Process and Service Programming

2.1. Running processes in Java with Runtime

I.E.S. Doctor Balmis

2.1. Running processes in Java with Runtime

2.1.1. Quick process launch

There are several methods defined in the Runtime class. These methods can be invoked to get the information about the runtime environment such as number of processors available to the JVM, about of memory available, loading native library, explicitly call garbage collector, and so forth.

> Specification java.lang.Runtime

Every Java program has an instance of the Runtime class, which encapsulates the runtime environment of the program. This class cannot be instantiated, but we can get a reference **singleton instance** to the Runtime of the currently running program with the help of the static method **java.lang.Runtime.getRuntime()**.

> **Design patterns: Singleton**
>
> ¿What are design patterns? ¿What is and what is used for the singleton pattern?
>
> Look how to implement a class with the singleton pattern.
>
> Refactoring.Guru design patterns

The Runtime class method we are interested in, to create a new processes is

> public Process exec(String command) throws IOException

This is a simple, not yet customizable, way to spawn a new sub-process.

```java
public static void main(String[] args) throws IOException {
    // Launch notepad app
    Runtime.getRuntime().exec("notepad.exe");

    // This way always works
    // String separator = System.getProperty("file.separator");
    // Runtime.getRuntime()
    //     .exec("c:"+separator+"windows"+separator+"notepad.exe");

    // This way used to work (UNIX style paths)
    // Runtime.getRuntime().exec("c:/windows/notepad.exe");
}
```

As you can see the argument to `exec` method is just the program we want to run. In this example, as *notepad* is in the system PATH it's not necessary to tell the path to the program. Otherwise, the path must be specified with the program name.

2.1.2 System properties and command shells

If we plan to code platform independent applications, we have to deal with many issues because of differences between OS. So sometimes we need to deal with specific OS information. A useful way to get that information is by getting System properties.

Specification System.getProperties

Some examples are provided here using System properties. Similar solutions can be used for other issues.

File separator

For file path or directory separator, the Unix system introduced the slash character / as directory separator, and the Microsoft Windows introduced backslash character \ as the directory separator. In a nutshell, this is / on UNIX and \ on Windows.

Then, ¿how can we code OS independent applications??

In Java, we can use the following three methods to get the platform-independent file path separator.

- System.getProperty("file.separator")
- FileSystems.getDefault().getSeparator() (Java NIO)
- File.separator Java IO

From now on, we are gonna use System properties in our applications for several situations using `System.getProperty(String propName)`. These properties are configured by the OS and the JVM, though we can modify them by setting the JVM running setting

> String separator = System.getProperty("file.separator");

or

> -Dfile.separator

Nevertheless is always a good practice to use slash character **/** in paths as Java is able to convert them to the system it is running on.

If we want to run an OS command we have to do it as we usually do, by using the command shell, where once again we find the troubleshot with UNIX / Windows.

Let's take a look at the way we can use the system properties, once again, to get a list of files in the user personal folder.

```java
// First we get the user folder path
String homeDirectory = System.getProperty("user.home");
// And then we set which OS are we running on
boolean isWindows = System.getProperty("os.name")
  .toLowerCase().startsWith("windows");

if (isWindows) {
    Runtime.getRuntime()
      .exec(String.format("cmd.exe /c dir %s", homeDirectory));
} else {
    Runtime.getRuntime()
      .exec(String.format("sh -c ls %s", homeDirectory));
}
```

non-interactive shell mode

In the previous code example, both for Windows and UNIX modifier **c** is used for command shells. This modifier tells the system to open a command shell, to run the companion command and close the shell after it has finished.

```java
// Calling app example
public void mouseClicked(MouseEvent e) {
  // Launch Page
  try {
    // Linux version
    Runtime.getRuntime().exec("open http://localhost:8153/go");
    // Windows version
    // Runtime.getRuntime().exec("explorer http://localhost:8153/go");
  } catch (IOException e1) {
    // Don't care
  }
}
```

System properties

> Our first applications in java is not gonna be an easy one.
>
> Using methods from System class and Runtime class, write the code for an app that shows
>
> - all the system properties configured in your OS
> - total memory, free memory, used memory and processors available
>
> Make a research into Runtime class methods. For System properties try to get a list or iterable data estructure to show each of the system properties and their values.

Proposed solution to previous activiy

```java
long freeMemory = Runtime.getRuntime().freeMemory();
long availableMemory = Runtime.getRuntime().totalMemory();
long usedMemory = availableMemory - freeMemory;

/*** Runtime.getRuntime() usage ***/
// Show system information
// Memory will be shown in MBytes formatted with 2-decimal places
DecimalFormat megabytes = new DecimalFormat("#.00");
System.out.println("Available memory in JVM(Mbytes): " +
        megabytes.format((double)availableMemory/(1024*1024)));
System.out.println("Free memory in JVM(Mbytes): " +
        megabytes.format((double)freeMemory/(1024*1024)));
System.out.println("Used memory in JVM(Mbytes): " +
        megabytes.format((double)usedMemory/(1024*1024)));

System.out.println ("Processors in the system: "
        + Runtime.getRuntime().availableProcessors());

/*** System.getProperties() usage ***/
// Show each pair of property:value from System properties

// 1st. As a lambda expression using anonymous classes
System.getProperties().forEach((k,v) -> System.out.println(k + " => " + v));

// 2nd. As a Map.entrySet
for (Map.Entry<Object, Object> entry : System.getProperties().entrySet()) {
    Object key = entry.getKey();
    Object val = entry.getValue();
    System.out.println("> " + key + " => " + val);
}

```

```java
// 3rd. As a Map.keySet
for (Object key : System.getProperties().keySet().toArray())
{
    System.out.println(">> " + key+":"+System.getProperty(key.toString()));
}

// Other methods found by students, based on a Properties object methods.
Properties prop = System.getProperties();
for (String propName: prop.stringPropertyNames()) {
  System.out.println(propName +  ":" + System.getProperty(propName));
}

// Or directly to the console using
prop.list(System.out);
```

Number format

In any programming language we have many different ways to format the information shown to the user. As in this first applications we are using the console as the system output, let's check the two main techniques we can use in Java

- NumberFormat

Using NumberFormat class or any of its descendants we can get control on how the numbers are shown with high precision, using numeric patterns.

```java
DecimalFormat numberFormat = new DecimalFormat("#.00");
// Hashes can be used instead of zeros to allow .30 to be shown as 0.3
// (additional digits are optional)
System.out.println(numberFormat.format(number));
```

- System.out.printf

Similar to C's printf syntax, we can use the java.util.Formatter syntax to set how data is visualized.

```java
System.out.printf("\n$%10.2f",shippingCost);
// numbers after % print preceding spaces to fill
// and justify numbers.
System.out.printf("%n$%.2f",shippingCost);
```