Adedayo Adesokan
CS151 Sec. 01
05/12/20

Extra Credit Write Up

Exercise 9.6:

In this exercise, you're supposed to create multiple threads that access and modify the same data. In this case, the data is the balance of a bank account. Because this data is shared amongst all the threads, if we do not synchronize the methods that affect the data, the bank account can become corrupted. I will use the following images below as examples. In the image below, the methods deposit() and withdrawal() are not synchronized. As a result, we can see that the data gets overridden at certain points because multiple threads are attempting to update the balance at the same time.

*BankAccount.java | *Deposit.java | *Withdrawal.java | *BankAccountTester.java

```java

    /*
     * Adds the given amount to the balance
     *
     * @param amount the amount you want to add to the balance
     */
    public void deposit(double amount) {
        if (amount > 0) {
            balance = balance + amount;
            System.out.println("New balance: $" + balance);
            System.out.println("You deposited: $" + amount);
            System.out.println();
        } else
            System.out.println("Please ensure that you're depositing more than 0 dollars");
    }

    /*
     * Subtracts the given amount from the balance
     *
     * @param amount the amount you want to subtract from the balance
     */
    public void withdrawal(double amount) {
        if (amount <= balance) {
            balance = balance - amount;
            System.out.println("New balance: $" + balance);
            System.out.println("You withdrew: $" + amount);
            System.out.println();
        } else
            System.out.println("Please ensure you have more money in your account than you are withdrawling");
    }
```

Console

```
<terminated> BankAccountTester [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (May 12, 2020, 8:46:49 PM)

New balance: $285.0
You withdrew: $10.0

New balance: $295.0
You deposited: $10.0

New balance: $320.0
New balance: $320.0
You withdrew: $20.0

New balance: $320.0
You withdrew: $5.0

You deposited: $50.0

New balance: $310.0
You withdrew: $10.0

New balance: $290.0
New balance: $300.0
You deposited: $10.0

You withdrew: $20.0

New balance: $295.0
New balance: $345.0
You withdrew: $5.0
```

However, if we make the deposit() and withdrawal() methods synchronized, as seen below, the data can't be overridden. This is possible because calling synchronized acquires the implicit parameter's lock and the lock isn't released until we leave the synchronized method.

*BankAccount.java *Deposit.java *Withdrawal.java *BankAccountTester.java

```java
     * Adds the given amount to the balance
     *
     * @param amount the amount you want to add to the balance
     */
    public synchronized void deposit(double amount) {
        if (amount > 0) {
            balance = balance + amount;
            System.out.println("New balance: $" + balance);
            System.out.println("You deposited: $" + amount);
            System.out.println();
        } else
            System.out.println("Please ensure that you're depositing more than 0 dollars");
    }

    /*
     * Subtracts the given amount from the balance
     *
     * @param amount the amount you want to subtract from the balance
     */
    public synchronized void withdrawal(double amount) {
        if (amount <= balance) {
            balance = balance - amount;
            System.out.println("New balance: $" + balance);
            System.out.println("You withdrew: $" + amount);
            System.out.println();
        } else
            System.out.println("Please ensure you have more money in your account than you are withdrawling");
    }

```

Console

<terminated> BankAccountTester [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (May 12, 2020, 8:16:46 PM)

```
New balance: $2325.0
You withdrew: $5.0

New balance: $2320.0
You withdrew: $5.0

New balance: $2315.0
You withdrew: $5.0

New balance: $2310.0
You withdrew: $5.0

New balance: $2305.0
You withdrew: $5.0

New balance: $2300.0
You withdrew: $5.0

New balance: $2295.0
You withdrew: $5.0

New balance: $2290.0
You withdrew: $5.0

New balance: $2285.0
You withdrew: $5.0

New balance: $2280.0
You withdrew: $5.0

New balance: $2275.0
You withdrew: $5.0

New balance: $2270.0
You withdrew: $5.0
```

Exercise 9.13:

In this exercise, you had to modify the car animation program that we previously worked on in lab 3. The modifications revolved around using threads to make each car move at different speeds. The way I chose to do this was by making the MoveableShape interface extend Runnable. This meant that the CarShape class had to implement the run() method as a result. As I implemented the run() method, I realized that I had to make the label repaint itself inside the method, so I created a private JLabel instance variable and added another method to the class that returned that label. I added this method, getLabel(), to the interface as well because the array list of MoveableShapes in the AnimationTester class also needed access to this label, so that it could add the label to the frame. I then created a thread inside the for-each loop in order to start the run() method of each CarShape in the tester class. Lastly, to make each car move at different speeds, I added a try-catch block to the run() method and called thread.sleep() within it. I then created another instance variable called sleepTime and added it to the constructor, that way each CarShape could have its own unique amount of time that its thread is asleep for.