

Llenguatges de programació

El llenguatge de programació que s'usa a l'assignatura és Java. Aquest llenguatge sorgeix el 1991 d'un projecte de Sun Microsystems de dissenyar un llenguatge per a la programació d'electrodomèstics. No és fins a finals del 1995 que apareix com llenguatge de programació per a ordinadors. Com a curiositat es pot citar que per primera vegada es va debatre a internet l'especificació d'un llenguatge de programació. Té l'avantatge d'estar fortament implantat en el mercat i que la majoria de vosaltres en sabeu poc o molt (més poc o que molt). A més, disposa d'eines i biblioteques de classes prou completes i depurades.

Anteriorment a l'assignatura s'havia usat Eiffel. Aquest fou concebut al 1985 pel Dr. Bertran Meyer, un pioner notable de la programació OO. Hi ha un reconeixement general de què és el llenguatge més ben dissenyat i més complet. Encara no s'ha introduït amb força en el mercat, tot i que en aquest moment hi ha algunes esperances que això canviï ja que forma part del conjunt de llenguatges que ha incorporat Microsoft a la seva plataforma ".net". Actualment les eines i biblioteques disponibles per a Eiffel no són tan completes i depurades com el que es pot trobar per altres llenguatges, i a vegades resulta massa complicat de treballar-hi.

No disposem de temps per fer un curs complet de Java, però, a més de la bibliografia recomanada a l'assignatura, a internet s'hi poden trobar bones referències i fins i tot material d'aprenentatge. Convé vigilar ja que hi ha subtils diferències entre les diferents versions del llenguatge.

Java s'executa amb una màquina específica (un intèrpret) anomenada *Java Virtual Machine* (JVM) a partir d'un codi neutre (o intermedi). JVM tradueix aquest codi intermedi a l'específic de la màquina on s'executa. El compilador converteix els fitxers font d'extensió ".java" al codi intermedi creant un fitxer per classe d'extensió ".class", és a dir, que un executable és un conjunt de fitxers d'extensió ".class" (*bytecodes*), normalment un per classe (dic normalment perquè quan hi ha classes internes no se satisfà aquesta condició). Cal tenir cura a controlar bé les versions dels fitxers ".class", és relativament fàcil de fer barreges poc adequades. Al compilador (que al DOS és *javac.exe*) només cal dir-li la classe que conté el programa i s'ocupa de compilar les classes que li calen. El compilador fa compilació incremental amb les classes que són d'accés "public" (simplement compara la data i hora del fitxer ".class" amb el corresponent ".java"). Es disposa d'eines que permeten emmagatzemar tots els fitxers d'extensió ".class" en un únic fitxer que sap executar directament la JVM cosa que en facilita la instal·lació i manipulació en general.

Comentarem els trets més importants del llenguatge. Per començar no hi ha una diferència clara entre els atributs i els mètodes que componen una classe. La sintaxi de Java està inspirada en la de C. Així:

```
if( condició ) {
    sentències;
}
else {
    sentències;
}

switch( expressió_1 ) {
    case expressió_2:
        sentències;
        break;
    ...
    default:
        sentències;
        break;
}
```

