Exercise 7 (10 points) – can be done in pair or individually

- The first lines of all source files must be comments containing names & IDs of all members. Also create file readme.txt containing names & IDs of all members

- Put all files (source, input, readme.txt) in folder Ex7_xxx where xxx = ID of the group representative, i.e. your source files must be in package Ex7_xxx (assumedly in Maven's src/main/java). Input files must be read from this path

- The group representative zips Ex7_xxx & submits it to Google Classroom. The other members submit only readme.txt. Email submission is not accepted

==

1. Complete class CustomerThread. Add more variables/methods as needed.

```
class CustomerThread extends Thread {
  private Product        product;
  private int            transactions = 5;
  private CyclicBarrier refundBarrier;

  public void run() {
    // Add a loop that process #transactions. For each transaction:

    // - Buy N items by calling product.buy(..)

    // - Wait at refundBarrier until all threads finish buying items

    // - Print the order it arrives at the barrier. The last thread that arrives must
    //   refund all items it bought by calling product.refund(..)

    // - Refunding must be completed before all threads start their next transactions,
    //   i.e. you need another barrier at the end (or the beginning) of each iteration
  }
}
```

2. Complete class Product. Add more variables/methods and modify existing method headers as needed.

```
class Product {
  private int balance;

  public int buy() {
    // - Random #items (0-100 & not exceeding current balance) to buy
    // - Update product balance and report balance update by thread
    // - Return #items (to be used in refunding)
  }

  public void refund(int items) {
    // - Update product balance and report balance update by thread
  }
}
```

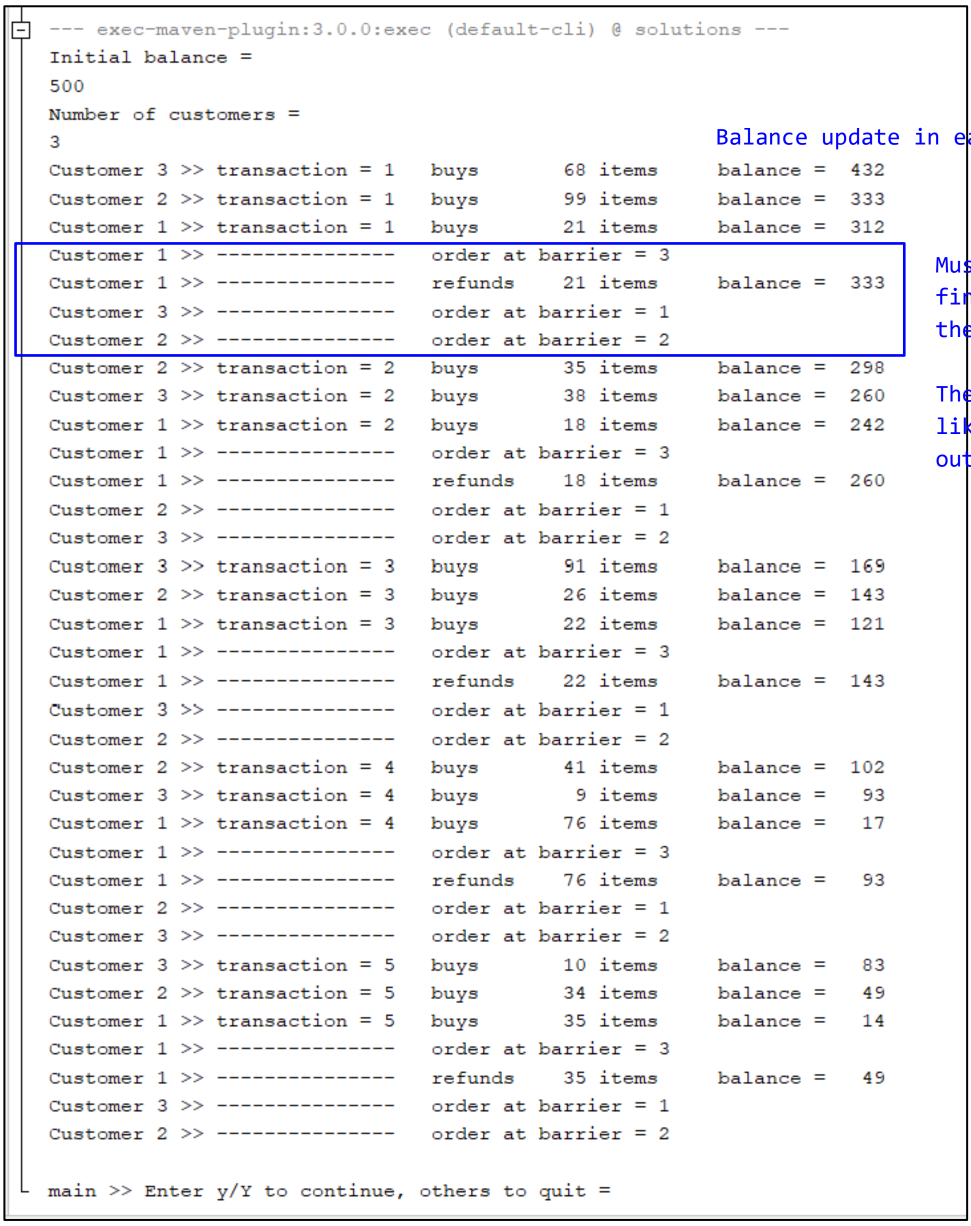
Note: You can use random object to get random integer within a certain bound

```
      Random rand = new Random();              // must import java.util.*;
      int number1 = rand.nextInt(10);          //  0 <= number1 < 10
      int number2 = rand.nextInt(10, 20);      // 10 <= number2 < 20
```

