1. (Prime Calculator) (50 Points)

In this work, you need to implement a class called **PrimeCalculator**, which **implements** the **Runnable interface**. By using the **run** method, it will calculate the amount of prime numbers between **[lowerBound, upperBound)**. The detail of PrimeCalculator class is as below.

Class PrimeCalculator **implements** Runnable:

- Fields:
  - lowerBound : int, the lower bound of the range.
  - upperBound : int, the upper bound of the range.
  - amount : int, the amount of the prime numbers between **[lowerBound, upperBound)**.
  - ifCalcualate : boolean, the flag indicating whether the result has been calculated.
- Methods:
  - PrimeCalculator(int lowerBound, int upperBound) : public, the constructor of the PrimeCalculator.
  - run() : void, which is **inherited** from the interface and **shall be implemented**.
  - getAmount() : int, which returns the amount of the prime numbers when the result has been calculated, and returns -1 otherwise.

Requirements :

- You should **implement the class** and **complete the test program** as shown below.
- The Classes must contain methods and fields mentioned above, and it also accpetable if your implement has more methods and fields than the requirement.

Test Programs:

- Test code :

```
    public class PrimeCalculatorTest {
    public static void main(String[] args){
        int result = 0;
        PrimeCalculator pc1 = new PrimeCalculator(1, 100000);
        PrimeCalculator pc2 = new PrimeCalculator(100001, 200000);
        PrimeCalculator pc3 = new PrimeCalculator(200001, 300000);
        PrimeCalculator pc4 = new PrimeCalculator(300001, 400000);

        Thread t1 = new Thread(____);
        Thread t2 = new Thread(____);
        Thread t3 = new Thread(____);
        Thread t4 = new Thread(____);

        result = ______;
        System.out.println("The amount of prime number from 1 to 400000 is : " +
String.valueOf(result));

        ______.______(______);
        ______.______(______);
        ______.______(______);
        ______.______(______);

        try {
            ______.______(______);
            ______.______(______);
            ______.______(______);
            ______.______(______);
        }
        catch (InterruptedException e){
            e.printStackTrace();
        }

        result = ______;
        System.out.println("The amount of prime number from 1 to 400000 is : " +
String.valueOf(result));

    }
}
```

- Output :

```
The amount of prime number from 1 to 400000 is : -4
The amount of prime number from 1 to 400000 is : 33860
```

2. (Ticket Window) (50 Points)

In this work, you need to write a class called **TicketWindow**, which will sales tickets to customers. There are ***n*** ticket windows serving customers **simultaneously**, and they will sell ***m*** tickets in total. Tickets are numbered ***1*** to ***m***, and **each ticket shouldn't be sold twice**. The detail of **TicketWindow** class is as below.

Class TicketWindow **implements** Runnable**:**

- **Fields :**
  - windowNumber : int, the index of the window.
  - ticketList : List<Integer>, the List of tickets need to be sold.
- **Methods :**
  - TicketWindow(int windowNumber, List<Integer> ticketList): public, the constructor of TicketWindow
  - void run() : public, which sells tickets in *ticketList* until *ticketList* is empty, and outputs a message to the console in format of "**Window [Window Number]: Ticket [Ticket Number] is sold.**" if a ticket is sold by this window.

Requirements :

- You should **implement the class**. The Class must contain methods and fields mentioned above, and it is also acceptable if your implement has more methods and fields than the requirement.

