# UT1 - RAT

## JAVA

Java surgió en 1991 por un grupo de ingenieros de Sun Microsystems que trataron de diseñar un nuevo lenguaje de programación para electrodomésticos, por eso llegaron a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Desarrollan un código neutro que no depende del tipo de electrodoméstico, el cual se ejecuta sobre una máquina virtual, y la llaman **Java Virtual Machine.** La JVM interpreta el código neutro y lo convierte a código particular de la CPU utilizada(”Write once, run everywhere”).

Ninguna empresa de electrodomésticos se interesó por el lenguaje, Java, como lenguaje de programación para computadoras se introdujo a finales de 1995, cuando se incorpora un intérprete Java en el programa Netscape Navigator, versión 2.0, produciendo una revolución en Internet.

Al programas Java no se parte de cero, se apoya en un gran número de clases preexistentes. Incorpora incorpora muchos aspectos que en otros lenguajes con extensiones (threads, ejecución remota, seguridad, etc.), por eso es un lenguaje ideal para aprender la informática moderna. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

El principal objetivo del lenguaje es llegar a ser el nexo universal que conecte a los usuarios con la información.

La compañía Sun describe Java como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”.

Los programas desarrollados en Java presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en Java tiene muchas posibilidades: ejecución como aplicación independiente (Stand-alone Application), ejecución como applet, ejecución como servlet, etc.. Un applet es una aplicación especial que se ejecuta dentro de un navegador o browser (por ejemplo Netscape Navigator o Internet Explorer) al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un servlet es una aplicación sin interface gráfica que se

ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

Además de incorporar la ejecución como Applet, Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, Java incorpora en su propio API estas funcionalidades.

Los nombres en Java son sensibles a mayúsculas y minúsculas. Las reglas del lenguaje respecto a los nombres de variables son amplias, pero es habitual seguir ciertas normas que facilitan la lectura y el mantenimiento de los programas.

1. Es habitual utilizar nombres con minúsculas, con las excepciones que se indican a continuación.
2. Se utiliza camelcase.
3. Los nombres de clases e interfaces empiezan siempre con mayúscula.
4. Los objetos, métodos y variables miembro, y variables locales de los métodos empiezan siempre con minúscula.
5. Las variables finales (constantes) se definen siempre con mayúsculas (Ejemplo: PI).

Aparece una clase que contiene el programa principal (aquel que contiene la función main()) y algunas clases de usuario (las específicas de la aplicación que se está desarrollando) que son utilizadas por el programa principal. Los ficheros fuente tienen la extensión *.java, mientras que los
ficheros compilados tienen la extensión *.class.*
*Un fichero fuente (*.java) puede contener más de una clase, pero sólo una puede ser public. El nombre debe coincidir con el de la clase public (con la extensión *.java). Si por ejemplo en un fichero aparece la declaración (public class MiClase {...}) entonces el nombre del fichero deberá ser MiClase.java. Si la clase no es public, no es necesario que su nombre coincida con el del fichero. Una clase puede ser public o package (default), pero no private o protected.

Una aplicación está constituida por varios ficheros *.class. Cada clase realiza unas funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases. La aplicación se ejecuta por medio del nombre de la clase que contiene la función main() (sin la extensión *.class). Las clases de Java se agrupan en packages, que son librerías de clases. Si las clases no se definen como pertenecientes a un package, se utiliza un package por defecto (default) que es el directorio activo.

Una **clase** es una agrupación de datos (variables o campos) y de funciones (métodos) que operan sobre esos datos. Los elementos declarados en una clase se denominan objetos de la clase. Las clases pueden tener variables static que son propias de la clase y no de cada objeto.

La **herencia** permite que se puedan crear nuevas clases basadas en clases existentes (re-utilizar código). Si una clase deriva de otra (extends) hereda todas sus variables y métodos. La clase derivada puede añadir variables y métodos o redefinir los heredados. No se puede realizar herencia múltiple en base a clases, pero se puede simularla en base a las interfaces.

Una **interface** es un conjunto de declaraciones de funciones. Si una clase implementa (implements) una interface, debe definir TODAS las funciones especificadas en la misma. Una clase puede implementar más de una interface. Una interface puede derivar de otra o incluso de varias interfaces.

Un **package** es una agrupación de clases. Existen una seria de packages incluidos en el lenguaje, además el usuario puede crear sus propios.

## Variables

Una variable es un nombre que contiene un valor que puede cambiar a lo largo del programa. Según el tipo de información de contienen, hay dos tipos principales:

- Variables de tipos primitivos: se define mediante un valor único.
- Variables referencia: son referencias o nombres de una información más compleja: arrays u objetos de determinada clase.

Según su papel en el programa las variables pueden ser:

- Variables miembro de una clase: se definen en una clase, fuera de cualquier método.

- Variables locales: Se definen dentro de un método, dentro de cualquier bloque de llaves {}.