3. Write another class that acts as main class. In its main method
   3.1 Ask user for initial balance. Create 1 Product with this initial balance.

   3.2 Ask user for #customers. Create CustomerThreads and a barrier for synchronizing them (as refundBarrier). All threads must see the same Product & the same barrier.

   3.3 Once all transactions by all threads are completed, ask whether the user wants to continue. If yes, then repeat 3.1 and 3.2



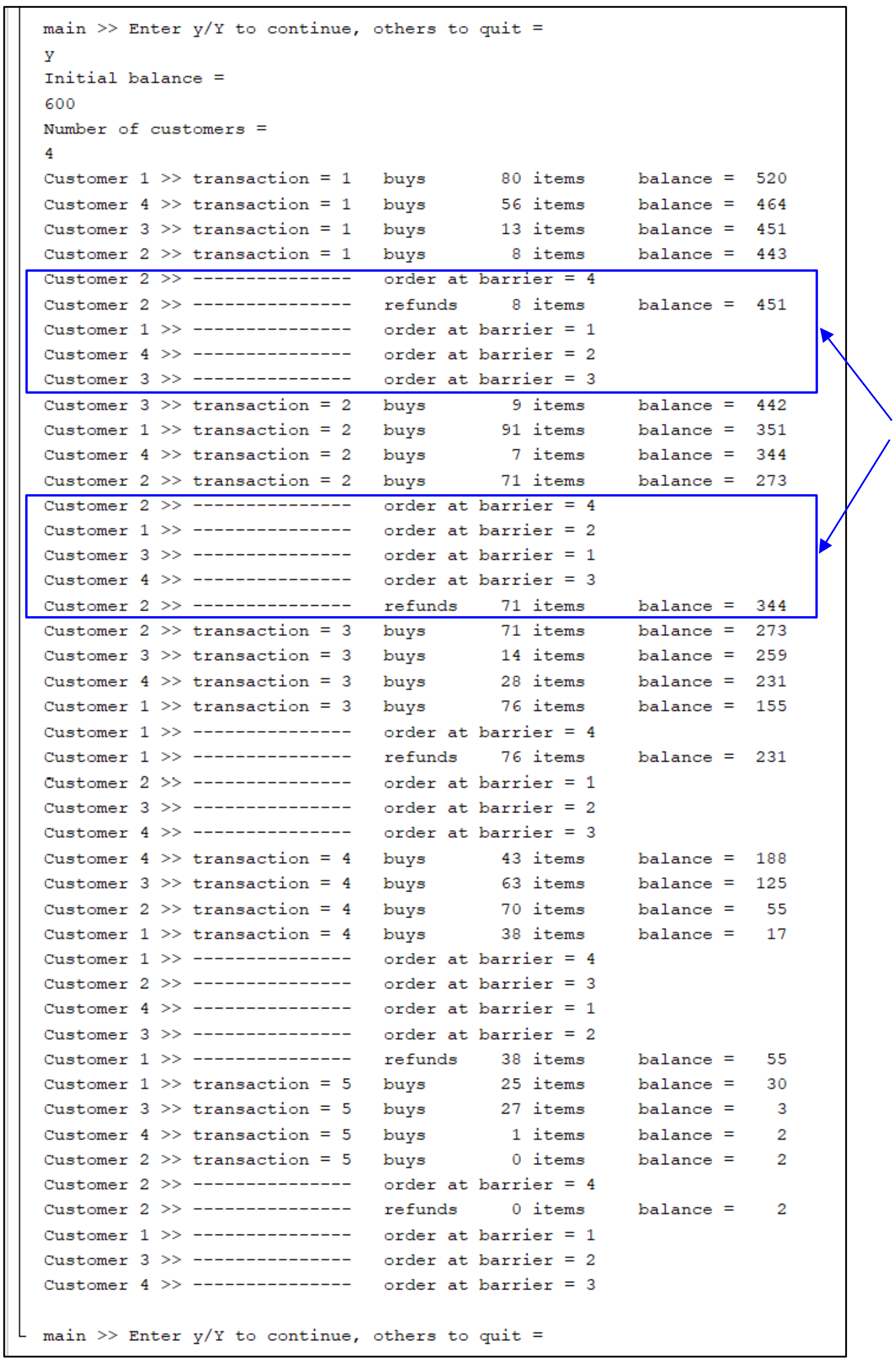
```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ solutions ---
Initial balance =
500
Number of customers =
3
Customer 3 >> transaction = 1   buys       68 items     balance =  432
Customer 2 >> transaction = 1   buys       99 items     balance =  333
Customer 1 >> transaction = 1   buys       21 items     balance =  312
Customer 1 >> ---------------   order at barrier = 3
Customer 1 >> ---------------   refunds    21 items     balance =  333
Customer 3 >> ---------------   order at barrier = 1
Customer 2 >> ---------------   order at barrier = 2
Customer 2 >> transaction = 2   buys       35 items     balance =  298
Customer 3 >> transaction = 2   buys       38 items     balance =  260
Customer 1 >> transaction = 2   buys       18 items     balance =  242
Customer 1 >> ---------------   order at barrier = 3
Customer 1 >> ---------------   refunds    18 items     balance =  260
Customer 2 >> ---------------   order at barrier = 1
