FILES and SERIALIZATION

UNIT 1

Xavier Ibáñez Català

Contents

- Files. Reading and Writing
- Serialization

Glossary

English	Spanish	English	Spanish
File	Archivo		
Folder	Directorio		
Path	Ruta		
Stream	Flujo		
Store	Almacenar		
Load	Cargar, recuperar		
Wrap	Envolver		
Casting	Conversión (de un tipo de dato a otro)		

Files. Types

- Binary:
  - Data is stored using bytes depending on the type of data (int, float…)
  - Cannot be read directly

- Text
  - Data is stored as character
  - Can be read directly (XML, JSON, etc.)

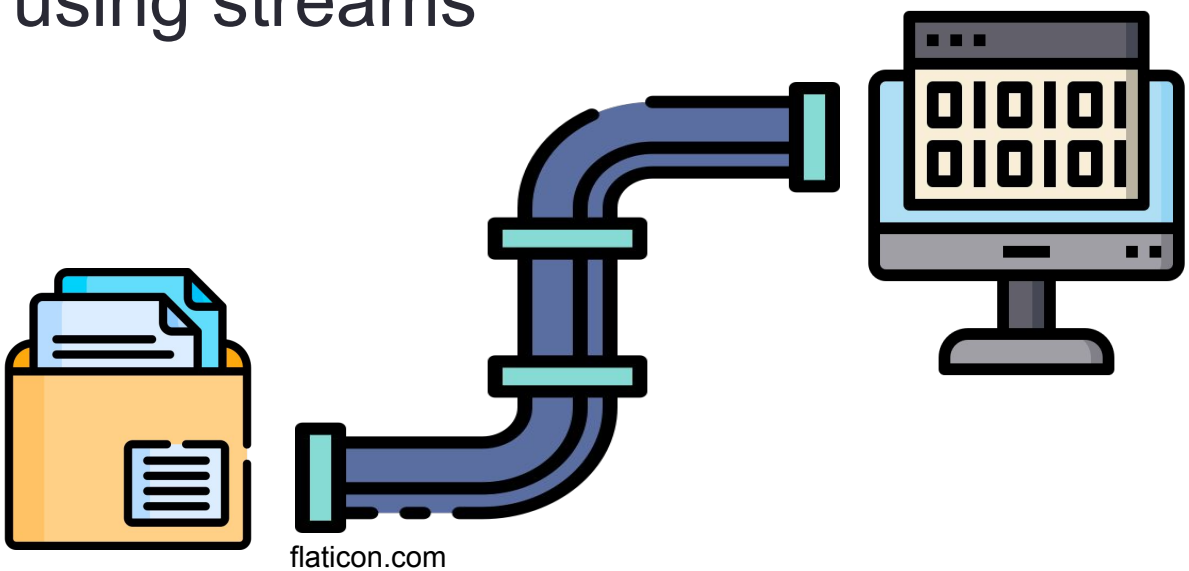
Files. Operations

- Read and write
  - Byte, byte arrays or data of a certain type
  - Character, char array or line

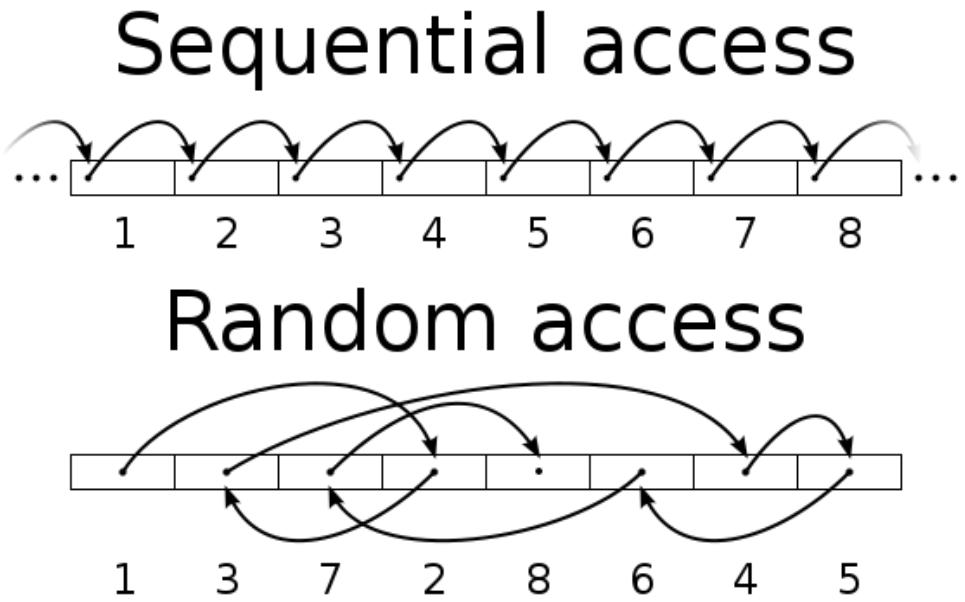
Files. Operations

- Read and write
  - Byte, byte arrays or data of a certain type
  - Character, char array or line

