DT271C - Seminar 1 - Processes and threads

Introduction

The purpose of this seminar is to study creation of processes and threads in Java. Furthermore, you should understand how the command shells work.

It is suggested that you work under Windows, but you could also do the tasks in Linux or any other OS, but then you need to change command names in some cases.

Read the complete document before you start with the specific tasks.

Good luck!

1 Preparations

Before starting with the programming, you are expected to do the following tasks:

- Read about the concepts of command line interface (CLI) or command shell. Use the following links

- https://www.makeuseof.com/tag/a-beginners-guide-to-the-windows-command-line/
- https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

- Learn few commands such as ls, date, time, ps, pwd, cat, dir, type, whoami, ping, cd etc. Study their functions (purposes) and their different usage forms (options or switches). You should identify which ones are Unix/Linux and which ones are Windows commands.

- Study the ProcessBuilder class in Java documentation.

- https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html

- Read about Windows task manager

- https://www.howtogeek.com/405806/windows-task-manager-the-complete-guide/

- Download the sample code (TestProcessBuilder.java) from canvas and make sure that you understand the code. Run it from the command line and not the IDE. Try to start for example notepad from within the Java command shell. You could also try to supply a path for a file when starting notepad.

```
***** Welcome to the Java Command Shell *****
If you want to exit the shell, type END and press RETURN.

jsh>notepad test.txt
```

Figure 1: Java command shell.

- Study the Java Thread class and how to create threads.

- https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

2 Programming tasks

Using the TestProcessBulder.java class as a base, implement task 1, task 2 and task 3 according to the descriptions below.

2.1 Task 1

In order to not block the command shell when a process is created, we will create a thread that manages the running of each application. Create a new class that implements the Runnable interface and which takes the string command as input in the constructor. This will be the class that calls ProcessBuilder in its run method.

Observe that since we need to have access to the string command in the run method of the thread, we need to store this when the thread is created. In the run method we need to add the code from the method createProcess, then we need to modify the old method createProcess as follows: Instead of calling ProcessBuilder in this method you should create an instance of the class that implements Runnable. Then start this instance as a thread.

2.2 Task 2

Users may sometime enter a wrong or an invalid command. You may have noticed the error message you get when you enter an invalid command. In addition to displaying the error message, it is desired that a log of the error commands is kept in memory for system troubleshooting purposes. When the user enters the command showerrlog, the shell should display all erroneous commands entered by the user during the current session. The task is to modify the shell program to keep a log of erroneous commands and show it to the user when the showerrlog command is invoked. Remember that showerrlog is not an OS command. It is an enhancement you have just added! Hint! How can you detect that a command has been entered that cannot be run?

2.3 Task 3

In this task you will create a simple stopwatch that is console based, running in a separate thread. Start with creating a new Java class that has a main method.

In the main method create a new thread and start it. The thread should do the following.

- When starting the thread, the elapsed time should be set to zero.
- Every 10ms, there should be a printout of the elapsed time in seconds. Use Thread.sleep()
- When the elapsed time has reached 60 seconds, then the stopwatch thread should exit and join with the main thread. Use Thread.join().
- Answer the following questions:
    1. What does the Thread.join() do? What happens if we remove it from the main method?
    2. Compare the real elapsed time using a stopwatch on your phone with your Java implementation. Does the elapsed time differ, why?

Figure 2: Output of stopwatch.

3 Final observations

Before you present your seminar, make sure that you have the answer to the questions given in the preparations. Also write down the following for each task to discuss during the seminar.

- How did you implement the task?
- Why did you solve it in the way you did it?
- What difference in behaviour did you notice?