Existe una serie de ***palabras reservadas*** las cuales tienen un significado especial para ***Java*** y por lo tanto no se pueden utilizar como nombres de variables. Dichas palabras son:

| | | | | | |
|---|---|---|---|---|---|
| abstract | boolean | break | byte | case | catch |
| char | class | const* | continue | default | do |
| double | else | extends | final | finally | float |
| for | goto* | if | implements | import | instanceof |
| int | interface | long | native | new | null |
| package | private | protected | public | return | short |
| static | super | switch | synchronized | this | throw |
| throws | transient | try | void | volatile | while |

### Tipos primitivos de Variables

Variables sencillas que contienen los tipos de información más habituales: boolean, char, int, float.

Java dispone de 8 tipos primitivos:

| Tipo de variable | Descripción |
|---|---|
| Boolean | 1 byte. Valores true y false |
| Char | 2 bytes. Unicode. Comprende el código ASCII |
| Byte | 1 byte. Valor entero entre -128 y 127 |
| Short | 2 bytes. Valor entero entre -32768 y 32767 |
| Int | 4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647 |
| Long | 8 bytes. Valor entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807 |
| Float | 4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38 |
| Double | 8 bytes (unas 15 cifras decimales equivalentes). De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308 |

Se utiliza void para indicar la ausencia de un tipo de variable determinado.

Una variable se define especificando el tipo y el nombre de la misma. Las variables primitivas de inicializan en 0 (boolean en false y char en ‘/0’), y las de referencia con el valor null.

Una **referencia** es una variable que indica dónde está en la memoria del ordenador un objeto, al declararla todavía no se encuentra “apuntando” a ningún objeto en particular. Si se desea que apunte a un nuevo objeto se utiliza el operador **new**. Este reserva en

la memoria del ordenador espacio para ese objeto (variables y funciones). También es posible igualar la referencia declarada a un objeto existente previamente.

**Visibilidad y vida de las variables**

La visibilidad, vida o scope de una variable es la parte de la aplicación donde dicha variable es accesible y puede ser utilizada. En general las variables declaradas dentro de {} (dentro de un bloque) son visibles y existen dentro de esas llaves.

- Las variables miembro de una clase declaradas como public son accesibles a través de una referencia a un objeto de dicha clase utilizando el operador punto (.).
- Las variables miembro declaradas como private no son accesibles directamente desde otras clases.
- Las funciones miembro de una clase tienen acceso directo a todas las variables miembro de la clase sin necesidad de anteponer el nombre de un objeto.
- Una clase derivada solo puede acceder directamente a las variables y funciones miembro de su clase base declaradas como public o protected.
- Es posible declarar una variable dentro de un bloque con el mismo nombre que una variable miembro (no de otra local). La variable local oculta la variable miembro en ese bloque, para acceder a la variable miembro oculta se tiene que utilizar el this.

La eliminación de los objetos la realiza el garbage collector, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto. Lo anterior significa que aunque una variable de tipo referencia deje de existir, el objeto al cual apunta no es eliminado si hay otras referencias apuntando a ese mismo objeto.

**Clases BigInteger y BigDecimal**

Destinadas a operaciones aritméticas que requieran gran precisión. La forma de operar con objetos de estas clases difiere de las operaciones con variables primitivas. En este caso hay que realizar las operaciones utilizando métodos propios de estas clases (add() para la suma, subtract() para la resta, divide() para la división, etc.).
Los objetos de tipo BigInteger son capaces de almacenar cualquier número entero sin perder información. Esto significa que es posible trabajar con enteros de cualquier número de cifras sin perder información durante las operaciones. Análogamente los objetos de tipo BigDecimal permiten trabajar con el número de decimales deseado.

## Operadores

Java es un lenguaje rico en operadores, que son casi idénticos a los de C/C++.

Operadores aritméticos: Son binarios (requieren dos operandos) que realizan las operaciones aritméticas.

Operadores de asignación: Permiten asignar un valor a una variable. ( =, +=, -=, etc.)

Operadores unarios: Los operadores + y -.

Operadores instanceof: Permite saber si un objeto pertenece a una determinada clase o no (boolean). Forma general: *objectName instanceof ClassName*

Operador condicional ?: Permite realizar bifurcaciones condicionales sencillas. *booleanExpression ? res1 : res2* Esto evalúa booleanExpression y devuelve res1 si es true y res2 si es false.

Operadores incrementales: (++) incrementa en una unidad la variable a la que se aplica, y (- -) la reduce en un unidad. (++i) primero se incrementa la variable y luego se utiliza, (i++) primero se utiliza y luego se incrementa la variable.

Operadores relacionales: Sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado es un valor booleano.

