Manual de Java - Tractament de fitxers

1 Introducció

Al llarg de la història, l'ésser humà ha hagut de poder deixar constància dels seus pensaments o vivències en algun suport físic. D'aquesta forma, queda immortalitzat (sempre que el dispositiu ho ha permès) i es pot transmetre a altres persones o bé recuperar-ho un mateix un cop ja ha estat oblidat.

En aquest sentit, històricament hem pogut veure diferents dispositius com pedres tallades, papirs, llibres, cintes de gramòfon, VHS (entre d'altres) i, a l'actualitat, dispositius digitals.

Al llarg d'aquesta unitat formativa veurem la utilitat i el funcionament del tractament de fitxers en sistemes digitals, desenvolupant mòduls que realitzin aquestes tasques. Per exemple, podem utilitzar els fitxers com a font de dades, com a mostratge de les dades, com a desa de dades per a una futura execució, com a configuració del propi programa, etc. Així, podem diferenciar els fitxers en fitxers de **dades** i fitxers de **programa**.

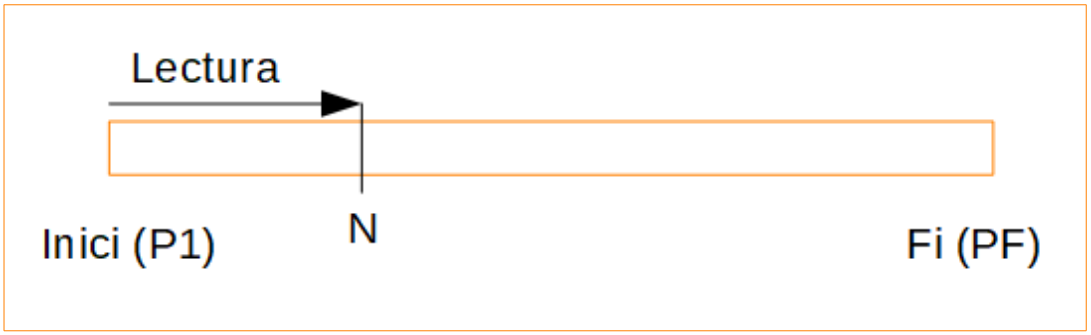
Accessos a les dades

La forma d'accedir a la informació en tots aquests dispositius també ha variat al llarg d'aquest temps. D'aquesta manera, tenim dos mètodes per accedir a les dades dels fitxers: l'accés **seqüencial** i l'accés **directe**.

Accés seqüencial

És la forma més simple d'accedir a la informació. En aquest mètode d'accés, la informació es llegeix de forma successiva des del primer punt.

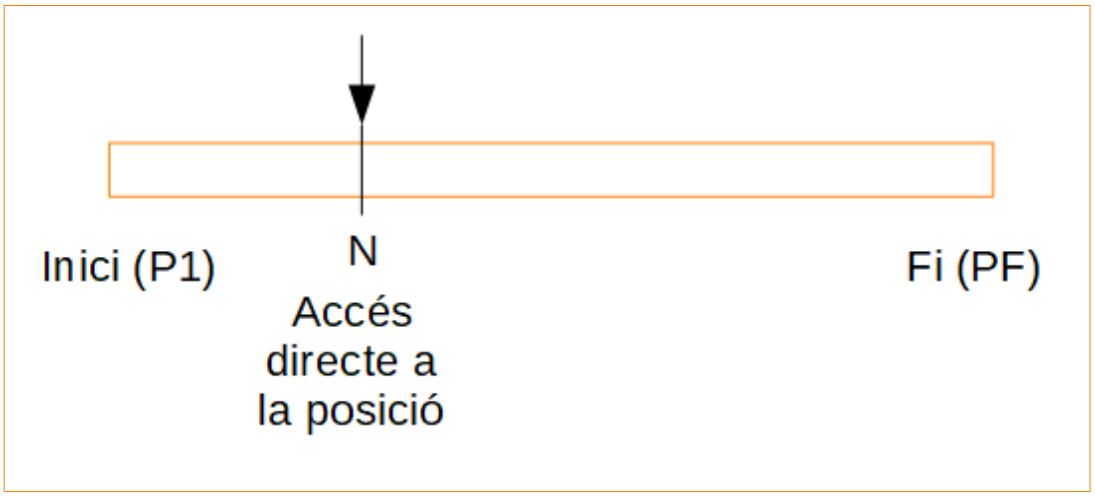
D'aquesta forma si volem accedir a la informació emmagatzemada a un punt N del fitxer, primer haurem de llegir les N-1 posicions anteriors:



És el mètode d'accés utilitzat als sistemes més antics, com el VHS i el CD.

