T.C.

MANİSA CELAL BAYAR UNIVERSİTY

ENGINEERING FACULTY

COMPUTER ENGINEERING

OPERATING SYSTEMS
PROJECT REPORT

190315058- Havva Beste Tekçeli

Submission Date: *25 Mayıs 2023*

Contents

Phone Operator Problem

There is a phone operation station with two ladies working in. The duty of these ladies is to establish connections between people trying to call each other on two sides of a city. Everything in this station is analogue; there are only two physical cables to make people talk. This means that only two phone calls may occur at the same time.

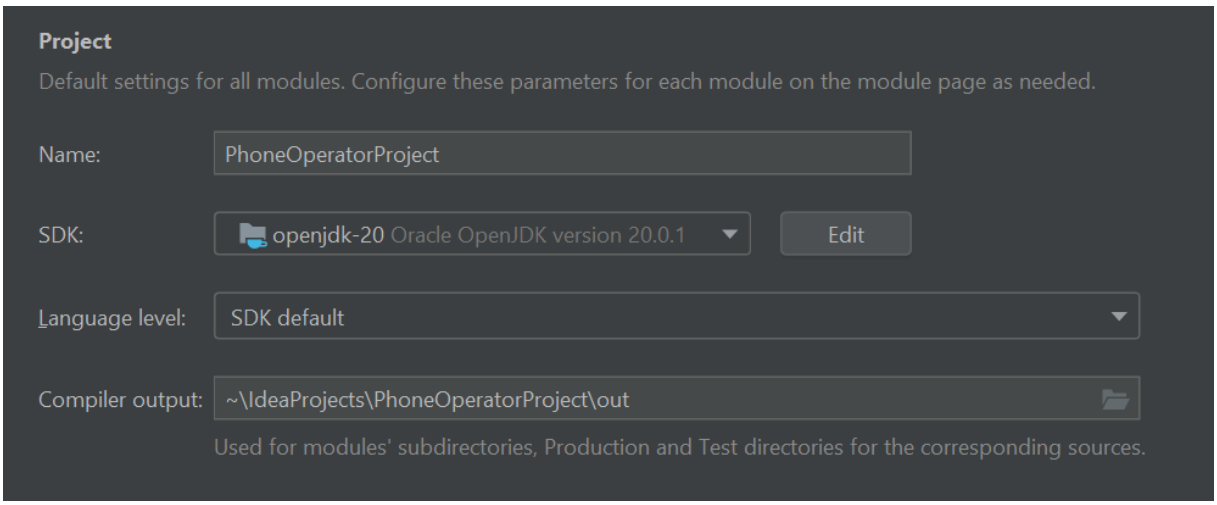
To make a phone call, you first have to catch a lady idle. If she is idle and one of the lines is also free, she can make you talk to your friend. If the ladies or the lines are busy, you have to wait until you catch first the lady, then the line respectively. There are 6 friends on this side of the city (say side A), trying to talk to their 6 common friends on the other side of the city (say side B).

Write a program to simulate the phone calls. Since at most two people can make a phone call with their friends at the same time, the program ends when each of 6 people on side A have talked to 6 people on side B (which makes 36 phone calls). Phone calls should always be established from side A to side B (do not design your solution in a two-direction manner).

The project will be implemented either with C or Java but using threads is a must.

Project Structure

In this Project, I used OpenJDK version 20.0.1



In this project, I have 3 classes apart from my main class. These are: Person, PhoneStation, and PhoneStationManager.

1. PhoneStation

PhoneStation class that represents a phone station with multiple lines.

lineInUse : It is an important variable that represents whether the line is used or not and holds ID which person is speaking on which line in the **PhoneStationManager's makeCall** method.

lock : It is a variable used to lock the used line and not be used by others until it is finished.

