Treball dirigit LP
(Q1 2024-25)
Hash 55590

Índex

1) Història

Cap al 1991, James Gosling, de l'empresa Sun Microsystems treballava en la creació d'una plataforma de software en la que usava C++ i al veure les problemàtiques que tenia amb l'herència múltiple, conversió automàtica de tipus, l'ús de punters i la gestió de memòria va decidir crear un llenguatge basat en C++ per solucionar aquests problemes que va batejar inicialment com a *Oak*.

Al 1994, va aparèixer la web del llenguatge, però aquest cop sota el nom de *Java* (nom inspirat en el cafè Indonesi), perque la marca *Oak* ja estava patentada. El 23 de gener de 1996 es va publicar el llenguatge que, malgrat no ser de codi obert *(open source),* era d'ús obert (*free to use*).

Java no només era un llenguatge de programació, també incloïa els següents elements:

- Màquina Virtual (JVM): Permetia executar aplicacions en diferents sistemes operatius sense modificar el codi font, pel que va impulsar Java al ser un llenguatge portable.

- Joc d'eines de desenvolupament (JDK): Incloïa el compilador, eines de depuració i altres utilitats per desenvolupar aplicacions.

- Interfície de Programació d'Aplicacions (API): Proveïa biblioteques estàndard per simplificar el desenvolupament, com ara:
    - "Java.util", "Java.io" per estructures de dades i entrada/sortida.
    - "Java.swing" i "Java.awt" per a interfícies gràfiques (*GUI*).

Al 2006, Sun Microsystems va publicar *Java OpenJDK*, una implementació de referència sota llicència *GNU GPL*, que va fer que Java esdevingués oficialment de codi obert (*open source*).

1.1) Predecessor

Com a predecessor de Java tenim C++, ja que com he mencionat, Java s'hi va inspirar.

1.2) Successor

Com a successor tenim Kotlin, un llenguatge creat al 2011 que funciona sobre la *JVM* i que ha estat dissenyat per ser completament interoperable amb

Java, solucionant moltes de les seves limitacions. Es va llançar oficialment el 2011 per *JetBrains* i al 2017 va ser declarat "*llenguatge preferit per al desenvolupament d'Android*".

2) Propòsit i paradigmes

Java és un llenguatge de programació multiparadigma, inicialment dissenyat com un llenguatge *imperatiu* i *orientat a objectes*. Amb l'evolució de les seves versions, ha adoptat característiques d'altres paradigmes, ampliant així les seves capacitats i flexibilitat. A continuació es descriuen 5 dels paradigmes que suporta Java, encara que n'hi ha molts més.

2.1) Paradigma imperatiu

Java segueix el paradigma imperatiu perquè permet indicar pas a pas les instruccions que el sistema ha d'executar per tal d'assolir un objectiu. Aquestes instruccions s'executen sobre variables, condicions, bucles i altres estructures de control per definir el flux d'execució del programa.

2.2) Paradigma orientat objectes

Al haver heretat conceptes de C++, Java és evidentment orientat a objectes. Conté classes que permeten encapsular la informació i simplificar l'estructura del codi mitjançant l'abstracció. Aquestes característiques fan que Java sigui ideal pel desenvolupament de projectes complexos.

2.3) Paradigma funcional

Al publicar-se *Java 8,* es van afegir *funcions lambda* i *interfícies funcionals* al llenguatge, afegint el paradigma funcional a la llista de paradigmes. Amb aquestes eines Java pot programar-se de forma declarativa, ja que des de llavors es pot indicar quin és l'objectiu sense haver d'indicar totes les instruccions necessàries per a arribar-hi.

2.4) Paradigma concurrent

També es poden gestionar múltiples tasques simultàniament o amb intervals de temps intercalats fent servir l'API de concurrència *"java.util.concurrent"* afegida a Java 5 al 2004.

2.5) Paradigma reflexiu

Java permet manipular elements del programa durant l'execució del codi. Amb l'API de reflexió (introduïda a Java 1.1 al 1997), es pot accedir a informació de classes, mètodes i atributs de manera dinàmica. Aquestes

eines funcionen inclús per a depurar programes, encara que no és el seu objectiu principal.

3) Sistema d'execució

El compilador de Java "*Javac*" inclòs al *JDK,* genera a partir de fitxers font (.java) fitxers emmagatzemats en format *bytecode* (.class). Aquest format és independent del sistema, fet que facilita la portabilitat dels programes. Després de compilar, la *JVM* s'encarrega d'interpretar els fitxers bytecode.