Accés directe o relatiu

És la forma més ràpida d'accedir a la informació. L'accés a les dades es fa accedint directament a la seva posició, especificant la posició relativa dins d'un conjunt de posicions. D'aquesta forma, si volem accedir a la posició N d'un fitxer, no haurem de d'accedir a totes les posicions anteriors. No obstant, haurem de tenir especificades on estan cada posició del fitxer:



Operacions bàsiques sobre fitxers

A l'hora de realitzar operacions sobre fitxers, hi ha un seguit d'operacions bàsiques que es poden realitzar:

- Obertura: s'obre el fitxer per poder ser tractat.
- Tancament: es tanca el fitxer i ja no es poden realitzar cap tipus d'operacions sobre ell.
- Escriptura: s'escriu contingut dins del fitxer.
- Lectura/consulta: es llegeix el contingut del fitxer.
- Destrucció: s'esborra el fitxer.
- Adjuntar: s'afegeix contingut al fitxer. És similar a la operació d'escriure, però en aquest cas s'afegeix al fitxer i no existeix la possibilitat de sobreescriure la informació existent.

2 Ús de fitxers a Java

Lectura i escriptura de fitxers

A Java, disposem d'una sèrie de llibreries bàsiques incloses a la *JRE* i *JDK* per tal de poder gestionar l'escriptura i lectura de dades a fitxers.

Totes aquestes llibreries, com hem vist fins ara, les hem d'importar:

```
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.util.Scanner;

import java.io.IOException;
```

De totes les que hem importat, no sempre les necessitarem totes:

- java.io.File – Classe que defineix de forma abstracta un fitxer i la seva ruta. Serveix, principalment, per gestionar aspectes externs als documents (permisos, existència, creació, destrucció, etc.). Per gestionar escriptura i lectura, s'utilitzen les classes següents.
- java.io.FileReader – Classe que ens permetrà llegir el text d'un document.
- java.io.FileWriter – Classe que ens permetrà escriure el text a un document.
- java.io.BufferedReader – Classe que ens habilitarà llegir el text d'un document emmagatzemant en memòria temporal (*buffering*) els caràcters per tal de realitzar una lectura més eficient que amb la classe FileReader. Basa el seu ús en objectes de la classe FileReader.
- java.io.BufferedWriter – Classe que ens habilitarà escriure el text a document emmagatzemant en memòria temporal (*buffering*) els caràcters per tal de realitzar una escriptura més eficient que amb la classe FileWriter. Basa el seu ús en objectes de la classe FileWriter.
- java.util.Scanner – Amb la llibreria Scanner, que hem utilitzat per llegir informació del terminal, podem llegir de forma fàcil el document línia a línia o paraula a paraula, com si estigués escrit al terminal.
- java.io.IOException – És una llibreria per poder controlar les excepcions (errors no controlats) durant l'execució de tasques d'entrada i sortida de dades (*IO*).

En aquest aspecte, per realitzar aquestes tasques necessitarem introduir una estructura *try/catch* per poder tractar els errors no controlats que retorni l'execució de les aplicacions:

```
try
{
    // Codi a executar
}
catch (IOException e)
{
    // Codi a executar en cas d'error
}
```

Segons aquesta estructura, s'executarà el codi *try* (el codi bàsic a executar). En el cas que hi hagi un error no controlat (excepció), s'executarà el codi comprès al *catch*. Un exemple:

```
try
{
    File file = new File("Ruta al fitxer de text");
    FileWriter writer = new FileWriter(file, true);
    writer.write("Text a afegir");
    writer.close();
}
catch (IOException e)
{
    System.out.println(e.toString());
}
```

