File management

Object serialization


Serialization is the process of converting an object into a sequence of bytes, so that we can easily store it or send it through some communication channel or stream. In this document we are going to learn how to serialize objects into binary files. This way, we can easily store them from our applications, and retrieve them whenever we want.

1. Object serialization in Java

In order to serialize the objects of a class, we firstly need to identify this class as *serializable*. This can be easily done by just implementing `Serializable` interface. This interface has no methods, so it just help us identify the affected class as serializable:

```
class Person implements Serializable
{
    ...
}
```

In order for a class to be serializable, every internal attribute must also be serializable (or a simple data type). If there's any attribute that we don't want to serialize, we need to tag it as `transient`. In the following example, whenever we serialize `Person` objects, we are going to store person's name, but not his/her age:

```
class Person implements Serializable
{
    private String name;
    private transient int age;

    ...
}
```

2. Dealing with serialized objects

If we want to manage serialized objects in Java, we need to make use of two special classes:

- `ObjectInputStream` class lets us read serialized objects from a file or input channel, through the `readObject` method

- `ObjectOutputStream` class lets us write serialized objects to a file or output channel, through the `writeObject` method

This is an example of how we can write serialized objects to a file:

```
try(ObjectOutputStream oos =
    new ObjectOutputStream(new FileOutputStream("people.dat")))
{
    oos.writeObject(new Person("John", 49));
    oos.writeObject(new Person("Susan", 45));
}
catch (IOException e)
{
    System.err.println("Error storing people");
}
```

This example reads information previously serialized:

```
try(ObjectInputStream ois =
    new ObjectInputStream(new FileInputStream("people.dat")))
{
    while(true)
    {
        Person p = (Person)(ois.readObject());
        System.out.println(p);
    }
}
catch (Exception e)
{
    System.err.println("Error storing people");
}
```

Regarding the object reader, there's no way to detect the end of file. Instead of this, an `EOFException` (*End-Of-File Exception*) is produced. That's why, if we want to read until the end of file, we actually implement and endless loop and wait until the exception is thrown.

2.1. Serializing collections

We can also serialize arrays or collections of data, as long as the objects contained in this collection are also serializable:

```
List<Person> people = new ArrayList<>();
people.add(new Person(...));
people.add(new Person(...));
oos.writeObject(people);
...
List<Person> people2 = (List<Person>)(ois.readObject());
```

2.2. Associations and serialization

If we have an association between two classes whose objects are being serialized, we don't need to care about the association when we retrieve the data from the file. For instance, if we store a couple of books sharing the same author, this author will still be shared between these books when we read them from the file.

> **Exercise 1:**
>
> Create a project called **TaskSerializer**, with the following elements:
>
> - A class called `Task` to represent tasks to be done. For each task we are going to store its description and the date to be finished (a string with the format *dd/mm/yyyy*).
> - The main program will try to read a list of tasks stored in a file called *tasks.dat* at the beginning, and print them in the screen. Then, it will ask the user if he/she wants to add a new task to the list. At the end, the new list will be written in the file before closing the program.