Per millorar el rendiment, java usa la compilació "*just-in-time"* que, en lloc d'interpretar els fitxers bytecode, els converteix en llenguatge assemblador durant l'execució del programa. Aquesta optimització millora l'eficiència respecte l'estructura del paràgraf anterior i és específica per cada sistema.

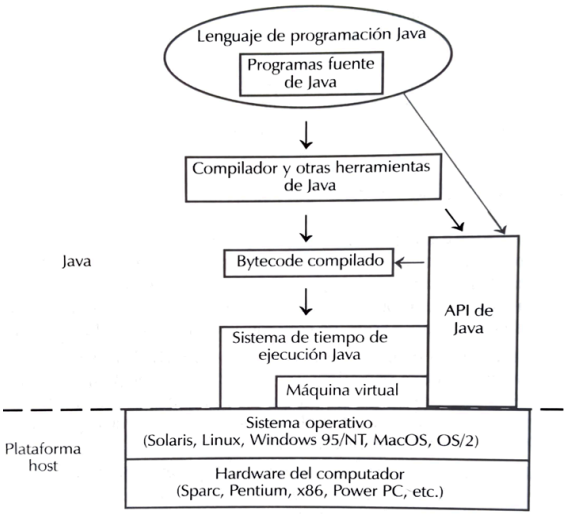

Fig 1: Procés d'execució a Java

4) Tipatge i tipus estàtics i dinàmics

És important diferenciar el tipatge i el tipus. El tipatge fa referència a la manera com els llenguatges gestionen els tipus de dades, mentre que el tipus defineix el format encapsulat de les dades encapsulades en una variable (per exemple, un enter o una cadena de caràcters).

Java segueix un tipatge *fort* i *estàtic*, ja que el tipus de cada variable es defineix durant la compilació i s'assegura que les operacions entre tipus incompatibles no es puguin realitzar, evitant així errors en temps d'execució. Cal recalcar que, a partir de *Java 10*, es va introduir la possibilitat d'inferir el tipus de les variables amb l'ús de "*var"*.

Java usa els tipus estàtics i dinàmics en diferents moments, usant el *late binding* (vist a classe). Aquesta característica permet que, quan un objecte té mètodes amb el mateix nom en una classe pare i una classe fill, es determini en temps d'execució quin mètode s'ha d'executar, en funció del tipus dinàmic de l'objecte. Així, en temps

de compilació es comprova el tipus estàtic, però és en temps d'execució quan el tipus dinàmic defineix el comportament del mètode a invocar.

5) Elements bàsics

5.1) Tipus de dades bàsics

Java disposa de 2 tipus principals:

- **Tipus primitius:** Són tipus simples, emmagatzemen únicament valors i són més eficients.

- **Tipus objecte:** Són instàncies de classes, pel que emmagatzemen mètodes addicionals, són menys eficients però més còmodes d'utilitzar.

Amb la sortida de *Java 5* es va afegir la possibilitat de convertir els tipus primitius a tipus objecte o viceversa, aquests canvis es poden fer amb *autoboxing* o *unboxing*, respectivament.

5.2) Ús de packages

Quan treballem amb diversos directorios en un projecte, és necessari organitzar les classes mitjançant una estructura de packages.

Per exemple, si volem usar des de "/home/A/classeA.java" una altra classe situada a un directori "/home/B/classeB.java", caldria:

- Que classeA inclogués la sentència "import B.classeB;" per localitzar classeB.
- Que classeB inclogués al principi del document "package B;" per definir l'ubicació de la classe.

El *classpath* a configurar seria el directori que contingui tots directoris i fitxers del projecte. A l'exemple anterior seria "/home".

5.3) Visibilitat de dades

A Java els atributs i mètodes poden tenir diferents modificadors de visibilitat:

- **public:** Dades accessibles des de qualsevol classe, independentment del paquet al qual pertanyin.

- **private:** Dades accessibles únicament des de la mateixa classe.

- **protected:** Dades accessibles des de la mateixa classe, des de classes del mateix paquet i des de classes fill.

- **Sense especificar:** En cas que no s'especifiqui cap tipus de visibilitat, les dades són accessibles des de qualsevol classe dins del mateix paquet.

6) Elements de programació funcional

A continuació s'expliquen els elements necessaris per programar funcionalment a Java. A l'apartat 8 del projecte es poden veure alguns exemples d'usos de programació funcional. Comparant els codis d'ambdós apartats es pot apreciar com s'escriu bastant més codi. Per fets com aquest, Java es sol considerar farragós.

