1. The two thread classes below are used to simulate a queuing system in a bank. The first class `CallingQueue` calls numbers 1 to 10, with a 200ms pause in between each number call. The second method `CustomerInLine` checks whether a given number is called. The third class `BankingQueue` is a driver class.

```
public class CallingQueue implements Runnable{
    private BankingQueue BQ;

    public CallingQueue(BankingQueue BQ) {
        this.BQ = BQ;
    }

    @Override
    public void run() {
      while (BQ.getNextInLine() <= 10) {
        System.out.format("Calling for the customer #%d\n",
BQ.getNextInLine() );
        BQ.incrementNextInLIne();
        try {
          Thread.sleep(200);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
  }

public class CustomerInLine implements Runnable {
    private BankingQueue BQ;
    private int targetNumber;

    public CustomerInLine(BankingQueue BQ, int targetNumber) {
        this.BQ = BQ;
        this.targetNumber = targetNumber;
    }

    @Override
    public void run() {
      while (true) {
        if (BQ.getNextInLine() >= targetNumber) {
          break;
        }
      }
      System.out.format("Great, finally #%d was called, now it is my
turn\n", targetNumber);
    }
  }

public class BankingQueue {
    static int nextInLine = 1;

    public int getNextInLine() {
        return nextInLine;
    }

    public void incrementNextInLIne() {
        nextInLine++;
    }
```

```
  public static void main(String[] args) throws Exception {
      BankingQueue BQ = new BankingQueue();
      Runnable cq = new CallingQueue(BQ);
      Runnable cil = new CustomerInLine(BQ, 4);
      Thread t1 = new Thread(cq);
      Thread t2 = new Thread(cil);
      t1.start();
      t2.start();
  }
}
```

Run the program to see if you can get the expected output below. If not, explain why and provide a solution with minimal modifications to the code.

```
Calling for the customer #1
Calling for the customer #2
Calling for the customer #3
Calling for the customer #4
Great, finally #4 was called, now it is my turn
Calling for the customer #5
Calling for the customer #6
Calling for the customer #7
Calling for the customer #8
Calling for the customer #9
Calling for the customer #10
```

2. In the code below, the `Account` class (with an initial balance of 0) was accessed by 10 threads, each adding 1 dollar to the account balance. Run the program to see if you can get the expected output, i.e. the final balance should be RM10. If not, explain why and provide a lock-free solution.

```
public class Account {
    private int balance = 0;

    public int getBalance() {
        return balance;
    }

    public void deposit(int amount) {
        int newBalance = balance + amount;
        System.out.println("The new balance is " + newBalance);
        try {
            Thread.sleep(5);
        }
        catch (InterruptedException ex) {
        }
        balance = newBalance;

    }
}

public class AddToAccount implements Runnable{
    private Account account = new Account();
```

```
    public AddToAccount(Account acc) {
        account = acc;
    }

    public void run() {
        account.deposit(1);
        System.out.println("Added 1 ringgit.");
    }
}

import java.util.concurrent.*;

public class TestAccount {

    public static void main(String[] args) {
        Account myAccount = new Account();

        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i=0; i<10; i++)
            executor.execute(new AddToAccount(myAccount));

        executor.shutdown();
        while (!executor.isTerminated()) {
        }

        System.out.println("The final balance is RM" +
myAccount.getBalance());
    }
}
```