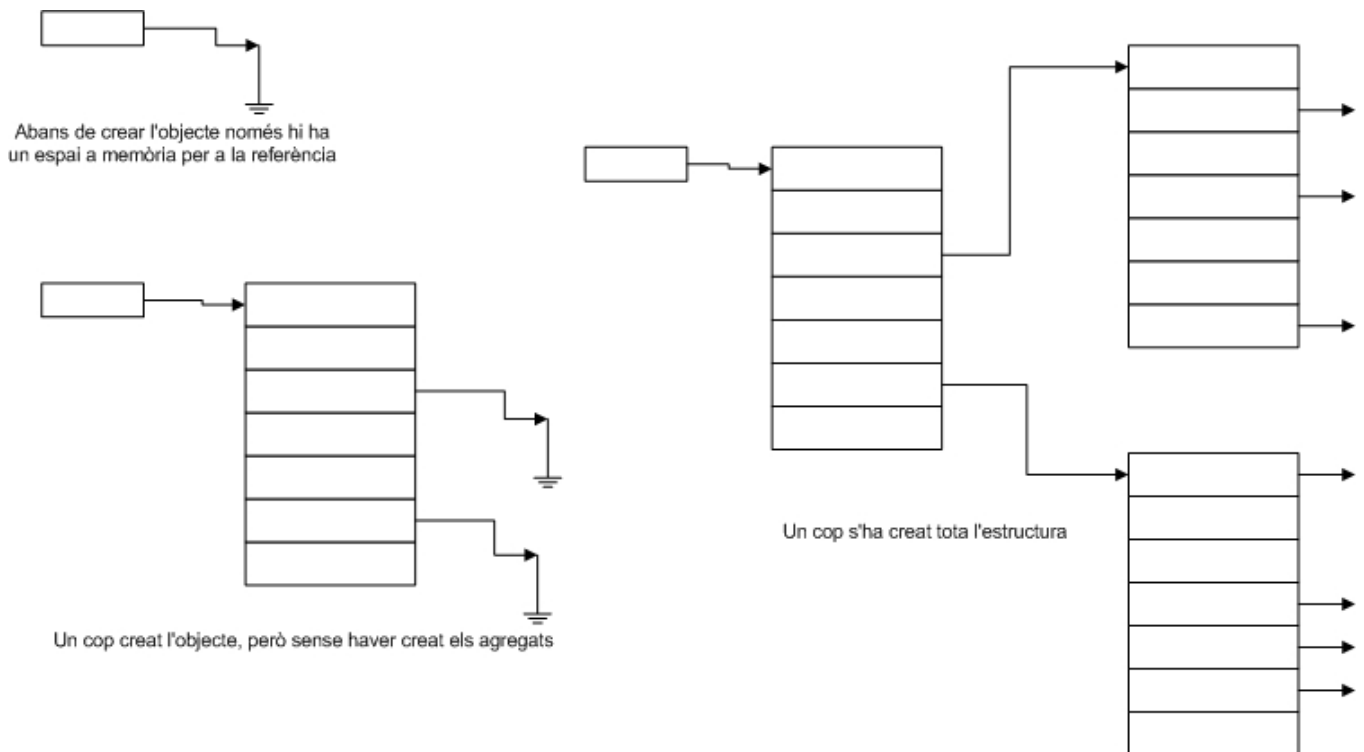
```
while( condició ) {
    sentències;
}
```

```
do {
    sentències;
} while( condició );
```

```
for( inici ; condició ; increment ) {
    sentències;
}
```

Els atributs poden ser simples o complexos. El tipus simples són: boolean, char, byte, short, int, long, float i double (en resum, un lògic, un per a caràcters, 4 enters i 2 reals).

Els atributs complexos poden sortir de les biblioteques de classes o formar part del desenvolupament, però les classes estaran definides en general en fitxers separats. Els atributs complexos s'implementen per defecte en l'estructura de la classe amb una referència (un punter) al lloc de la memòria on s'instancia la verdadera estructura complexa.



Convé tenir ben present aquest tret ja que dona l'explicació del comportament real de l'assignació, de la còpia de classes, i del pas de classes com a paràmetres. A més, justifica la complexitat de les operacions de guardar i recuperar classes en fitxers i bases de dades. En general els llenguatges orientats a objectes implementen així els objectes en memòria.

Els valors dels atributs són inicialitzats automàticament: el valor lògic amb *fals*, el caràcter amb el *caràcter nul*, tots els numèrics amb *zero* i les referències es creen *buides* (això obliga al programador a instanciar sempre l'objecte referenciat).

A Java (igual que a C) existeix el mot *void* que indica l'absència de tipus (imprescindible per escriure procediments).

Totes les classes n'hereten alguna altra. En alguns casos aquesta herència és explícita, escrita en el programa. En la resta de casos es produeix per defecte quan no hi ha la clàusula que explicita d'herència. En Java la classe que s'hereta per defecte es diu OBJECT. Convé conèixer quines són les conseqüències d'aquesta herència (atributs i mètodes) ja que tota classe la rep directa o indirectament.