6.1) Funcions lambda

Aquestes eines, permeten definir funcions anònimes. Es solen usar quan es volen fer operacions senzilles o es volen passar funcions simples com a arguments. Es declaren com "(paràmetres) -> {cos}" Un exemple d'aquestes funcions podria ser, per exemple, sumar 1 a un enter: `x -> x + 1`

6.2) Funcions d'ordre superior

Les funcions d'ordre superior reben altres funcions com a entrada o retornen una funció. Alguns exemples podrien ser funcions com ara *filter* o *map* (vistes a classe amb Haskell). Per poder aplicar funcions d'ordre superior sobre dades han de ser dades en format "*Stream*", que permet aplicar operacions funcionals. Un exemple de funció d'ordre superior es filter, que es pot usar amb una funció lambda com a paràmetre: `list.filter(n -> n % 2 == 0)`

6.3) Interfícies funcionals

Una interfície funcional és aquella que emmagatzema un mètode abstracte. Aquestes interfícies són fonamentals per la programació funcional en Java, ja que simplifiquen l'ús de lambda expressions. Es poden declarar o importar de java.util.function exemples:

- **Function<T, R>:** rep un valor de tipus T i en retorna un de tipus R.
  `Function<Integer, String> toString = x -> "Valor: " + x`

- **BiFunction<K,V, R>:** rep dos valors de tipus K i V i en retorna un de tipus R.

```
BiFunction<Integer, Integer, Integer> suma= (x,y) -> x + y
```

- **Predicate<T>:** rep un valor de tipus T i retorna un booleà.

```
Predicate<Integer> isEven = x -> x % 2 == 0
```

- **Consumer<T>:** rep un valor de tipus T i no té valor de retorn.

```
Consumer<String> print = x -> System.out.println(x)
```

- **Supplier<T>:** no rep cap valor i retorna un valor de tipus T.

```
Supplier<Double> randomValue = () -> Math.random()
```

7) Capacitats i limitacions dels elements de programació funcional

Capacitats

- Declarativitat
  Amb els elements funcionals, java pot usar un codi declaratiu sense estar forçat a usar l'imperatiu, com anteriorment.

- Funcions d'ordre superior
  Les funcions d'ordre superior permeten manipular col·leccions i fluxos de dades de manera funcional.

- Compatibilitat amb els altres paradigmes
  Java permet combinar el paradigma funcional amb el paradigma orientat a objectes. Això facilita la integració de funcionalitats noves en codi ja existent sense necessitat de reestructurar tot el projecte.

- Compatibilitat amb interfícies existents
  Les interfícies funcionals faciliten l'aplicació de conceptes funcionals a biblioteques ja existents de Java.

Limitacions

- Complexitat
  Tot i adaptar-se a la programació funcional, el codi pot ser més llarg i complex, ja que cal treballar amb interfícies funcionals, streams i operacions encadenades, especialment per als desenvolupadors sense experiència prèvia en paradigmes funcionals.

- No és un llenguatge funcional pur
  Java és multiparadigma i permet modificar l'estat de variables o estructures dins de les funcions, fet que va en contra dels principis de la programació funcional pura.

8) Exemples de codi

8.1) Hello world

```
class HelloWorld{
    public static void main (String args[]) {
        System.out.println("Hola món!");
    }
}
$ javac HelloWorld.java
$ java HelloWorld
Hola món!
```

8.2) Factorial

```
public class Factorial {

    public static int factorial(int n) {
        if (n == 0 || n == 1) return 1;
        return factorial(n-1)*n;
    }
    public static void main(String[] args) {

        for (int i = 0; i <= 5; ++i) {
            System.out.println("Factorial de " + i + ": " + factorial(i));
        }
    }
}
Factorial de 0: 1
Factorial de 1: 1
Factorial de 2: 2
Factorial de 3: 6
Factorial de 4: 24
Factorial de 5: 120
```

8.3) Nombres de Fibonacci

```
public class Fibonacci {
    public static int fib(int n) {
        if (n == 0 || n == 1) return n;
        return fib(n - 1) + fib(n - 2);
    }



    public static void main(String[] args) {
        for (int i = 0; i <= 5; ++i) {
```

```
            System.out.println("Fibonacci de " + i + ": " + fib(i));
        }
    }
}
$ javac Fibonacci.java
$ java Fibonacci
Fibonacci de 0: 0
Fibonacci de 1: 1
Fibonacci de 2: 1
Fibonacci de 3: 2
Fibonacci de 4: 3
Fibonacci de 5: 5
```

8.4) Late Binding

```
class Animal {
    void parlar() {
        System.out.println("grr");
    }
}
class Gat extends Animal {
    void parlar() {
        System.out.println("mèu");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal g;
        g = new Gat();
        g.parlar();
    }
}
$ javac Herencia.java
$ java Herencia
mèu
```

