunicación.

Clase

Una clase contiene (encapsula) miembros: atributos (variables de instancia) y métodos.

Esquema general de definición de una clase en Java

```
[acceso][final] class NombreDeLaClase
      [extends SuperClase] [implements Interfaz]{

    [acceso][final] tipo variableDeInstancia1;
    //...
    [acceso][final] tipo variableDeInstanciaN;

    [acceso][final] NombreDeLaClase (parámetros) {
          /* Constructores de la clase. */
    }

    [acceso][final] tipo nombreMetodo1(parámetros)
                [throws lista_de_excepciones]{
          /* Cuerpo del método. */
    }
    //...
    [acceso][final] tipo nombreMetodoN(parámetros)
                [throws lista_de_excepciones]{
          /* Cuerpo del método. */
    }
}
```

Ciclo de vida de un objeto

1º. **Creación del objeto**.
   a) **Declaración:** Mediante la declaración se crea una variable (`var`) para hacer referencia a un objeto (No crea el objeto). Sintaxis:
   `NombreDeLaClase var = null;`
   b) **Instanciación**: Crea un objeto en memoria con `new` mediante la llamada al constructor de la clase y devuelve la referencia al objeto recién creado. Sintaxis:
   `var = new NombreDeLaClase();`
   c) **Inicialización:** en la llamada al constructor de la clase se inicializan todas las variables de instancia.

2º. **Uso del objeto**: Se usa el punto para hacer referencia a los miembros de un objeto: `var.miembro;`
Dentro de la clase se usa `this` para hacer referencia al propio objeto: `this.miembro;`

3º. **Eliminación del objeto**: En Java la eliminación de objetos (recogida de basura) que no se usarán más se realiza de forma automática.

Polimorfismo

Java lo implementa mediante la **sobrecarga de método**, que permite declarar dentro de una misma clase dos o más métodos con el mismo nombre y tipo devuelto, variando la lista de parámetros (número y/o secuencia de tipos). Es muy usual sobrecargar el constructor. No se permite declarar dentro de una misma clase dos métodos con el mismo nombre y misma lista de parámetros variando el tipo devuelto.

Herencia (extends)

Permite la especificación de una clase más general, llamada superclase o clase base y una clase más específica, llamada subclase o clase derivada utilizando la palabra reservada `extends`. La subclase hereda todas las variables de instancia y los métodos definidos por la superclase y añade sus propios elementos.

Declaración de una subclase

```
class NombreSubclase extends NombreSuperclase{
    /* Código de la subclase. */
}
```

Los constructores se ejecutan en orden de derivación desde la superclase a la subclase.

super: Dentro de una subclase se usa **super** para hacer referencia a su superclase. Usos:

1) Referencia a miembros de la superclase

   super.miembro;

2) Llamada al constructor de la superclase (tiene que ser la primera sentencia ejecutada dentro del constructor de la subclase):

   super (parámetros **);**

Referencias de la superclase: Una variable del tipo de la superclase puede hacer referencia a un objeto de la subclase, pero no puede acceder a miembros que no estén definidos en la superclase.

Sobrescritura de métodos: Se dice que un método de una subclase **sobrescribe** al método de su superclase, cuando en la subclase se reescribe el método con el mismo nombre y tipo que el método de la superclase. El uso de una variable referencia a la subclase invocando un método sobrecargado invoca al método de la subclase y no el de la superclase.

Sobrecarga en herencia: Cuando el método de la subclase tiene el mismo nombre que el método de su superclase, pero varía el número o tipo de sus parámetros es sobrecarga usando herencia pero no sobrescritura de método.

Selección de método dinámica: Cuando una variable (*var*) de la superclase que referencia a una subclase, se usa para invocar a un método sobrescrito, la selección de método dinámica invoca al método de la subclase:

```
SuperClase var = new Subclase();
var.metodo();
```

Clase abstracta: Es una clase que no se puede instanciar. Además de los métodos implementados puede contener métodos abstractos. Un método abstracto está declarado pero no está implementado en esta clase. El método será implementado en alguna de sus subclases. Cualquier clase que contenga algún método abstracto debe ser declarada como abstracta. Se puede declarar una clase abstracta sin ningún método abstracto.