Customer 3 >> ---------------   order at barrier = 2
Customer 3 >> transaction = 3   buys       91 items     balance =  169
Customer 2 >> transaction = 3   buys       26 items     balance =  143
Customer 1 >> transaction = 3   buys       22 items     balance =  121
Customer 1 >> ---------------   order at barrier = 3
Customer 1 >> ---------------   refunds    22 items     balance =  143
Customer 3 >> ---------------   order at barrier = 1
Customer 2 >> ---------------   order at barrier = 2
Customer 2 >> transaction = 4   buys       41 items     balance =  102
Customer 3 >> transaction = 4   buys        9 items     balance =   93
Customer 1 >> transaction = 4   buys       76 items     balance =   17
Customer 1 >> ---------------   order at barrier = 3
Customer 1 >> ---------------   refunds    76 items     balance =   93
Customer 2 >> ---------------   order at barrier = 1
Customer 3 >> ---------------   order at barrier = 2
Customer 3 >> transaction = 5   buys       10 items     balance =   83
Customer 2 >> transaction = 5   buys       34 items     balance =   49
Customer 1 >> transaction = 5   buys       35 items     balance =   14
Customer 1 >> ---------------   order at barrier = 3
Customer 1 >> ---------------   refunds    35 items     balance =   49
Customer 3 >> ---------------   order at barrier = 1
Customer 2 >> ---------------   order at barrier = 2

main >> Enter y/Y to continue, others to quit =
```