Comprovar si existeix un fitxer

Per comprovar si existeix un fitxer, hem de generar una variable de tipus *File* amb la ruta del fitxer a comprovar. Un cop tenim aquesta variable, podem comprovar si existeix amb la funció *exists()*:

```
File file = new File("Ruta del fitxer");
file.exists();
```

En aquest aspecte, el caràcter contrabarra (\) és el caràcter que denota els caràcters d'escapament (\n – línia nova, \t, - tabulacions, etc.). Per aquest motiu, quan introduïm una ruta a Windows hem d'utilitzar la doble contrabarra (\\) per marcar la contrabarra simple (la primera contrabarra anul·la l'efecte de caràcter d'escapament de la segona). Un exemple de ruta:

```
File file = new File("C:\\Users\\Carpeta\\fitxer.txt");
```

Crear un fitxer

Per crear un fitxer, hem de generar una variable de tipus *File* amb la ruta del fitxer a crear. Un cop tenim aquesta variable, podem crear el fitxer amb la funció *createNewFile()*:

```
File file = new File("Ruta del fitxer");
file.createNewFile();
```

Obrir fitxer

Per obrir els fitxers, hem de generar les variables adients segons l'acció que hem de realitzar:

```
File file = new File("Ruta del fitxer");
FileReader reader = new FileReader(file);
FileWriter writer = new FileWriter(file);
Scanner sc = new Scanner(file);
BufferedReader BFreader = new BufferedReader(reader);
BufferedWriter BFwriter = new BufferedWriter(writer);
```

D'aquesta forma, primer de tot generarem la instància abstracta que gestionarà la ubicació del fitxer.

Si volem llegir dades, haurem d'obrir el fitxer utilitzant la classe *FileReader* i, si ho volem fer de forma eficient, després generant el *buffer* amb la classe BufferedReader.

Si volem llegir-les amb la classe *Scanner*, només haurem de generar la variable de tipus *Scanner* com si anéssim a llegir dades pel terminal. En aquest cas, en comptes d'utilitzar l'entrada del sistema (*System.In*), utilitzarem la instància que haurem generat de la classe *File*.

Si volem escriure dades, haurem d'obrir el fitxer utilitzant la classe *FileWriter* i, si ho volem fer de forma eficient, després generant el *buffer* amb la classe BufferedWriter.

Tancar fitxer

Per tancar els fitxers només haurem de cridar a la funció *close()* de cadascuna de les diferents instàncies:

```
sc.close();
reader.close();
writer.close();
BFreader.close();
BFwriter.close();
```

Llegir fitxer

Per llegir els fitxers amb la classe *FileReader*, un cop tenim generada la variable, podrem llegir cadascun dels caràcters del document amb la funció *read()*:

```
FileReader reader = new FileReader(file);
int lectura = reader.read();
while (lectura != -1)
{
    System.out.println((char)lectura);
    lectura = reader.read();
}
```

En aquest cas, la classe *FileReader* llegeix cada caràcter en format **numèric**. D'aquesta forma, si volem convertir el valor enter en un caràcter, haurem de realitzar una conversió explícita (recordeu-ho de la UF1). El valor que retorna en cas que arribi al final del document és el valor **-1**.

Per solucionar aquest problema, podem realitzar la lectura amb la classe *BufferedReader* mitjançant la funció *readLine()*:

```
try {
    File file = new File("Ruta al fitxer de text");
    FileReader reader = new FileReader(file);
    BufferedReader BFreader = new BufferedReader(reader);
    String linia = BFreader.readLine();
    while (linia != null) {
        System.out.println(linia);
        linia = BFreader.readLine();
    }
    BFreader.close();
}
catch (IOException e) {
    e.printStackTrace();
}
```

En aquest cas, llegirem tota la línia sencera com a **text**. El valor que retorna en cas que arribi al final del document és el valor **null**.