Declaración de una clase abstracta

```
abstract class NombreDeLaClaseAbstracta{
    abstract tipo nombre( parámetros );
}
```

Interfaz

Abstracción de la funcionalidad de una clase. Declara los métodos pero no implementa ninguno. Una clase que implemente una interfaz (indicado en su declaración) debe añadir el cuerpo de los métodos declarados en la interfaz, o bien, esa clase debe ser abstracta.

Definición de la interfaz

```
interface NombreDeLaInterfaz {
    tipo var_final1 = valor;
    // …
    tipo var_finalN = valor;

    tipo_devuelto metodo1 ( parámetros );
    // …
    tipo_devuelto metodoM ( parámetros );
}
```

Por defecto, los atributos en una interfaz son *public*, *static* y *final*.

Por defecto, los métodos de una interfaz son *public*.

Declaración de una clase que implementa una interfaz

```
class NombreClase implements NombreInterfaz {
    /* Código que implementa la interfaz.*/
    tipo_devuelto metodo1 ( parámetros ){
        /* Cuerpo del método de la interfaz. */
    }
    tipo_devuelto metodoM( parámetros ) {
        /* Cuerpo del método de la interfaz. */
    }
}
```

Paquetes

Son contenedores de datos y código (clases, interfaces y otros paquetes). Permiten la organización del código. Cada paquete tiene un nombre y permite restringir la visibilidad del código.

Cada fichero contiene en su primera línea el nombre del paquete al que pertenece. Los paquetes se nombran separados por un punto empezando desde el paquete más general. Un paquete está implementado en un directorio con el mismo nombre. La jerarquía de los paquetes y de los directorios debe coincidir.

Si un fichero no declara el nombre del paquete al que pertenece, se asocia al paquete por defecto y sin nombre, que corresponde al directorio actual (".").

Una clase pública debe implementarse en un fichero con el mismo nombre. Una clase no pública puede implementarse en el fichero de otra pública o en un fichero con cualquier nombre.

Asociación del fichero a un paquete (1ª línea del fichero)

```
package paq1[.paq2[...]];
```

Para usar una clase que pertenece a un determinado paquete hay que importar dicho paquete.

Importación de una clase que pertenece a un paquete

```
import paquete.Clase;
```

Encapsulación

Permite que un objeto oculte a otros datos y código, impidiendo su uso. Al declarar variables miembro y métodos en una clase es necesario indicar a qué objetos les está permitido utilizarlos. Eso se hará mediante palabras reservadas que preceden la declaración e indican el nivel de acceso.

Niveles de acceso: Determinan el acceso permitido a cada uno de los miembros de una clase.

Niveles de acceso

Acceso	Palabra reservada	Descripción
Privado	**Private**	Accesible sólo por el propio objeto.
Paquete	Ninguna	Es el nivel por defecto. Accesible en el ámbito del paquete.
Protegido	**Protected**	Accesible en el ámbito del paquete y en las subclases (aunque estén fuera del paquete).
Público	**Public**	Accesible por cualquier objeto.

Acceso a miembros según el nivel

Acceso	clase	package	subclase	todo
Público	X	X	X	X
Protegido	X	X	X	
Paquete	X	X		
Privado	X			

Final

Es una palabra reservada en Java. Su significado cambia según el contexto en el que se utilice (clase, método o atributo).

Significado de `final`

Contexto	Descripción
Clase	No se permite la herencia.
Método	No se permite la sobrescritura.
Atributo	No se permite la modificación.

Clase Object

Es la superclase de todas las clases en Java. Algunos métodos de la clase **`Object`:**

```
boolean equals(Object obj)
String  toString()
```

Miembros estáticos (`static`)

Son aquellos que no requieren instanciación de la clase para ser utilizados, y se puede acceder a ellos mediante el nombre de la clase (en vez de una referencia a un objeto). Si una variable de instancia se declara como estática, todos los objetos de la clase la comparten.

Matrices

Una matriz es un objeto (debe ser instanciado) que agrupa elementos del mismo tipo (el primer elemento es el de índice **0**).