Operadores lógicos:

| Operador | Nombre | Utilización | Resultado |
|---|---|---|---|
| `&&` | AND | `op1 && op2` | true si op1 y op2 son true. Si op1 es false ya no se evalúa op2 |
| `\|\|` | OR | `op1 \|\| op2` | true si op1 u op2 son true. Si op1 es true ya no se evalúa op2 |
| `!` | negación | `! op` | true si op es false y false si op es true |
| `&` | AND | `op1 & op2` | true si op1 y op2 son true. Siempre se evalúa op2 |
| `\|` | OR | `op1 \| op2` | true si op1 u op2 son true. Siempre se evalúa op2 |

Operador de concatenación de cadenas de caracteres (+): Ejemplo:
*System.out.println("El total asciende a " + result + " unidades");*

Operadores que actúan a nivel de bits: Las operaciones de bits se utilizan con frecuencia para definir señales o flags (variables de tipo entero en la que cada uno de sus bits indican si una opción está activada o no).

| Operador | Utilización | Resultado |
|---|---|---|
| >> | op1 >> op2 | Desplaza los bits de op1 a la derecha una distancia op2 |
| << | op1 << op2 | Desplaza los bits de op1 a la izquierda una distancia op2 |
| >>> | op1 >>> op2 | Desplaza los bits de op1 a la derecha una distancia op2 (positiva) |
| & | op1 & op2 | Operador AND a nivel de bits |
| \| | op1 \| op2 | Operador OR a nivel de bits |
| ^ | op1 ^ op2 | Operador XOR a nivel de bits |
| ~ | ~op2 | Operador complemento |

Orden en que se ejecutan los distintos operadores en una sentencia, de mayor a menor precedencia:

| | |
|---|---|
| postfix operators | `[] . (params) expr++ expr--` |
| unary operators | `++expr --expr +expr -expr ~ !` |
| creation or cast | `new (type)expr` |
| multiplicative | `* / %` |
| additive | `+ -` |
| shift | `<< >> >>>` |
| relational | `< > <= >= instanceof` |
| equality | `== !=` |
| bitwise AND | `&` |
| bitwise exclusive OR | `^` |
| bitwise inclusive OR | `\|` |
| logical AND | `&&` |
| logical OR | `\|\|` |
| conditional | `? :` |
| assignment | `= += -= *= /= %= &= ^= \|= <<= >>= >>>=` |

En Java, todos los operadores binarios, menos los de asignación, se evalúan de izquierda a derecha. Los de asignación se evalúan de derecha a izquierda, lo que significa que el valor de la derecha se copia sobre la variable de la derecha.

## Estructuras de programación

Las estructuras de programación o estructuras de control permiten tomar decisiones y realizar un proceso repetidas veces. La sintaxis de Java coincide prácticamente con la utilizada en C/C++.

Una **expresión** es un conjunto variables unidos por operadores. Son órdenes que se le dan a la computadora para que realice una tarea determinada.

Una **sentencia** es una expresión que acaba en punto y coma (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia. Por ejemplo:
*i = 0; j = 5; x = i + j;*// Línea compuesta de tres sentencias.

**Comentarios**

Se puede comentar una línea con // todo lo que esté a la derecha de las barras Java lo toma como un comentario. También se puede comentar un párrafo con /* … */

Existe además una forma especial de introducir los comentarios (utilizando /**…*/ más algunos caracteres especiales) que permite generar automáticamente la documentación sobre las clases y packages desarrollados por el programador.

**Bifurcación if:** Se escribe de la siguiente forma

```
if (booleanExpression) {
    statements;
}
```

Las {} sirven para agrupan en un bloque las sentencias que se van a ejecutar, y no son necesarias si sólo hay una sentencia dentro del if.

**Bifurcación if else:**

```
if (booleanExpression) {
    statements1;
}
else {
    statements2;
}
```

**Bifurcación if elseif else:**

```
if (booleanExpression1) {
    statements1;
} else if (booleanExpression2) {
    statements2;
} else if (booleanExpression3) {
    statements3;
}
else {
    statements4;
}
```

**Sentencia switch:** Se trata de una alternativa a la bifurcación if elseif else cuando se compara la misma expresión con distintos valores. Su forma general es la siguiente

```
switch (expression) {

        case value1: statements1; break;
        case value2: statements2; break;
        case value3: statements3; break;
        case value4: statements4; break;
        case value5: statements5; break;
        case value6: statements6; break;
        [default: statements7;]

}
```

Cada sentencia case se corresponde con un único valor de expression. No se pueden establecer rangos o condiciones sino que se debe comparar con valores concretos.

Los valores no comprendidos en ninguna sentencia case se pueden gestionar en default, que es
opcional.

En ausencia de break, cuando se ejecuta una sentencia case se ejecutan también todas las que van a continuación, hasta que se llega a un break o hasta que se termina el switch.

**Bucles**

Un bucle se utiliza para realizar un proceso repetidas veces. Se denomina también lazo o loop. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta

de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones.

Bucle while: Las sentencias se ejecutan mientras booleanExpression sea true

```
while (booleanExpression) {
    statements;
}
```

Bucle for: La sentencia initialization se ejecuta al comienzo del for, e increment después de statements. La booleanExpression se evalúa al comienzo de cada iteración, el bucle termina cuando la expresión de comparación toma el valor false.

```
for (initialization; booleanExpression; increment) {
    statements;
}
```

Bucle do while: Es como el bucle while pero el control está al final del bucle (lo que hace que el bucle se ejecute al menos una vez, independientemente de que la condición se cumple o no. Una vez ejecutados los statements, se evalúa la condición, si es true se vuelven a ejecutar las sentencias y si es false finaliza el bucle.

```
do {
    statements
} while (booleanExpression);
```