Per llegir els fitxers amb la classe *Scanner*, el funcionament és el mateix com si estiguéssim escrivint el text per pantalla:

```
Scanner sc = new Scanner(file);
while (sc.hasNextLine()) {
    System.out.println(sc.nextLine());
}
```

Escriure fitxer

Per llegir els fitxers amb la classe *FileWriter*, un cop tenim generada la variable, podrem llegir cadascun dels caràcters del document amb la funció *write()*:

```
FileWriter writer = new FileWriter(file);
writer.write("Text a escriure");
```

Per utilitzar la classe *BufferedWriter*, la funció és la mateixa (*write()*):

```
try
{
    File file = new File("Ruta al fitxer de text");
    FileWriter writer = new FileWriter(file);
    BufferedWriter BFwriter = new BufferedWriter(writer);
    BFwriter.write("Text a afegir");
    BFwriter.close();
}
catch (IOException e)
{
    System.out.println(e.toString());
}
```

Cal comentar que, en aquests dos casos, el **text** que hem d'escriure **sobreescriurà** el **text existent**.

A més, si **no tanquem** els documents, els canvis **no es desaran** en aquests.

Afegir text al final del fitxer

Per afegir el text al final del document en comptes de sobreescriure'l, només haurem de passar un booleà ***true*** com a **segon paràmetre** a la declaració de la variable *FileWriter:*

```
FileWriter writer = new FileWriter(file, true);
writer.write("Text a afegir al final");
```

```
try
{
    File file = new File("Ruta al fitxer de text");
    FileWriter writer = new FileWriter(file, true);
    BufferedWriter BFwriter = new BufferedWriter(writer);
    BFwriter.write("Text a afegir");
    BFwriter.close();
}
catch (IOException e)
{
    System.out.println(e.toString());
}
```

Esborrar fitxer

Per esborrar un fitxer, hem de generar una variable de tipus *File* amb la ruta del fitxer a esborrar. Un cop tenim aquesta variable, podem esborrar el fitxer amb la funció *delete()*:

```
File file = new File("Ruta del fitxer");
file.delete();
```

Fitxers codificats d'imatge

En molts dels llenguatges de programació, els fitxers d'imatges es poden tractar com a fitxers de matrius bidimensionals de dades.

A Java, tenim dues llibreries que ens poden ajudar a tractar-les:

```
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
```

Aquestes dues llibreries gestionen:

- java.awt.image.BufferedImage – aquesta classe ens permetrà gestionar l'accés als valors dels píxels de les imatges que hem llegit. També ens permetrà modificar els seus valors.
- Javax.imageio.ImageIO – aquesta classe ens permetrà gestionar la lectura i escriptura de les imatges al sistema. Per poder obrir una imatge, a més, necessitarem la classe *File* que hem vist anteriorment.

Obrir imatge

Per tal d'obrir una imatge, haurem de generar una variable de tipus *File* (com en el cas dels fitxers normals de text).

Un cop tenim aquesta variable generada, hem de generar una variable de tipus *BufferedImage* (semblant a la *BufferedReader*) per tal de poder llegir el contingut de la imatge. Per llegir-la com a tal, haurem d'utilitzar la funció *read()*:

```
BufferedImage image = null;
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
try
{
    File input_file = new File(ruta);
    image = ImageIO.read(input_file);
}
catch(IOException e)
{
    System.out.println("Error: "+e);
}
```

Per declarar la variable de tipus BufferedImage (la variable image de l'exemple), necessitarem els següents paràmetres:

- ***Width***, que serà l'especificació de **l'amplada** en píxels de la imatge.
- ***Height***, que serà l'especificació de **l'alçada** en píxels de la imatge.
- **Codificació**, que serà la codificació de color. En el nostre cas, utilitzarem la codificació Alpha-Vermell-Verd-Blau (*ARGB*) en comptes de la *RGB* clàssica, i haurà de ser de tipus ***BufferedImage.TYPE_INT_ARGB*** (com a l'exemple).

Un cop definides les metadades de la imatge ja la podrem llegir. Per llegir-la, l'únic paràmetre que ha de tenir la funció *read* és la variable de tipus *File* que gestiona la ruta i el nom del fitxer de la imatge.

Tancar i escriure imatge

Per tal d'escriure la imatge, també haurem de generar una variable de tipus *File*. Que marcarà la ruta on haurem d'escriure la imatge.