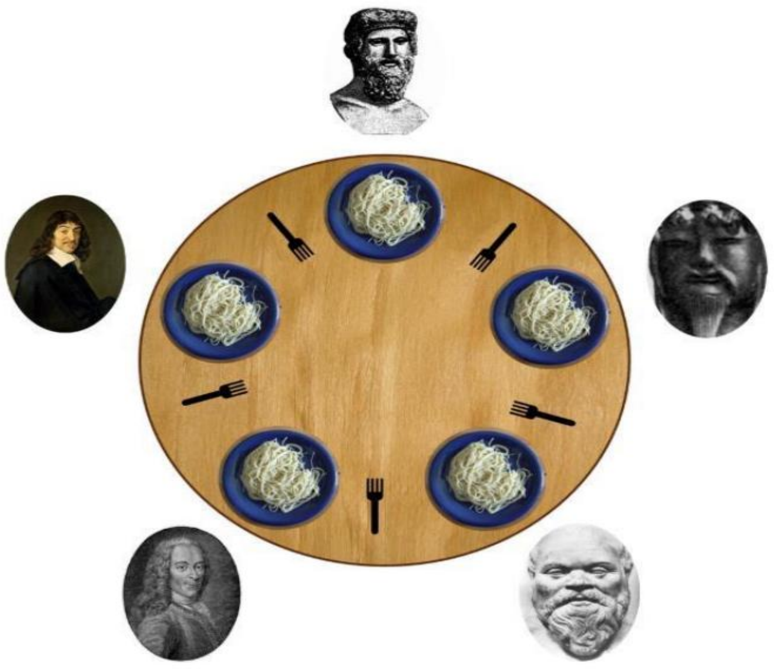
- You should **guarantee *ticketList* is always accessed or modified by one thread at the same time**. Any possible solution is welcome.
- You should **write a test program** consists of **3 ticket windows** and **20 tickets** to test your solution. An example output is shown below.

Example output:

```
Window 1: Ticket 1 is sold.
Window 3: Ticket 2 is sold.
Window 2: Ticket 3 is sold.
Window 2: Ticket 4 is sold.
Window 3: Ticket 5 is sold.
Window 1: Ticket 7 is sold.
Window 3: Ticket 6 is sold.
Window 3: Ticket 8 is sold.
Window 2: Ticket 9 is sold.
Window 2: Ticket 10 is sold.
Window 3: Ticket 11 is sold.
Window 1: Ticket 12 is sold.
Window 1: Ticket 13 is sold.
Window 1: Ticket 14 is sold.
Window 3: Ticket 16 is sold.
Window 3: Ticket 17 is sold.
Window 1: Ticket 15 is sold.
Window 3: Ticket 18 is sold.
Window 3: Ticket 19 is sold.
Window 3: Ticket 20 is sold.
```

3. (Dining Philosopher Problem)(Bonus question: 20Points)

Five silent philosophers numbered from 1 to 5 sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers. Each philosopher must alternately eat and think. However, a philosopher can only eat spaghetti when they have both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can only take the fork on their right or the one on their left as they become available and they cannot start eating before getting both forks. In this work, you need to write a program to simulate this situation.



Class Philosopher **implements** Runnable**:**

- **Fields :**
  - philosopherNumber : int, the index of the window.
- **Methods :**
  - Philosopher(int philosopherNumber) : public, the constructor of Philosopher
  - void run() : public, which simulates the activities of a philsopher, and outputs a message to the console in format of **"Philosopher [Philosopher Number] [do something]."** if he or she do something.

Requirements :

- You should **implement** the class. The Class must contain methods and fields mentioned above, and it is also acceptable if your implement has more methods and fields than the requirement.
- You should **guarantee no deadlock will happen during the progress**. Any possible solution is welcome.
- You should **write a test program to test your solution**. An example output is shown below.

Example output:

```
Philosopher 1 takes left fork.
Philosopher 1 takes right fork.
Philosopher 3 takes left fork.
Philosopher 3 takes right fork.
Philosopher 4 takes left fork.
Philosopher 1 eats spaghetti.
Philosopher 1 puts left fork down.
Philosopher 3 eats spaghetti.
Philosopher 2 takes right fork.
Philosopher 1 puts right fork down.
Philosopher 1 thinks.
Philosopher 3 puts left fork down.
Philosopher 3 puts right fork down.
Philosopher 3 thinks.
Philosopher 2 takes left fork.
Philosopher 2 eats spaghetti.
Philosopher 5 takes left fork.
…
```