THE UNIVERSITY OF SYDNEY

SOFT3410 Lab 1

Contention

The goal of this lab is to understand the problems that can arise in even simple examples of contention between threads.

Exercise 1: A bank with single customer

To begin with we will create a Bank class that has a single account, and methods to add or subtract money from the account.

```
1  public class Bank {
2      private int account = 100;
3      public static void main(String[] args) {
4          Bank account = new Bank();
5          Thread adding = new Thread(new AddMoney(account));
6          Thread subtracting = new Thread(new TakeMoney(account));
7          adding.start();
8          subtracting.start();
9      }
10
11     public void addMoney(int amount) {
12         if (amount < 1) {
13             System.out.print("Trying to add a negative amount to the account!");
14             System.out.println(" Transaction rejected.");
15             return;
16         }
17         int newBalance = account + amount;
18         System.out.print("Account balance is " + account + ". Adding " + amount + ".");
19         account = newBalance;
20         System.out.println("New balance is " + account + ".");
21         return;
22     }
23
24     public void subtractMoney(int amount) {
25         if (amount < 1) {
26             System.out.print("Trying to subtract a negative amount from the account!");
27             System.out.println(" That's generous, but the transaction is rejected.");
28             return;
29         }
30         int newBalance = account - amount;
31         System.out.print("Account balance is " + account + ". Subtracting " + amount + ".");
```

```
32          account = newBalance;
33          System.out.println("New balance is " + account + ".");
34          return;
35      }
36  }
```

Next we implement our two thread classes.

```
1   public class AddMoney implements Runnable {
2       private Bank account;
3       public AddMoney(Bank account) {
4           this->account = account;
5       }
6       public void run() {
7           for (int i = 0; i < 60; ++i) {
8               try {
9                   Thread.sleep(500);
10              } catch (Exception e) {
11                  System.err.println("Already interrupted.");
12              }
13              account.addMoney(1000);
14          }
15      }
16  }
17
18  public class TakeMoney implements Runnable {
19      private Bank account;
20      public AddMoney(Bank account) {
21          this->account = account;
22      }
23      public void run() {
24          for (int i = 0; i < 60; ++i) {
25              try {
26                  Thread.sleep(500);
27              } catch (Exception e) {
28                  System.err.println("Already interrupted.");
29              }
30              account.subtractMoney(1000);
31          }
32      }
33  }
```

The main method in Bank creates two threads, one to add money, one to remove money. Since the same amount of money is added as is removed, the end result should be that there is no change in the balance. Run the Bank's main method. You should soon see errors in the displayed account balance.

Duration: 15 min

Exercise 2: Hiding the problem

Without using any locking or synchronisation, modify the bank class so that it produces fewer errors in its maths. How effective are your changes if you add more threads?

Duration: 25 min

Note

In this example we could modify the Bank class to reduce the errors produced by concurrent access to the account. However, without some form of synchronisation we still can't guarantee that the result will be correct.