- Generally using streams



Files. Access



- Take a minute to think about the pseudocode of each access mode

Files. Access

- Sequential
  - *Open the file*
  - *Read from file*
  - *While not end of file*
    - *Process what you read*
    - *Read next*
  - *End of loop*
  - *Close file*

- Random access
  - *Open the file*
  - *As long as we want to process information*
    - *Move pointer*
    - *Read the register*
    - *Process the register*
  - *End of loop*
  - *Close file*

Files. Access

- Sequential
  - *Open the file*
  - *Read from file*
  - *While not end of file*
    - *Process what you read*
    - *Read next*
  - *End of loop*
  - *Close file*

- Random access
  - *Open the file*
  - *As long as we want to process information*
    - *Move pointer*
    - *Read the register*
    - *Process the register*
  - *End of loop*
  - *Close file*

Files. Access

Feature	Sequential Access	Random/Direct Access
Access order	Sequential	Random/direct
Speed of access to specific records	Slow	Fast
Flexibility for modifications	Low	High
Implementation complexity	Low	High
Typical usage	Batch processing, backups	Databases, interactive applications

Reading Text Files in Java

- Read one char: only FileReader

```
FileReader f = new FileReader ("doc.txt");
int i;
while ((i = f.read ())! = -1)
   System.out.println ((char) i);
f.close ();
```

- Read lines: FileReader + BufferedReader

```
FileReader f = new FileReader ("doc.txt");
BufferedReader l = new BufferedReader (f);
String line;
while ((line = l.readLine ())! = null)
   System.out.println (line);
f.close ();
```

Reading ALL LINES of Text Files

`List<String> Files.readAllLines (Path path)`

- Reads ALL lines of a text file
- Returns a List with one String per line
- Not recommended for BIG files

```
try {
    List<String> lines = Files.readAllLines(Paths.get("demo.txt"));
    for(String line: lines) {
        System.out.println(line);
    }
} catch (IOException ex) {
    System.out.format("I/O error: %s%n", ex);
}
```

Writing Text Files in Java

- Write text: FileWriter + BufferedWriter

```
File fw = new FileWriter("doc2.txt");
BufferedWriter f = new BufferedWriter(fw);
for (int i=0; i<10; i++)
  f.write("Línea número: " + x + "\n");
f.close();
```

- Write one char: only FileWriter
  - Use write() method of FileWriter

Reading Binary Files in Java

- FileInputStream + DataInputStream
  - FileInputStream.read(): reads 1 byte. Returns -1 when EOF
  - DataInputStream: methods to read primitive data types (reads and converts)
    - readChar(), readInt(), readFloat()…

```
File f = new File("nombredelfichero");
FileInputStream fi = new FileInputStream(f);
DataInputStream d = new DataInputStream(fi);
int i = d.readInt();
char c = d.readChar();
```

Writing Binary Files in Java

- FileOutputStream + DataOutputStream
  - FileOutputStream.write(): writes 1 byte.
  - DataOutputStream: methods to write primitive data types (converts and writes)
    - writeChar(), writeInt(), writeFloat()…

```
File f = new File("nombredelfichero");
FileOutputStream fo = new FileOutputStream(f);
DataOutputStream d = new DataOutputStream(fo);
d.writeInt(1);
d.writeChar('1');
```

Task

Make a program that creates a byte stream and then converts it to a character stream.

