COS 226 Practical Assignment 6

- Date Issued: **13:30 Monday 17th October 2016**
- Date Due: **17:00 Friday 28th October 2016**
- Submission Procedure: **Upload via the CS Website**
- Submission Format: **Compressed source files, as described below.**
- This assignment consists of **2 tasks** for a total of **20 marks**.

1 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be able to give you the solutions. They will be able to help you with coding, debugging and understanding the concepts explained both in the textbook and during lectures.

2 Submission Instructions

You will produce one source file for each task. The module website provides one upload slot for each task. Submit the source file for each task via the corresponding upload slot. You must archive your source files. **Do not** upload machine executable binaries. Compress and upload **only** the specified plain text source file for each task. The assignment submission system will close the normal upload slots automatically on the deadline.

3 Mark Allocation

Each assignment is divided into some number of small, independent tasks. Each task will be allocated some marks. Therefore:

1. your source file must compile with the provided framework
2. your program must produce the expected output
3. your program must not throw any exceptions
4. your program must terminate gracefully (i.e. return from the main method without errors)

3.1 Source Code

Before you begin, download the source code for this assignment from the module website.

4 Introduction

This assignment presents a small problem that can be solved using locks and conditions. You are provided with a class, *SavingsAccount*, that represents an account at a bank. *SavingsAccount* extends *Account*, which records the available balance and declares a *deposit* and a *withdraw* operation to modify the balance. Internally, the *deposit* and *withdraw* methods must make use of the *changeAmount* method to alter the balance of the account.

There is no overdraft facility available for the account so a withdrawal cannot take place if the requested amount is greater than the amount in the account. Deposits and withdrawals can take place concurrently.

SavingsAccount is tested by means of the *Executor* class. The executor does not take any arguments and, instead, makes use of two, separate test functions to ensure that the savings account class works correctly.

Study *Executor*, *TestThread*, and *Account* to get an idea of how the system works.

Task 1 ..... (10 marks)

Complete the *deposit* and *withdraw* methods of *SavingsAccount* so that concurrent deposits and withdrawals are executed correctly and so that when a withdrawal requests more money than what is available, the call to *withdraw* will block until the balance is sufficient to cover the withdrawal.

Complete this task in the single SavingsAccount.java file and compress and submit this file as the answer to this assignment to the upload slot Practical6 task1.

Task 2 ..... (10 marks)

Assume that there are two types of withdrawals: an **ordinary** withdrawal and a **preferred** withdrawal. Preferred withdrawals get preference over ordinary withdrawals, which means that if there are pending ordinary wthdrawals, a subsequent preferred withdrawal request will be completed before any ordinary withdrawal when sufficient money is available in the account.

Complete your implementation of *SavingsAccount* so that preferred withdrawals are allowed to pre-empt ordinary withdrawals.

Complete this task in the single SavingsAccount.java file and compress and submit this file as the answer to this assignment to the upload slot Practical6 task2.