



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

**Parallel Programming**  
**Assignment 7: Synchronization And Resource Sharing**  
**Spring Semester 2017**

Assigned on: **04.04.2016**

Due by: **10.04.2016**

## Overview

In this exercise, we look at one of the major problems that arise in writing parallel programs: Accessing state that is shared between multiple threads.

## Getting Prepared

- Download the ZIP file named `assignment7.zip` on the course website.
- Import the project in Eclipse: Click on *File* in the top-menu, then select *Import*. In the dialog, select *Existing Projects into Workspace* under the *General* directory, then click on *Next*. In the new dialog, select the radiobox in front of *Select archive file* to import a ZIP file. Then, click *Browse* on the right side of the text-box to select the ZIP file you just downloaded from the website (`assignment7.zip`). After that, you should see `assignment7` as a project under *Projects*. Click *Finish*.
- If you have done everything correctly, you should now have a project named `assignment7` in your *Package Explorer*.

## Banking System

The CEO of the bank you are working for has recently bought some expensive multi-core machines. The vendor ensured to the CEO that their four core machines will quadruple the number of transactions per second the bank can handle. After they installed the new machines, the CTO realized that their code base was not able to exploit the parallelism offered by their new servers. In order to keep his job, he called you for help.

The code skeleton you downloaded and imported contains the relevant parts of the KBS code base. There are two classes, `Account` and `BankingSystem`. `Account` represents the bank account of a client, while `BankingSystem` provides the essential functionality of the bank such as `transferMoney`, to transfer money between two accounts, and `totalMoneyInBank`, used by them to figure out how much money they currently maintain.

In the following tasks, we will first try to make `Account` and `BankingSystem` thread-safe, after that we will continue to increase the transaction performance of the system.

**Task 1 - Problem identification:** When the CTO installed the codebase on his new machines, he wanted to test the performance of the system. He wrote two simple test (see `BankingThroughputTest`) that spawns a few threads and runs a fixed amount of transactions on random accounts. On one thread, everything worked fine (`testTransactionThroughputSingle`). However as soon as he started using multiple threads (`testTransactionThroughputParallel`), sometimes the test failed by triggering the following assertion on:

```
assertThat("Did not loose any money.", bs.totalMoneyInBank(), is(sum));
```

Try to reproduce the behavior on your machine. Understand what the test does by looking at the provided classes in `src` and `test`, then explain what went wrong and why (by providing an exact set of steps that can lead to such a scenario). Write down the scenario in your report.

**Task 2 - Synchronized:** Having heard that the `synchronized` keyword is something to use in combination with threads, the CTO changed the signature of `transferMoney` in the `BankingSystem` class by adding `synchronized`:

```
public synchronized boolean transferMoney(Account from, Account to, int amount)
```

Now, the test did no longer fail, but, unfortunately, the performance reported by the parallel test-case was much worse than the sequential version:

```
Sequential using 1 Threads:
Completed 80000000 transaction in 1.920 sec: 4.1662e+07 transactions/sec
Parallel using 8 Threads:
Completed 80000000 Transaction in 13.443 sec: 5.9510e+06 transactions/sec
```

Because the CTO could not figure out what the problems with his code is, you should explain to him why the parallel version is slower than the sequential version, what the problem with the current code is and propose a way so he could fix it. Provide your answers in written form in the report.

**Task 3 - Locking:** Impressed with your explanations in the previous task, the CTO thinks you should go ahead and implement your proposed changes to the `BankingSystem` and `Account` class such that it continues to be thread-safe but the transaction performance improves considerably with more threads on a multi-core machine. You are free to change and add code in any way, but be sure to keep the interface of `BankingSystem` the same for unit testing. Also, make sure that the implementation still corresponds to the one described in the Javadoc headers.

Try to answer following questions:

**Question 1:** What if a transaction happens from and to the same account ?

```
transferMoney(a, a, X);
```

If this is a problem in your implementation, try to correct your code such that this is no longer an issue.

**Question 2:** Explain what measures you took in order to ensure that your code does not suffer from deadlocks?

**Task 4 - Summing up:** The function `sumAccounts` in the `BankingSystem` class is supposed to return the sum of money of all the accounts provided in the argument to the function at any point in time. However, at some point the CTO noticed that this function sometimes returns incorrect results, especially when a lot of transactions are happening concurrently. The CTO promises you a huge bonus if you can fix the function, such that it returns the correct amount of money at any point in time (i.e., while transactions are happening concurrently).

**Question 3:** Find out what is wrong with the current implementation by describing a scenario that can lead to an incorrect summation of the accounts.

**Question 4:** Change the implementation such that it now works for an arbitrary number of accounts.

**Question 5:** Are there ways to parallelize the summation? If so, describe how you could do it.

## Submission

In order for us to grade your exercises and give you feedback, you need to submit your code to the Subversion repository. You will find detailed instructions on how to install and set-up Eclipse for use with Subversion in Exercise 1.

Once you have completed the skeleton, commit it to SVN in a directory named `assignment7` by following the steps described below. The questions that require written answers should all be recorded in a single file named `report.pdf` and placed in the base directory of your project (i.e., in folder `assignment7`).

- **Check-in your project for the first time**

- Right click your created project called **assignment7**.
- In the menu go to **Team**, then click **Share Project**.
- In the dialog that now appears, select **SVN** as a repository type, then click **Next**.
- In case you have submitted Exercise 1, choose **Use existing repository location** and select the pre-defined URL in the dialog that should look like this  
**`https://svn.inf.ethz.ch/svn/vechev/pprog17/students/NETHZ_USERNAME`**  
Click Finish. Otherwise follow the steps in Exercise 1 to set-up a repository location.

- **Commit changes in your project**

- Now that your project is connected with the SVN server, you need to make sure that every time you change your code or your report, at the end you submit it to the SVN server as well.
- Right click your project called **assignment7**.
- In the menu go to **Team**, then click **Commit**.
- In the Comment field, enter a comment that summarizes your changes.
- Then, click on **Ok**.