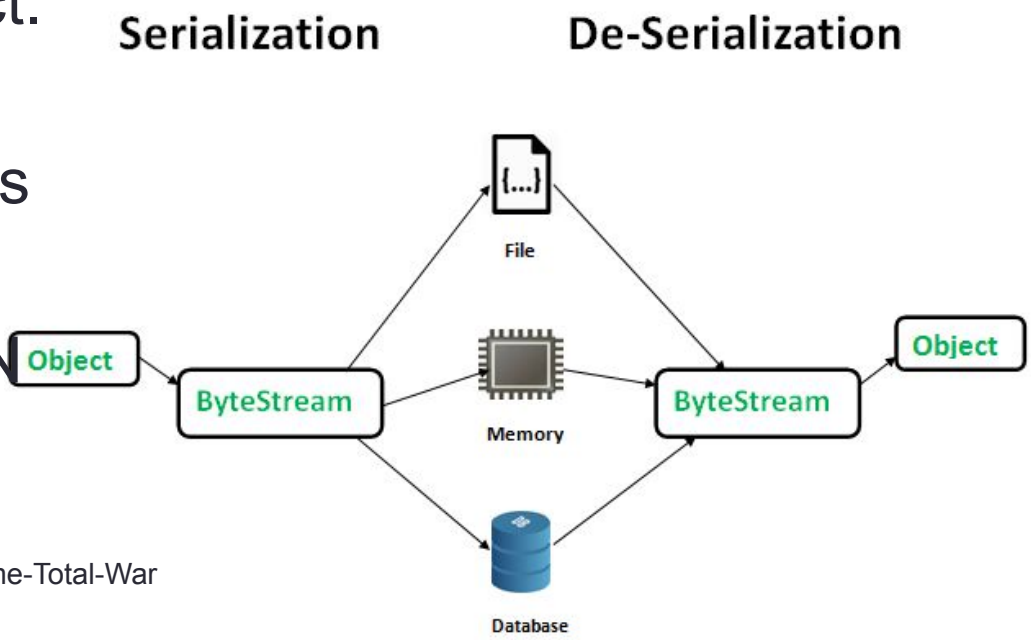
ACT 1.1 - DICTIONARY

Remember the use of dynamic data structures

Serialization

- **What is serialization?**
- Serialization is the process of converting the state of an object to a format that can be stored or transported (a sequence of bytes).
- The opposite is deserialization, which turns an information sequence into an object.

- **Binary**: object ***TO*** bytes
- **XML**: object ***TO*** XML
- **JSON**: object **TO** JSON



https://www.wikihow.com/Unlock-Factions-in-Rome-Total-War

Serialization

- **To serialize an object**
  - Its class must implement **Serializable interface** (java.io)
    - Empty interface
  - If it contains objects, their classes should also implement Serializable

```
class A implements Serializable{
    Int num;
    String str;
    // B also implements Serializable interface.
    B obj = new B();
}
```

Binary Serialization

- **FileOutputStream + ObjectOutputStream**

```
FileOutputStream f = new FileOutputStream('file.bin');
ObjectOutputStream output = new ObjectOutputStream(f);

// Method for serialization of object
output.writeObject(object);

output.close();
f.close();
```

Binary De-serialization

- **FileInputStream + ObjectInputStream**

```
FileInputStream f = new FileInputStream('file.bin');
ObjectInputStream input = new ObjectInputStream(f);

// Method for deserialization of object
Demo object1 = (Demo) input.readObject();

input.close();
f.close();
```

Binary De-serialization

- **FileInputStream + ObjectInputStream**

```
FileInputStream f = new FileInputStream('file.bin');
ObjectInputStream input = new ObjectInputStream(f);

// Method for deserialization of object
Demo object1 = (Demo) input.readObject();

input.close();
f.close();
```

Binary De-serialization

- **FileInputStream + ObjectInputStream exceptions**

`FileInputStream` exceptions:

- **`FileNotFoundException`:** This exception is thrown if the specified file does not exist or cannot be opened for reading.
- **`IOException`:** This is a general exception that can be thrown for various I/O errors, such as file access denied, disk full, or read errors.

`ObjectInputStream` exceptions:

- **`EOFException`:** This exception is thrown if the end of the file is reached before the expected object has been read.
- **`IOException`:** This exception can also be thrown for various I/O errors, such as file access denied, disk full, or read errors.
- **`StreamCorruptedException`:** This exception is thrown if the stream is corrupted or if there is a mismatch between the serialized data and the expected object type.
- **`ClassNotFoundException`:** This exception is thrown if the class of the deserialized object cannot be found in the classpath.

XML Serialization

- Object saved as a label with the name of the object's class
- Attributes converted to an internal tag
  - Name of the attribute → name of the tag
  - Value of the attribute → text in the label
- If the class contains a list
  - A label will also be created with the name of the list