A més necessitarem fer una crida a la funció ***write*** de la classe *ImageIO*:

```
try
{
    File output_file = new File(ruta);
    ImageIO.write(image, "jpg", output_file);
}
catch(IOException e)
{
    System.out.println("Error: "+e);
}
```

Aquesta funció *write* té tres paràmetres principals:

- La **imatge**, que haurà de ser una variable de tipus *BufferedImage* amb la informació de la imatge.
- El **format** en el que codificarem la imatge (per norma general utilitzarem *jpg*).
- La **ruta**, que serà una variable de tipus *File* on hi haurà especificada la ruta i el nom de la imatge al sistema.

Visualitzar contingut dels píxels d'una imatge

Una imatge, a Java, és tractada com a una matriu de MxN píxels (amplada per alçada). D'aquesta forma, si tenim la variable de tipus *BufferedImage image*, podem extreure el valor del píxel en una posició amb la funció *getRGB*:

```
for (int ii = 0; ii < width; ii++)
{
    for (int jj = 0; jj < height; jj++)
    {
        image.getRGB(ii,jj);
    }
}
```

No obstant, aquest valor serà un enter (32 bits), con cada 8 bits correspon a un valor (canal alfa, color vermell, color verd i color blau):

```
int[] pARGB = new int[4];
pARGB[0] = (pixel >> 24) & 0xff; // canal Alfa
pARGB[1] = (pixel >> 16) & 0xff; // vermell
pARGB[2] = (pixel >> 8) & 0xff;  // verd
pARGB[3] = (pixel) & 0xff;       // blau
```

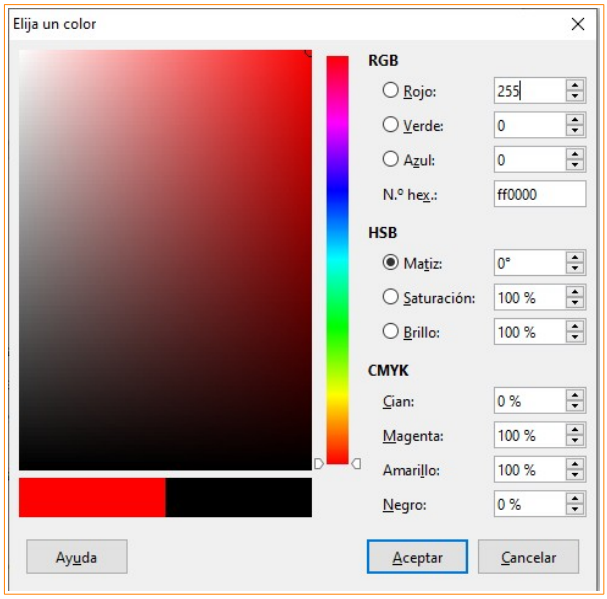
Per exemple, el color blanc pur, que a l'espai RGB tindria el valor 255 a les tres capes de color, està codificat amb el valor -1. Això és degut a que la codificació dels negatius en Java utilitza el complement a 2, i com els enters són 32 bits, això resulta en la codificació 11111111111111111111111111111111:

- En lila, els bits corresponents al canal Alfa (11111111 → 255)
- En vermell, els bits corresponents a la capa vermella de color (11111111 → 255)
- En verd, els bits corresponents a la capa verda de color (11111111 → 255)
- En blau, els bits corresponents a la capa blava de color (11111111 → 255)