Balance update in each line must be correct

Must be printed after all threads finish buying & before all of them start next transactions.

The last thread that arrives is likely (but not always) quicker out of the barrier



```
main >> Enter y/Y to continue, others to quit =
y
Initial balance =
600
Number of customers =
4
Customer 1 >> transaction = 1   buys       80 items      balance =  520
Customer 4 >> transaction = 1   buys       56 items      balance =  464
Customer 3 >> transaction = 1   buys       13 items      balance =  451
Customer 2 >> transaction = 1   buys        8 items      balance =  443
Customer 2 >> ---------------   order at barrier = 4
Customer 2 >> ---------------   refunds     8 items      balance =  451
Customer 1 >> ---------------   order at barrier = 1
Customer 4 >> ---------------   order at barrier = 2
Customer 3 >> ---------------   order at barrier = 3
Customer 3 >> transaction = 2   buys        9 items      balance =  442
Customer 1 >> transaction = 2   buys       91 items      balance =  351
Customer 4 >> transaction = 2   buys        7 items      balance =  344
Customer 2 >> transaction = 2   buys       71 items      balance =  273
Customer 2 >> ---------------   order at barrier = 4
Customer 1 >> ---------------   order at barrier = 2
Customer 3 >> ---------------   order at barrier = 1
Customer 4 >> ---------------   order at barrier = 3
Customer 2 >> ---------------   refunds    71 items      balance =  344
Customer 2 >> transaction = 3   buys       71 items      balance =  273
Customer 3 >> transaction = 3   buys       14 items      balance =  259
Customer 4 >> transaction = 3   buys       28 items      balance =  231
Customer 1 >> transaction = 3   buys       76 items      balance =  155
Customer 1 >> ---------------   order at barrier = 4
Customer 1 >> ---------------   refunds    76 items      balance =  231
Customer 2 >> ---------------   order at barrier = 1
Customer 3 >> ---------------   order at barrier = 2
Customer 4 >> ---------------   order at barrier = 3
Customer 4 >> transaction = 4   buys       43 items      balance =  188
Customer 3 >> transaction = 4   buys       63 items      balance =  125
Customer 2 >> transaction = 4   buys       70 items      balance =   55
Customer 1 >> transaction = 4   buys       38 items      balance =   17
Customer 1 >> ---------------   order at barrier = 4
Customer 2 >> ---------------   order at barrier = 3
Customer 4 >> ---------------   order at barrier = 1
Customer 3 >> ---------------   order at barrier = 2
Customer 1 >> ---------------   refunds    38 items      balance =   55
Customer 1 >> transaction = 5   buys       25 items      balance =   30
Customer 3 >> transaction = 5   buys       27 items      balance =    3
Customer 4 >> transaction = 5   buys        1 items      balance =    2
Customer 2 >> transaction = 5   buys        0 items      balance =    2
Customer 2 >> ---------------   order at barrier = 4
Customer 2 >> ---------------   refunds     0 items      balance =    2
Customer 1 >> ---------------   order at barrier = 1
Customer 3 >> ---------------   order at barrier = 2
Customer 4 >> ---------------   order at barrier = 3

main >> Enter y/Y to continue, others to quit =
```

Order line & refund line of the last thread (at the the barrier) need not be consecutive, depending on threads' competition for System.out