8.5) Elements funcionals

8.5.1) Exemple amb interficies funcionals

```
import java.util.function.Function;

public class Funcional 1 {

    public static void main(String[] args) {
       Function<Integer, Integer> sumar = x -> x + 1;
       System.out.println(sumar.apply(1));
    }
}
$ Javac Funcional1.java
$ Java Funcional1
2
```

8.5.2) Exemple (funcions d'ordre superior i lambda expressions)

```
import Java.util.Arrays;
import Java.util.List;
import Java.util.stream.Collectors;

public class Funcional2 {
    Funcional2() {
         List<Integer> nombres = Arrays.asList(1,2,3,4,5,6,7,8,9,10);
        //resultat s'ha inferit com un a List<Integer> per mitjà de var
         var resultat = nombres.stream()
              .filter(n -> n % 2 == 0)//Filtrar parells
              .map(n -> 2*n)// Multiplicar per 2
              .collect(Collectors.toList());// Stream -> List

        System.out.println(resultat);

    }
    public static void main(String[] args) {
        Funcional f = new Funcional2();
    }
}
$ Javac Funcional2.java
$ Java Funcional2
[4, 8, 12, 16, 20]
```

9) Referències

Llibre

- Jaworski, J. (1997). Java™ Guía de desarrollo. Sams Publishing.
- **Fig 1** Jaworski, J. (1997). Java: Guía de desarrollo (pàg. 5). Sams Publishing.

Web

- Contributors to Wikimedia projects. (2024, October 12). Java (llenguatge de programació).Viquipèdia, L'enciclopèdia Lliure. https://ca.wikipedia.org/wiki/Java_(llenguatge_de_programació)
- Contributors to Wikimedia projects. (2024, March 23). Kotlin. Viquipèdia, L'enciclopèdia Lliure. https://ca.wikipedia.org/wiki/Kotlin

- Java SE Specifications. (n.d.). https://docs.oracle.com/Javase/specs/
- Lesson: Concurrency (The JavaTM Tutorials > Essential Java Classes). (n.d.). https://docs.oracle.com/javase/tutorial/essential/concurrency/
- Primitive data types (The JavaTM Tutorials > Learning the Java Language > Language Basics). (n.d.). https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

- Autoboxing and unboxing (The JavaTM Tutorials > Learning the Java Language > Numbers and Strings). (n.d.). https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html

- CloudDevs. (2024, January 4). Java Lambdas and Streams: Functional Programming Paradigm. https://clouddevs.com/Java/lambdas-and-streams/

- Conceptes avançats. (n.d.). https://gebakx.github.io/lp/avancats.html#1

10) Qualitat de la informació obtinguda les fonts citades

- Jaworski, J. (1997). Java™ Guía de desarrollo. Sams Publishing

Aquest llibre m'ha estat molt útil per entendre història del llenguatge i el procés de compilació d'un sistema basat en Java. M'han estat útil també exemples de codi, en els que m'he inspirat per crear els exemples que he proporcionat a l'apartat 8.

- Viquipèdia

Aquesta font no és una font fiable per extreure informació però m'ha donat idees generals i conceptes sobre els que poder fer recerca de Java. També m'ha anat bé per omplir el subapartat de Kotlin, ja que necessitava informació general. En definitiva, m'és útil per dirigir la meva recerca sobre informació però no puc treure informació únicament d'aquesta font perquè per si sola no és prou fiable.

- Oracle

La web d'Oracle m'ha estat útil ja que m'ha proporcionat una documentació extensa per moltes versions de java. En un inici, al veure l'apartat d'especificacions, no em va satisfer com a font, però més endavant vaig veure que en adreces més concretes hi ha informació ben explicada. He posat posat 3 exemples de URLs més concretes on he accedit (Concurrency, Primitive data types i Autoboxing and unboxing), però n'hi ha moltíssimes a Oracle.

- CloudDevs

També he accedit a webs més concretes, com ara CloudDevs, que m'han ajudat a la programació funcional en Java, entre d'altres. Aquesta font coneguda per publicar articles tècnics relacionats amb el desenvolupament de programari, pel que l'he considerat fiable. A més, al provar els exemples que m'oferien al meu ordinador he pogut comprovar el funcionament de la informació obtinguda per aquesta font.

- Diapositives de classe (Github Gerard Escudero)

Les disapositives de conceptes avançats d'en Gerard Escudero, m'han estat útils al explicar el comportament de Java en el late binding, per exemple. És una font fiable ja que va ser redactada per un home especialitzat en aquest camp.