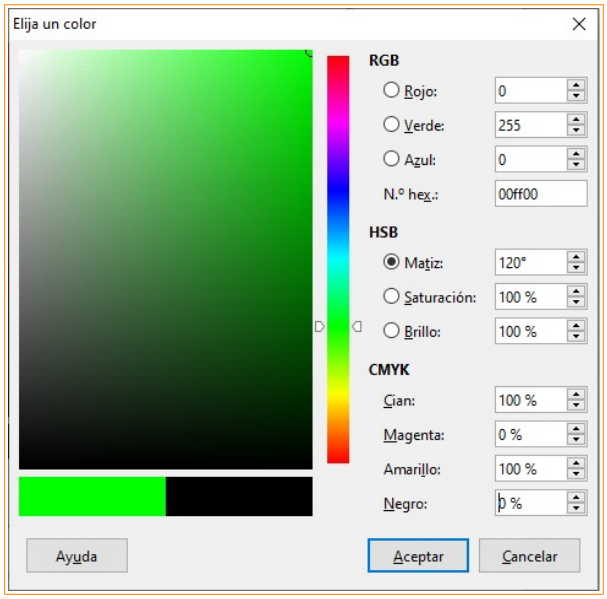
D'aquesta forma, realitzant el moviment de bits adient (com vam veure a la UF1) podem extreure cada un d'aquests valors de forma fàcil.

Cal dir, que l'escala de colors RGB està compresa entre el valor 0 (negre) i el valor 255 (el color més pur):

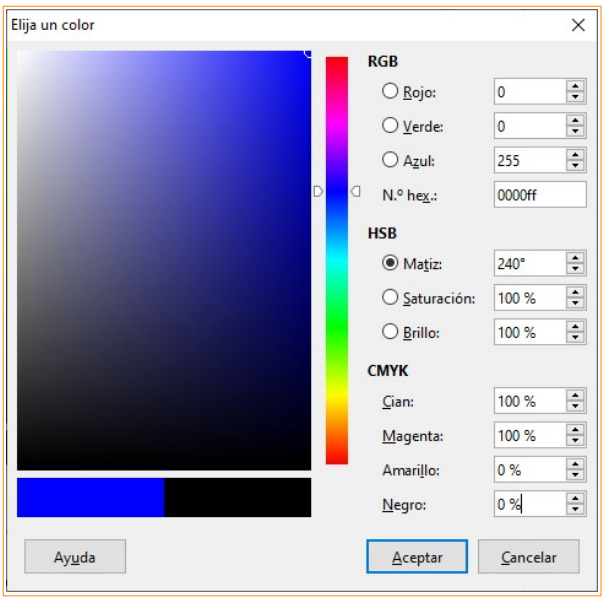
- Canal vermell: 0 → negre, 255 → color vermell pur



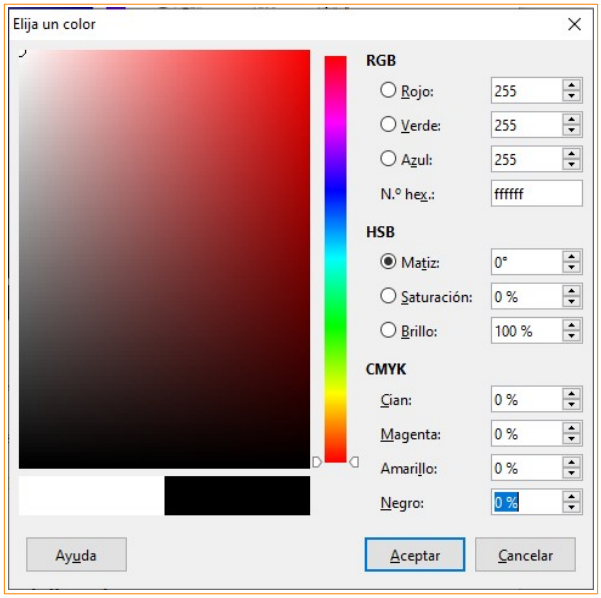
- Canal verd: 0 → negre, 255 → color verd pur



- Canal blau: 0 → negre, 255 → color blau pur



- Qualsevol combinació de valors dins d'aquests rangs és l'addició de colors per generar tota l'escala cromàtica. La suma dels tres valors 255 (la suma de vermell, verd i blau més purs) donarà com a resultat el color blanc:



Modificar el contingut del píxel d'una imatge

Per muntar el valor que desitgem en un píxel, només haurem de fer el pas invers de la lectura.