Els mètodes creadors tenen tots el nom de la classe, es diferencien pels paràmetres (quantitat i tipus), per instanciar un objecte s'usa la clàusula *new*, el nom de la classe i el paràmetres.

```
String literal  
literal = new String("això és text")
```

O simplement:

```
String literal = "això és text"
```

O bé:

```
String literal = new String("això és text")
```

Els **mètodes** de la classe són procediments i funcions. En Java, com en C, un procediment és declara com una funció que retorna *void*, és a dir, es declaren igual i difereixen en el tipus del retorn.

```
return <<nom>> (paràmetres)
```

El tipus retornat (*return*) és únic però aquest pot ser una estructura tan complexa com calgui. El resultat es retorna amb la clàusula *return* i a continuació un resultat del tipus de *return*. Tota funció ha d'executar una clàusula *return* abans d'acabar.

Sempre passen els paràmetres per valor i no permeten especificar altra cosa, però quan es passa una classe complexa el valor és el d'una referència, és a dir, que s'està fent un pas per referència.

Per declarar **constants** es disposa de les clàusules *final* i *static* que juntes indiquen que un atribut és constant:

```
public static final double PI=3.14159265358979323846;
```

La clàusula *final* indica que no es pot modificar i *static*, l'existència d'una única còpia per a tots els objectes de la classe que existeix, fins i tot, abans d'instanciar el primer objecte d'aquesta classe. Però convé tenir present que la clàusula *final* té un sentit diferent quan s'aplica a un mètode i encara un altre quan s'aplica a una classe.

Visibilitat: qualsevol atribut o mètode d'una classe és visible en tota la classe, però és possible limitar aquesta visibilitat des de les altres classes del programa. Per defecte aquests són visibles per les classes que pertanyen al mateix paquet que la classe. (Per definir els paquets Java disposa de la clàusula *package* explicada a continuació.)

Existeixen les clàusules *private*, *protected* i *public* per modificar la visibilitat:

- *private*, només la classe, ni tan sols qui l'hereti.
- *protected*, el membres del mateix paquet (*package*) i qui l'hereti.
- *public*, accés sense restriccions.

Una classe o és `public` o només visible en el `package` (que és valor per defecte). En canvi per un atribut o mètode les coses van així:

Visibilitat (des de...)	<code>public</code>	<code>protected</code>	<code>private</code>	<code>default</code>
La pròpia classe	Sí	Sí	Sí	Sí
Una altra classe del propi paquet	Sí	Sí	No	Sí
Una altra classe fora del paquet	Sí	No	No	No
Una subclasse del propi paquet	Sí	Sí	No	Sí
Una subclasse fora del propi paquet	Sí	Sí	No	No

Java disposa de la clàusula `package` per a la creació de **paquets** (instància dels temes i subsistemes que s'hagi definit a la fase d'anàlisi). L'estructura de paquets de Java no té cap relació amb la d'herències. Perquè una classe pertanyi a un paquet (de nom `<<nomDePaquet>>` per exemple) el fitxer ".java" ha de començar amb la declaració:

```
package <<nomDePaquet>>;
```

L'estructura de paquets ha de correspondre amb la de directoris i subdirectoris on s'emmagatzemen el fitxers font (.java) i objecte (.class) de les classes.

Del mecanisme que permet la **compatibilitat entre tipus** el Java en diu **cast**. És automàtic entre classes numèriques simples i en l'herència de fills a pares cosa que permet crear un objecte d'una classe pare amb l'estructura d'un fill ja que un objecte d'una classe fill té completa l'estructura del pare, i en general tindrà alguns atributs i operacions més que els objectes de la classe pare (per exemple, `PERSONA unaPersona = new CIUTADA ()`, imaginant que la classe `CIUTADA` hereta directa o indirectament de la classe `PERSONA`). També admet el **cast** de pares a fills, però en aquest cas cal fer-lo explícitament, i definir la part que en falta.

La pròpia sintaxi del llenguatge impedeix l'**herència múltiple**, és a dir, que impedeix heretar més d'una classe. La clàusula per a l'herència és `extends`, té una sintaxi que només permet l'herència simple.

