



Process and Service Programming

2.4 Annex I - System properties and console programming



I.E.S.
Doctor Balmis

PSP class notes (https://psp2dam.github.io/psp_sources) by Vicente Martínez is licensed under
CC BY-NC-SA 4.0  (<http://creativecommons.org/licenses/by-nc-sa/4.0/?ref=chooser-v1>)

2.4 Annex I - System properties and console programming

- I.1 System properties and command shells
- I.2 Console I/O format
 - Console output
 - Console input
 - String format
 - Number format
 - Colours in console applications

I.1 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`

([https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#getProperties\(\)](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#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 troubleshoot 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.

```

1 // First we get the user folder path
2 String homeDirectory = System.getProperty("user.home");
3 // And then we set which OS are we running on
4 boolean isWindows = System.getProperty("os.name")
5     .toLowerCase().startsWith("windows");
6
7 if (isWindows) {
8     Runtime.getRuntime()
9         .exec(String.format("cmd.exe /c dir %s", homeDirectory));
10 } else {
11     Runtime.getRuntime()
12         .exec(String.format("sh -c ls %s", homeDirectory));
13 }

```

java

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.

Next you can look at a handler event manager for a mouse clic, into a graphic application, to open a web site in a browser. The code shows how to do it in *X like operating system and one way to do it in Windows systems is commented.

```

1 // Calling app example
2 public void mouseClicked(MouseEvent e) {
3     // Launch Page
4     try {
5         // Linux version
6         Runtime.getRuntime().exec("open http://localhost:8153/go");
7         // Windows version
8         // Runtime.getRuntime().exec("explorer http://localhost:8153/go");
9     } catch (IOException e1) {
10         // Don't care
11     }
12 }

```

java

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 estruture to show each of the system properties and their values.

I.2 Console I/O format

Character codification

One aspect to take into account when working with streams is the encoding of the information exchanged between processes, which depends on the operating system we are working on. Most systems (GNU/Linux, Mac OS, Android, iOS...) use UTF-8 encoding, based on the Unicode standard.

For its part, MS Windows uses its own formats, incompatible with the rest, such as Windows-1252. So to correctly handle data in Java when using more advanced inter-process communication mechanisms, it will be necessary to take into account the type of encoding that the system itself uses.

```
1 // Getting the default encoding
2 System.out.println(System.getProperty("file.encoding"));
3 // Setting the encoding
4 System.setProperty("file.encoding", "UTF-8");
5
6 // Reading with a specific encoding
7 BufferedReader reader = new BufferedReader(new InputStreamReader(System.in, "UTF-8"));
8 // Writing with a specific encoding
9 PrintWriter writer = new PrintWriter(new OutputStreamWriter(System.out, "UTF-8"));
10
11 // Stdin with a specific encoding
12 Scanner scanner = new Scanner(System.in, "UTF-8");
13 // Stdout with a specific encoding
14 System.out.println(new String("Hello, World!".getBytes("UTF-8")));
```

java

Console output

In Java, we can use the `System.out` object to print to the console. We can use the `println` method to print a line to the console.

```
System.out.println("Hello, World!");
```

java

Console input

In Java, we can use the `System.in` object to read from the console. We can use the `Scanner` class to read from the console.

```
Scanner scanner = new Scanner(System.in);
String name = scanner.nextLine();
```

java

String format

In Java, we can use the `String` class to format the output. We can use the `format` method to format the output. This method is similar to the `printf` method in C.

```
String.format("The value of PI is %.2f", Math.PI);
```

java

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** (<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/DecimalFormat.html>)

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

```
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));
```

java

- **System.out.printf** (<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Formatter.html>)

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

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

java

Colours in console applications

There is a way to print in different colours when using the console. Here you have got an example code with some colours and the way to use it.

```
public class UsingColoursInConsole {  
  
    public static final String ANSI_RESET = "\u001B[0m";  
    public static final String ANSI_BLACK = "\u001B[30m";  
    public static final String ANSI_RED = "\u001B[31m";  
    public static final String ANSI_GREEN = "\u001B[32m";  
    public static final String ANSI_YELLOW = "\u001B[33m";  
    public static final String ANSI_BLUE = "\u001B[34m";  
    public static final String ANSI_PURPLE = "\u001B[35m";  
    public static final String ANSI_CYAN = "\u001B[36m";  
    public static final String ANSI_WHITE = "\u001B[37m";  
  
    public static final String ANSI_BLACK_BACKGROUND = "\u001B[40m";  
    public static final String ANSI_RED_BACKGROUND = "\u001B[41m";  
    public static final String ANSI_GREEN_BACKGROUND = "\u001B[42m";  
    public static final String ANSI_YELLOW_BACKGROUND = "\u001B[43m";  
    public static final String ANSI_BLUE_BACKGROUND = "\u001B[44m";  
    public static final String ANSI_PURPLE_BACKGROUND = "\u001B[45m";  
    public static final String ANSI_CYAN_BACKGROUND = "\u001B[46m";  
    public static final String ANSI_WHITE_BACKGROUND = "\u001B[47m";  
  
    public static void main(String[] args) {  
        System.out.println(ANSI_GREEN + ANSI_WHITE_BACKGROUND + "Hello"  
            + ANSI_BLUE + ANSI_YELLOW_BACKGROUND + " Bye bye" + ANSI_RESET);  
    }  
}
```

java