	Unidimensional	Bidimensional
Declaración	`tipo matriz[ ];` `tipo [ ] matriz;`	`tipo matriz [ ] [ ];` `tipo [ ] [ ] matriz;`
Instanciación	`matriz =` `  new tipo[TAM];`	`matriz =` `  new tipo[TAM1][TAM2];`
Asignación	`matriz [indice] =` `  valor;`	`matriz [ind1][ind2] =` `  valor;`

Estas operaciones se pueden realizar en la declaración:

Unidimensional	Bidimensional
`tipo matriz [ ] =` `  new tipo [TAM];`	`tipo matriz [ ] [ ] =` `  new tipo [TAM1][TAM2];`

`tipo nombre [ ] =`
`  { valor0,`
`  ...,`
`  valortam-1};`
	`tipo nombre [ ] =`
	`  {{ valor00, valor01, …, valor0n },`
	`  { valor10, valor11, …, valor1n },`
	`  ...`
	`  { valorm0, valorm1, …, valormn }};`

El atributo `length` de una matriz indica el número de elementos.

Paso de argumentos

En Java la forma de pasar los argumentos en la llamada a un método depende del tipo del argumento:

1. **Tipo simple (por valor)**: Se copia el valor del argumento en el parámetro formal del método. Los cambios que se realizan sobre el parámetro formal del método no tienen efecto sobre el argumento utilizado en la llamada.
2. **Objeto (por referencia)**: El parámetro formal recibe la referencia del argumento utilizado en la llamada. Dentro del método, esta referencia se utiliza para acceder al argumento real especificado en la llamada. Esto significa que los cambios realizados al parámetro afectarán al argumento utilizado en la llamada.

Devolución de objetos

Un método puede devolver cualquier tipo de dato, incluyendo los tipos de clases definidos por el programador.

Argumentos en la línea de órdenes

Un programa comienza a ejecutarse con la invocación al método **`main()`** de la clase que se proporciona al intérprete de Java. Los argumentos en línea de comandos se proporcionan en una tabla de objetos de tipo **`String`**, que es el parámetro del método **`main(String args[])`**. En esta tabla, **`args[0]`** es el primer argumento, el que sigue al nombre de la clase (y no el nombre del programa como en C).

Ejemplo: que imprime los argumentos en la línea de órdenes.

```
public class ClasePrincipal {
  public static void main( String args[ ] ) {
    for (int i = 0; i<args.length; i++)
      System.out.println("args["+i +"]: "+args[i]);
  }
}
```

java.lang

Paquete que contiene clases generales (**`String`**, envolventes, …).

`String`: clase para representar y manejar cadenas de caracteres. Los objetos de tipo **`String`** no pueden modificarse, por lo que al cambiar una cadena de caracteres se construirá un nuevo objeto. Acepta el operador + para concatenar cadenas de caracteres entre sí o cadenas de caracteres con otros tipos de datos. Ejemplo:

```
String str = new String("abc");
```

Algunos métodos de la clase **`String`**:

`String` **`concat`**`(String str)`
Concatena el argumento al final de la cadena (devuelve un nuevo String)

`int` **`length`**`()`
Devuelve el número de caracteres de la cadena

`int` **`compareTo`**`( String otroString)`
Devuelve 0 si la cadena es igual a la cadena argumento

`int` **`indexOf`**`(int ch)`
Devuelve la posición de la primera ocurrencia del carácter

`String substring ( int beginIndex)`
Devuelve la subcadena desde el índice indicado en el argumento

`String` **`toLowerCase/toUpperCase`**`()`
Métodos para convertir todos los caracteres a minúsculas o mayúsculas

Clases envolventes (wrappers): Representación de tipos simples como objetos. Encapsulan el dato de un tipo simple y ofrecen una serie de métodos para manejarlo. Además proporcionan miembros (variables y métodos) de clase para realizar distintas operaciones sobre datos de tipos simples.

Los tipos de datos simples (boolean, char, int, long, double y float) no son objetos. Se pueden construir objetos a partir de los tipos de datos simples mediante el uso de clases envolventes (`Boolean`, `Character`, `Integer`, `Long`, `Double` y `Float`). En estas clases existen métodos para obtener el valor del tipo de dato simple asociado al objeto y otros métodos. Por ejemplo, la clase `Integer` es la envolvente para el tipo de dato simple `int`. Se puede construir un objeto de tipo `Integer` (con valor 4) de la siguiente forma:

```
Integer obj1 = new Integer(4);
Integer obj2 = new Integer("4");
Integer obj3 = Integer.valueOf(4);
```

Para obtener el valor `int` de un objeto de tipo `Integer`:

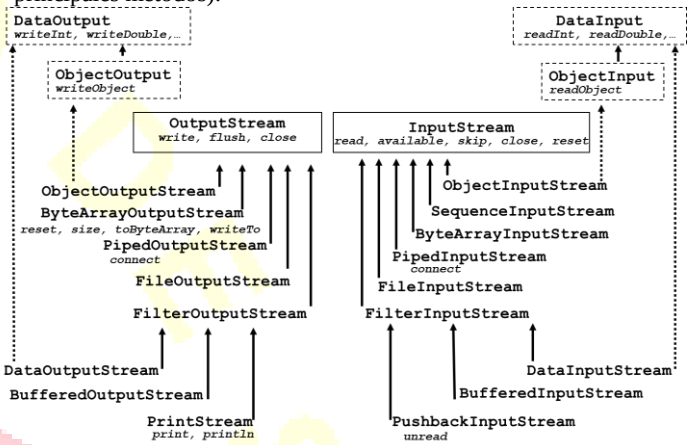
```
int valor = obj1.intValue();
```

Existen métodos y constructores similares en las demás clases envolventes.

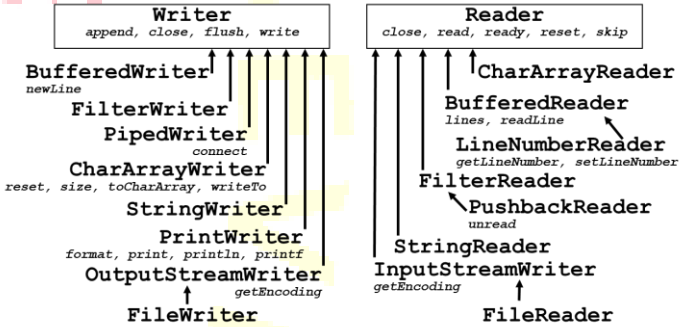
java.io

Los programas Java realizan la entrada y salida de datos a través de flujos (streams). Un flujo es una abstracción que produce o consume información. Existen dos tipos de flujos binarios y de texto.

La jerarquía de clases de los flujos binarios (en cursiva sus principales métodos):



Jerarquía de clases de los flujos de texto (en cursiva sus principales métodos):



Flujos predefinidos: **`System.in`** (**`InputStream`**), **`System.out`** (**`PrintStream`**) y **`System.err`** (**`PrintStream`**).

Excepciones

Situaciones anormales durante la ejecución de un programa. En Java son objetos del tipo `Exception` que se crean cuando se produce una situación excepcional (error producido en un fragmento de código) que la describe y se puede lanzar para avisar de esta situación. Existen excepciones ya creadas o bien se pueden declarar las propias excepciones derivando de `Exception`:

```
class ClaseExcepcion extends Exception {
   /* Código de la clase ClaseExcepcion. */
}
```

Un método que reciba la excepción puede elegir *capturarla* y gestionarla él mismo. La gestión de excepciones se realiza mediante bloques `try/catch/finally`. Las sentencias `try` pueden estar anidadas.

Bloque de gestión de excepciones

```
try {
    /* Bloque de código. */
} catch (TipoExcep1 refObjetoException) {
    /* Gestor de excepciones para TipoExcep1. */
} catch (TipoExcep2 refObjetoException) {
    /* Gestor de excepciones para TipoExcep2. */
} finally {
    /* Bloque de código que se ejecutará antes de
       que termine el bloque try. */
}
```

Cualquier excepción que no es capturada por el programa será tratada por el gestor por defecto, que muestra un mensaje describiendo la excepción, imprime el trazado de la pila del lugar donde se produjo la excepción y termina el programa.

Lanzamiento de una excepción:

```
throw objetoDeLaClaseExcepcion;
```

Para obtener un objeto que se lance con `throw`:

1. se puede usar la referencia de la cláusula `catch`, o
2. crear el objeto con el operador `new`:

```
new ClaseExcepcion();
```

El flujo de la ejecución se detiene inmediatamente después de la sentencia `throw` y cualquier sentencia posterior no se ejecuta.

Declaración de excepciones en métodos: Si un método es capaz de provocar una excepción que no maneja él mismo, debería especificar este comportamiento para que los métodos que lo llaman puedan protegerse frente a esta excepción. Para ello se incluye una cláusula `throws` en la declaración del método, que lista los tipos de excepciones que un método puede lanzar, si no están declaradas se producirá un error de compilación.