En aquest cas, haurem de fer córrer els blocs de 8 bits el número de posicions, unint amb l'operand | tots els valors, que desitgem fins obtenir el enter codificat de 32 bits:

```
int pixel = 0;
pixel = (pARGB[0] << 24) | (pARGB[1] << 16) | (pARGB[2] << 8) | (pARGB[3]);
// pARGB[0] és el valor del canal Alfa
// pARGB[1] és el valor del píxel vermell
// pARGB[2] és el valor del píxel verd
// pARGB[1] és el valor del píxel blau

image.setRGB(ii,jj, pixel);
```

Per registrar el valor del píxel a la imatge, haurem de fer-ho amb la funció *setRGB*, passant com a paràmetre la posició del píxel a la imatge i el valor del color codificat corresponent.

Fitxers de configuració: XML

Un dels usos principals de la gestió de fitxers a l'hora de generar programes informàtics és modular aquests per poder configurar-los i modificar el seu funcionament sense modificar l'estructura del codi. D'aquesta forma, els fitxers de configuració són els encarregats de definir quin serà el funcionament propi del programa (dins d'uns marges).

Un dels principals tipus de fitxers utilitzats per aquest motiu són els fitxers XML. El llenguatge d'etiquetes és molt interessant a l'hora de generar fitxers de configuració degut a la classificació dels diferents paràmetres que es pot establir mitjançant l'arbre DOM que es genera dins del propi document.

Les principals llibreries que utilitzarem, a Java, són les següents:

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
```

Aquestes llibreries gestionen:

- javax.xml.parsers.DocumentBuilderFactory – Permet definir les instàncies necessàries per tal de poder generar un document DOM d'un document XML.
- javax.xml.parsers.DocumentBuilder – Permet definir, a partir de la instància DocumentBuilderFactory, l'objecte de tipus Document que contindrà la modularització DOM del document XML.

- org.w3c.dom.Document – Interfície que ens permetrà emmagatzemar i gestionar les variables que emmagatzemin l'anàlisi i la segmentació del model DOM dels documents XML.
- org.w3c.dom.NodeList – Interfície que ens permetrà emmagatzemar i gestionar un llistat de nodes del document. Entenem com a nodes cada una dels punts o segments d'un document XML.
- org.w3c.dom.Node – Interfície que ens permetrà emmagatzemar i gestionar les característiques d'un dels nodes del document XML en concret.

Per poder utilitzar totes aquestes classes i interfícies haurem de seguir una sèrie de passes:

1. Generar la variable de tipus File que contindrà la declaració de la ruta al fitxer (com en els casos anteriors).
2. Generar la variable de tipus *DocumentBuilderFactory*, que emmagatzemarà la instància que utilitzarem per generar el document DOM.
3. Generar la variable de tipus *DocumentBuilder*, que utilitzarem per poder analitzar el document XML.
4. Generar la variable de tipus *Document*, que serà on emmagatzemarem l'anàlisi i la transformació de document XML a un objecte DOM intel·ligible pel llenguatge de programació.
5. Recordeu realitzar l'accés al fitxer utilitzant una estructura *try/catch* per tal de poder gestionar les possibles excepcions que retorni el codi.

```
File fXmlFile = new File("ruta al fitxer XML");
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(fXmlFile);
```

Un cop tenim el document analitzat, podem anar a buscar tots els nodes que tinguin una etiqueta amb la funció *getElementByTagName*:

```
NodeList nList = doc.getElementsByTagName("nom_etiqueta");
```

Aquesta funció ens retornarà **una llista** (una estructura semblant a un vector, les veurem més endavant) amb la informació de tots els nodes que tenen aquest nom. D'aquesta forma, si volem avaluar-los, haurem de recórrer a un bucle per tal de poder accedir a la informació de cadascun per separat:

```
for (int i = 0; i < nList.getLength(); i++)
{
    Node nNode = nList.item(i);
    System.out.println("Element actual: " + nNode.getNodeName());
    System.out.println("Valor actual: " + nNode.getTextContent());
}
```

Amb la funció *getTextContent*, podrem obtenir el valor especificat entre l'etiqueta que obre la secció i la que la tanca. Per exemple:

- <etiqueta1>Hola món</etiqueta1>
  - Ens retornarà el valor **Hola món**

Si necessiteu més informació podeu consultar el següent enllaç:

- https://mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/