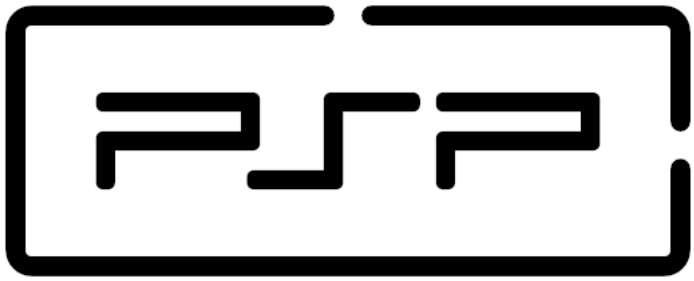

Programación de Servicios y Procesos

2.1. Creación de procesos con Java con Runtime

2.1. Creación rápida de procesos con Java con Runtime

2.1.1. Creación rápida de procesos

La clase java.lang.Runtime se usa principalmente para interactuar con el JRE de Java. Esta clase proporciona métodos para lanzar procesos, llamar al recolector de basura (Garbage Collector), saber la cantidad de memoria disponible y libre, etc.

> Especificación java.lang.Runtime

Cada aplicación en Java tiene acceso a una única instancia de *java.lang.Runtime* a través del método `Runtime.getRuntime()` que devuelve la instancia **singleton** de la clase Runtime.

> **Patrones de diseño: Singleton**
>
> ¿Qué son los patrones de diseño? ¿Qué es y para qué se usa el patrón de diseño singleton?
>
> Investiga cómo realizar una clase que siga el patrón de diseño singleton.
>
> Refactoring.Guru Patrones de diseño

El método que nos interesa a nosotros para la creación de procesos es

> public Process exec(String command) throws IOException

Veamos un ejemplo sencillo de uso de este método

```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");
}
```

Se puede observar que en el parámetro que pasamos al método `exec` indicamos el programa que queremos ejecutar. En este caso, como el *notepad* se encuentra en el PATH del sistema, no es necesario indicar la ruta donde se encuentra el programa. En otro caso, sí tendríamos que hacerlo.

2.1.2 Propiedades del sistema y comandos del sistema

Si tenemos pensado desarrollar aplicaciones que funcionen en diferentes SO tendremos que enfrentarnos a la problemática del funcionamiento diferente de los distintos SO.

Vamos a ver algunos ejemplos que pueden servir como guía para otros problemas similares a los expuestos.

File separator

Para indicar las rutas en un sistema los sistemas UNIX emplean el caracter **/** como separador mientras que los sistemas Windows usan el caracter **** . En resumen, / en *X y \ en Windows.

¿Cómo podemos hacer entonces que nuestras aplicaciones sean independientes del SO en el que se ejecutan?

Para este tipo de cuestiones vamos a utilizar de forma recurrente las propiedades del sistema mediante `System.getProperty(String propName)` . Estas propiedades se configuran con el propio sistema operativo, aunque las podemos modificar usando los parámetros de ejecución de la máquina virtual

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

o

> -Dfile.separator

Aunque siempre es una buena práctica usar el caracter **/** en las rutas ya que Java es capaz de convertirlas al sistema en el que se ejecuta.

Si lo que queremos es ejecutar un comando del SO, tenemos que hacerlo, al igual que si lo hacemos manualmente, a través del shell del sistema, donde volvemos a encontrar la dicotomía entre sistemas UNIX y sistemas Windows.

Vamos a ver el código que, a través de las propiedades del sistema, nos permite obtener un listado de los archivos existentes en la carpeta personal del usuario.

```java
// Primero obtenemos la carpeta del usuario
String homeDirectory = System.getProperty("user.home");
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));
}
```

Modo shell no interactivo

Como se puede observar, tanto para Windows como UNIX se ha usado el modificador **c** del comando. Este modificador indica que se abra un shell, se ejecute el comando recibido y se cierre el proceso del shell.

A continuación podemos ver un ejemplo de respuesta ante la pulsación de un botón, en una app gráfica, paraabrir una página en el navegador. Tenemos cómo se haría en sistemas *X y comentado una de las formas de hacerlo en Windows.

```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

Vamos a crear nuestro primer programa en Java, que no va a ser tan sencillo como pueda parecer

Usando métodos de las clases System y Runtime hacer un programa que muestre

- todas las propiedades establecidas en el sistema operativo y sus valores.
- memoria total, memoria libre, memoria en uso y los procesadores disponibles

Mira los métodos que proporcionan las clases Runtime y system. Intenta obtener una lista u otra estructura de datos que te permita recorrer las propiedades para ir mostrando sus nombres y valores.

Formato numérico

Todos los lenguajes de programación tienen varias formas de mostrar la información al usuario. Cuando se trata de mostrar información a a través de la consola, tenemos un par de alternativas para formatear la información numérica.

- NumberFormat

Si usamos la clase NumberFormat o cualquiera de sus descendientes podemos controlar con bastante precisión cómo se verán los números, usando patrones.

```java
DecimalFormat numberFormat = new DecimalFormat("#.00");
// Si usamos hashes en vez de ceros permitimos que .30 se vea como 0.3
// (los dígitos adicionales son opcionales)
System.out.println(numberFormat.format(number));
```

- System.out.printf

Heredado de la sintaxis de la función printf de C, podemos utilizar la sintaxis de java.util.Formatter para configurar cómo será visualizada la información.

```java
System.out.printf("\n$%10.2f",shippingCost);
// % rellena con hasta 10 posiciones los números
// para justificarlos a la derchan.
System.out.printf("%n$%.2f",shippingCost);
```

Usando colores en la salida por consola

Hay una forma sencilla de mostrar información por consola usando diferentes colores. Os dejo un ejemplo de código con la definición de algunos colores y la forma de usarlos.

```java
public class UsarColoresEnConsola {

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 + "Hola" + ANSI_BLUE + ANSI_YELLOW_BACKGROUND + " Adiós"
    }
}
```