Antonio J. Sierra Collado
María Teresa Ariza Gómez
Javier Muñoz Calle
Francisco José Fernández Jiménez
Juan Antonio Ternero Muñiz
Isabel Román Martínez
Germán Madinabeitia Luque
José Manuel Fornés Rumbao
Nicolás Guil Mata

Herramientas Java

Compilar	`javac [-classpath dirs] [-d dst] [opciones] [./subdir/]clase(s).java`
Depurar	`jdb [-classpath dirs] [-sourcepath src] [opciones] clase_inicial [argumentos]`
Ejecutar	`Java [-classpath dirs] [opciones] [paquete.]clase_inicial [argumentos]` `java [-classpath dirs] [opciones] -jarfichero.jar [argumentos]`
Empaquetar	`jar cvfe fichero.jar [paquete.clase_inicial] [-C src] ./subdir/clase(s).class` `jar cvfm fichero.jar Manifest [-C src] ./subdir/clase(s).class`
Desemp.	`jar xvf fichero.jar [-C dst]`
Generar doc.	`javadoc [-d dst] [-sourcepath src] [tipo] [opciones] [paquetes] [ficheros_java]`

- `argumentos`: parámetros pasados al método "main()" de la clase_inicial por la que comienza la ejecución.
- `dirs`: conjunto de directorios (separados por ":") donde localizar los ficheros ".class" o ".jar" necesarios para la compilación/ejecución. Esta opción sobrescribe la variable de entorno CLASSPATH. Si no se define ninguno de ambos, por omisión "`dirs`" es el directorio de trabajo "./".
- `dst`: carpeta donde ubicar los ficheros generados (".class", ".html").
- `src`: ubicación de los ficheros fuente (a depurar, empaquetar o de los que generar documentación).
- `subdir`: conjunto de subdirectorios para llegar al fichero "clase.java" o "clase.class" indicado (dicha clase deberá contener la línea "package" con dicho conjunto de subdirectorios). "jar" admite múltiples ficheros de clase, directorios completos y comodines " (si se indica "subdir", "paquete" deben indicar dichos directorios).
- `tipo`: permite seleccionar de qué clases y métodos generar la documentación en función de su nivel de acceso ("`public`", "`protected`" por omisión, "`private`" o "`package`" para public y protected).

Funcionamiento:

- `javac`: lee ficheros "`clase.java`" y genera ficheros "`clase.class`".
- `jdb`: dentro del depurador se pueden emplear múltiples comandos, tales como: "`help`", "`quit`", "`run [clase [argumentos]]`" (método "`main`" de la "clase_inicial" por defecto), "`stop at [paquete.]clase:linea`", "`stop in [paquete.]clase.metodo`" (método "<init>" para el constructor), "`stop`", "`clear`", "`cont`", "`step`", "`next`", "`catch [paquete.]excepción`", "`ignore`", "`print [expresión_Java]`", "`set elemento=expresión_Java`", "`list`", "`where`", "`locals`", "`classes`", "`methods clase`", "`fields clase`", ...
- `java`: comienza la ejecución en el fichero "clase_inicial.class"(método "`public static void main(String[] args)`"), expresamente indicado o recogido en el Manifest del "`fichero.jar`".
- `jar`: para ficheros ".jar" que requieren un método "main()" de inicio (no existente en caso de librerías "`.jar`") puede indicarse una "clase_inicial" por la que comenzar la ejecución con el parámetro "e", o bien usando el parámetro "m" y un fichero de texto "Manifest" que contenga la línea "`Main-Class: paquete.clase_inicial`" (terminada con el carácter de nueva línea).
- `javadoc`: genera documentación HTML (un fichero "`.html`" por cada clase, y un fichero "index.html" de índice) a partir de los comentarios, existentes en el código fuente de los paquetes y ficheros Java indicados, ubicados inmediatamente antes de una clase, interfaz, método o campo, y que presenten el patrón "`/** Comentario */`" (en los comentarios se pueden usar etiquetas HTML y etiquetas javadoc tales como "`@author`", "`@version`", "`@since`", ...). La primera frase de cada comentario (texto hasta el primer punto seguido de espacio) se toma como resumen.