XML Serialization

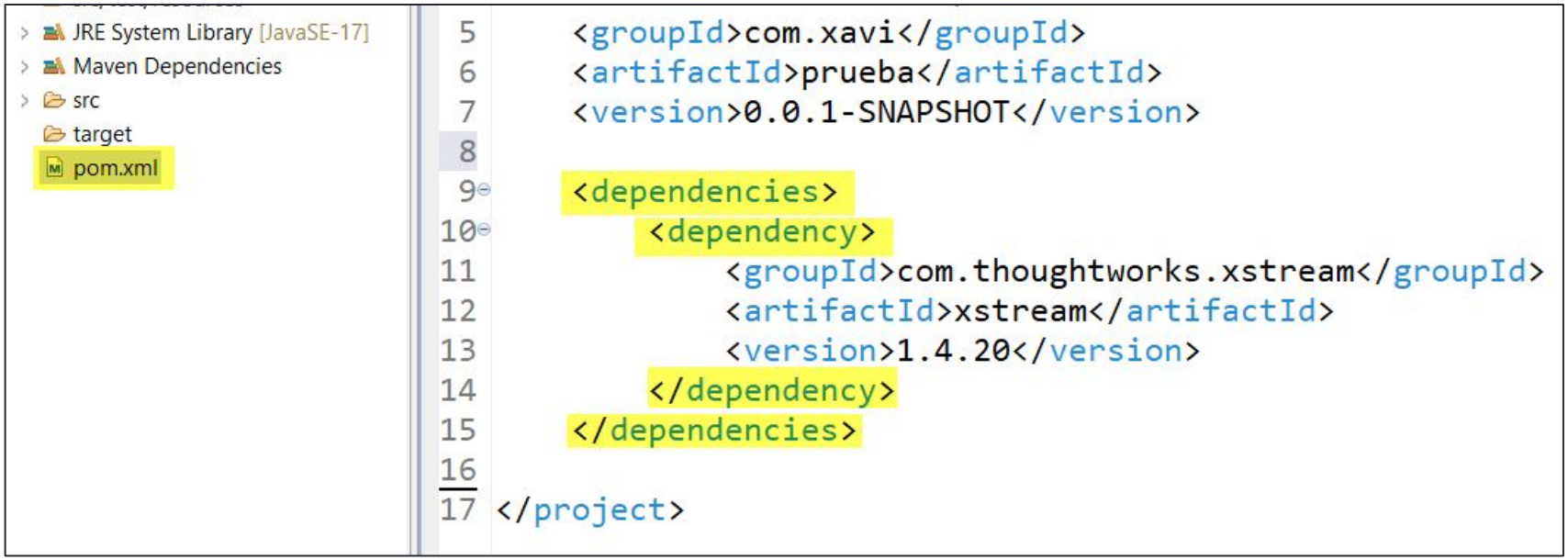
- For example:

```
public class Person implements Serializable {
   private String nombre;
   private int edad;
   .....
   }

public class PersonList implements Serializable, List{
   private List<Persona> lista = new ArrayList<Persona>();
   .....
   }
```

Adding dependency in Maven

- Go to *pom.xml* and add tag <*dependencies*> and, inside, copy the [dependency]



XML Serialization

- Create Maven project:
  - Add dependency
- Import:

```
import com.thoughtworks.xstream.XStream;
```

- Create XStream and allow types:
  - Used classes must be allowed to prevent attacks to your deserialization entry point

```
// Create XStream and allow types to use
XStream flujox = new XStream();
flujox.allowTypes(new Class[] {Person.class, PersonList.class});
```

XML Serialization

- Serialize and deserialize

```
// Create XStream
XStream flujox = new XStream();

//Fill list
…

//Serialize to XML
flujox.toXML(lista, new FileOutputStream("PersonsInfo.xml"));

//Allow types to use, before deserialization
flujox.allowTypes(new Class[]{Person.class, PersonList.class});

//DeSerialize from XML
PersonList newlista =
      (PersonList) flujox.fromXML( new File("PersonsInfo.xml");
```

XML Serialization

```
<ficheros.xml.PersonList>
   <lista>
      <ficheros.xml.Person>
         <nombre>Ana</nombre>
         <edad>14</edad>
      </ficheros.xml.Person>
      <ficheros.xml.Person>
         <nombre>Luis</nombre>
         <edad>15</edad>
      </ficheros.xml.Person>
   </lista>
</ficheros.xml.PersonList>
```

ACT 1.2 - AGENDA

Use XML serialization

JSON - JavaScript Object Notation

- JSON is enclosed by **{...}**
  - Each field is *key_name : value*
  - Keys are Strings:
    - Cannot be null
    - Case-sensitive
    - Cannot be duplicated
  - Values can be of types
    - null
    - Number
    - String
    - Boolean: true or false
    - Array : enclosed by **[...]**
    - Object: enclosed by **{...}** . Objects can contain other objects

```
{
     "name" : "Pepe",
     "surname" : "García García",
     "age" : 30,
     "married" : false,
     "address" : {
          "street" : "Colon",
          "zipcode" : "46001"
          },
     "sports" : ["hiking", "mtb"]
}
```

ACT 1.3 - AGENDA

Modify Agenda to use JSON serialization
(Jackson library)

References

- Based on the work of Cristina Ausina
- Pictures from flaticon.com
- Some modifications added by Héctor Llorens

- https://x-stream.github.io/
- https://github.com/FasterXML/jackson