UFAZ / Strasbourg University

Object Oriented Programming

Year 1 – Common curriculum

Tutorial / Lab session #6: Java I/O, streams, and serialization

Exercise 1 – Importing CSV files

The CSV (comma separated values) is a very common way to store data in a file. Being able to read such files using Java will allow us to perform some operations on those data: for instance, analyze data with machine learning algorithms.

A CSV file contains a list of entries (one entry = one line) and each entry is made of a list of values separated by a comma.

1. What Java collection would you use to store an entry? A list of entries?
2. Create a class `CSVReader`. This class has 2 member variables: a file name and a list of entries. The constructor of your class should take a filename as argument.
3. The class String offers a method to split a string (see Fig. 1). How will this method come handy for the task at hand?
4. Write a method `void importData()`. You will access the file by using a Reader and you will do buffered reading. Don't forget to close your stream(s).
5. Write the method `int getNumberOfEntries()` that returns the total number of entries imported from the file.
6. Write the method `ArrayList<String> getEntry(int i)` that returns the entry at index i. The class `ArrayList<>` offers the method `get(index)` to retrieve the element at index i.

Exercise 2 – Introduction to serialization

Serialization is a mechanism provided by Java that allows one to represent an object as a sequence of bytes and that includes information about the type of the object and the types of the data contained in the object. Once serialized, the object can be recreated in memory through deserialization (in other word, we can directly affect an object after deserialization; no further operation on the original object must be performed – such as extracting member variables, etc.).
In order to serialize/deserialize object, we are going to use the classes `ObjectOutputStream` (and the method `writeObject()`) and `ObjectInputStream` (and the method `readObject()`).

One must make their object serializable (i.e. they must implement the interface Serializable) in order to use this mechanism. Lucky for us, most classes of the Java API are serializable and we will be able to use this mechanism without much effort. Making our own objects serializable is (for now) out of the scope of this tutorial.

1. Create a class `SaveRestoreObjFromFile`. This class doesn't have any member variables.
2. Write a static method `void saveToFile(String outputFile, Object object)` that write the object given as parameter in the file. (nb : by convention, serialized object in a file are stored in a `.ser` file).
3. Write a static method `Object restoreFromFile(String inputFile)` that returns the object read from the file.
4. In your main, try out your methods by exporting the entries imported from the CSV (exercise 1) and importing them into an object of matching type.

`public String[] split(String regex)`

Splits this string around matches of the given regular expression.

This method works as if by invoking the two-argument `split` method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string `"boo:and:foo"`, for example, yields the following results with these expressions:

Regex	Result
`:`	`{ "boo", "and", "foo" }`
`o`	`{ "b", "", ":and:f" }`

Figure 1 - String split using a separator

```
public boolean addAll(int index,
            Collection<? extends E> c)
```

Inserts all of the elements in the specified collection into this list, starting at the specified position. Shifts the element currently at that position (if any) and any subsequent elements to the right (increases their indices). The new elements will appear in the list in the order that they are returned by the specified collection's iterator.

Specified by:

`addAll` in interface `List<E>`

Overrides:

`addAll` in class `AbstractList<E>`

Parameters:

`index` - index at which to insert the first element from the specified collection

`c` - collection containing elements to be added to this list

Returns:

`true` if this list changed as a result of the call

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

`NullPointerException` - if the specified collection is null

Figure 2 - Adding a collection in an ArrayList<>

`public E get(int index)`

Returns the element at the specified position in this list.

Specified by:

`get` in interface `List<E>`

Specified by:

`get` in class `AbstractList<E>`

Parameters:

`index` - index of the element to return

Returns:

the element at the specified position in this list

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

Figure 3 - Retrieving an entry from an ArrayList<>