Es pot considerar que l'herència múltiple es cobreix de forma parcial o força limitada amb les **interfícies** de Java a través de la clàusula `implements`, però de fet una interfície no és una classe ja que només admet definir o enunciar mètodes diferits que haurà d'implementar qui l'hereti i no té atributs, és a dir, que ve a ser una classe sense atributs on totes les operacions estan diferides. La combinació de les interfícies i el `cast` havia permès a les versions anterior de Java implementar d'alguna manera la genericitat.

Així com la clàusula `extends` només admet un nom de classe la `implements` admet una llista d'interfícies. La sintaxi és:

```
class <<nom>> extends <<pare>> implements <<llistaInterfícies>> {
```

Es pot declarar una classe com a diferida (`abstract`), quan tingui algun element diferit (també declarat com a `abstract`), però no es podrà instanciar objectes reals d'aquest tipus. Es declaren

```
[public] abstract class <<nom>>...
```

Fins a la versió 1.5 Java no ha donat suport complet a la **genericitat**, és a dir, força recentment.

De fet, està inspirat en els *templates* del C++ que serveixen per implementar la genericitat en aquell llenguatge. Un exemple en Java:

```
List<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

On `List` és una *generic interface* que pren un paràmetre tipus, en aquest cas, `Integer`. Caldrà especificar el paràmetre tipus cada cop que es creï un objecte de la classe `List`. S'haurà definit:

```
public interface List<E> {
```

La classe `LinkedList` haurà implementat `List` o haurà heretat d'algú que ho hagi fet perquè hi hagi possibilitat de *cast* entre ella i `List`.

Una classe genèrica admetrà més d'un paràmetre (així: `class HashTable <Key, Value>`) i es pot imposar que la classe paràmetre hagi ser tal que hereti d'una classe concreta (amb `extends` o `implements`).

```
interface Hashable {...};
class HashTable <Key implements Hashable, Value> {...};
```

Admet un tractament dels errors a través de les classes derivades d'`Exception` (*java.lang.Exception*) emprant els blocs `try`, `catch` i `finally`. També hi ha un mecanisme que permet incorporar l'especificació formal (pre i postcondicions, invariants, etc.) en el font del programa i després usar-la per comprovar la correctesa del programa en la fase de prova. Es fa amb la sentència `assert`, aquesta permet la verificació de predicats (expressions booleanes), que poden o no generar un avís. Normalment només s'habiliten durant la fase de proves i no s'usen durant l'exploració normal.

L'habilitació es fa amb el modificador *ea* en el moment de compilar, és a dir, posant `javac -ea ...`)

Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (*threads*, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Por eso muchos expertos opinan que *Java* es el lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes.

Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazales, Alberto Larzabal, Jesús Calleja i Jon García. Escuela Superior de Ingenieros Industriales de San Sebastián. Universidad de Navarra.

El compilador Java v1.5

Cal disposar com a mínim del JDK (Java Development Kit), que us podeu baixar de <http://java.sun.com>.

Resulta còmode que la PATH de DOS inclogui el directori BIN del compilador (per exemple: "C:\Archivos de programa\fib\jdk1.5\bin").

La variable d'entorn CLASSPATH indica al compilador els directoris on buscar les classes. En les darreres versions no cal que CLASSPATH inclogui el directori actual ni els de les biblioteques (*libraries*) estàndard.

La sentència de compilació:

```
javac <<nomDeFitxer>>.java
```

Cal posar l'extensió ".java", en canvi a la sentència o ordre d'execució no s'hi posa l'extensió:

```
java <<nomDeFitxer>> [arguments]
```

Es poden muntar biblioteques de classes en fitxers ".jar" o ".zip" (aquests sense comprimir). També es poden emmagatzemar en un únic fitxer ".jar" tots els fitxers ".class" d'un executable i executar-lo directament amb la JVM. En el directori BIN hi ha una eina d'utilitat per muntar fitxers ".jar". Si convé que el compilador tracti el fitxer ".jar" com un directori o biblioteca de classes més s'haurà d'afegir el nom complet del fitxer a CLASSPATH

A l'entorn d'execució només cal disposar de la JVM, es pot baixar del mateix lloc descarregant *J2SE Runtime Environment (JRE)*.