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

7 usages
class PhoneStation {
    1 usage
    private Lock lock = new ReentrantLock();
    2 usages
    private int lineInUse = -1;

    4 usages
    public Lock getLock() { return lock; }

    1 usage
    public int getLineInUse() { return lineInUse; }

    2 usages
    public void setLineInUse(int lineInUse) { this.lineInUse = lineInUse; }
}
```

2.PhoneStationManager

PhoneStationManager class manages the phone station and ensures that simultaneous **calls in lines and also it prevents 2 different person can't call same friend**

```
1 usage
private PhoneStation line1 = new PhoneStation();
1 usage
private PhoneStation line2 = new PhoneStation();
2 usages
private Map<Integer, Lock> friendLocks = new HashMap<>();
```

2.1 makeCall method

```
1 usage
public void makeCall(int personID, int friendID) throws InterruptedException {
    PhoneStation[] lines = { line1, line2 };
    Lock friendLock = getFriendLock(friendID); // Retrieve the lock for the friend

    try {
        PhoneStation lineInUse = null;
        friendLock.lock(); // Acquire the lock for the friend
        while (lineInUse == null) {
            for (PhoneStation line : lines) {
                if (line.getLock().tryLock()) {
                    try {
                        if (line.getLineInUse() == -1) {
                            System.out.println("Person " + personID + " is making a call to Friend " + friendID +
                                    " on side B using Line " + (lines[0] == line ? "1" : "2"));
                            line.setLineInUse(personID);
                            lineInUse = line;
                            break;
                        }
                    } finally {
                        line.getLock().unlock();
                    }
                }
            }
        }

        Thread.sleep( millis: 1000); // Simulate the call duration

        lineInUse.getLock().lock();
        try {
            System.out.println("Person " + personID + " has finished the call on Line " +
                    (lines[0] == lineInUse ? "1" : "2"));
            lineInUse.setLineInUse(-1);
        } finally {
            lineInUse.getLock().unlock();
        }
    } finally {
        friendLock.unlock(); // Release the lock for the friend
    }
}
```

The **getFriendLock(friendID)** method is called to retrieve the lock associated with the friendID. This lock prevents for making calls to the same friend simultaneously. It iterates through the **lines array**, which contains the two PhoneStation objects **line1** and **line2**. For each line, it tries to acquire the lock associated with that line using **line.getLock().tryLock().**

If the lock is acquired (**tryLock() returns true**), it checks if the line is not currently in use (**line.getLineInUse() == -1**). If an available line is found, it enters the if block**,** indicating that the person can make the call using that line.

Inside the second if block, it prints a message indicating that the person (**personID**) is making a call to the friend (**friendID**) using the line number (1 or 2). It sets the line as "in use" by the person by calling **line.setLineInUse(personID)**. The variable **lineInUse** is assigned the current line to track which line is being used.

The code simulates the call duration by calling Thread.sleep(1000) for 1 second.

After the simulated call duration, it locks the **lineInUse** to perform operations related to ending the call using lineInUse.getLock().lock().It prints a message indicating that the person (**personID**) has finished the call on the line number 1 or 2. It sets the line as available for future calls by calling **lineInUse.setLineInUse(-1).**

Then It will be releasing the lock which is for using line **lineInUse.getLock().unlock()** and for friend **friendLock.unlock()**

3.Person

A class named "Person" that extends the built-in Java class "Thread." In Java, extending the "Thread" class allows a class to be executed as a separate thread of execution.

```java
import java.util.List;

2 usages
class Person extends Thread {
    2 usages
    private int personId;
    2 usages
    private PhoneStationManager phoneStationManager;
    5 usages
    private List<Integer> friendIdList;


    1 usage
    public Person(int personId, PhoneStationManager phoneStationManager, List<Integer> friendIdList) {
        this.personId = personId;
        this.phoneStationManager = phoneStationManager;
        this.friendIdList = friendIdList;
    }
```

3.1 run()

It contains a loop that randomly selects a friend from the list, removes the **friendID** from the list, makes a call using the **phoneStationManager**, and pauses before making the next call. The loop continues until the **friendIdList** becomes empty.

```java
    @Override
    public void run() {
        try {
            while (!friendIdList.isEmpty()) {
                int randomFriendIndex = (int) (Math.random() * friendIdList.size());
                int friendId = this.friendIdList.get(randomFriendIndex);
                friendIdList.remove(randomFriendIndex);
                phoneStationManager.makeCall(personId, friendId);
                Thread.sleep( millis: 1000); // Pause between making each call
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

4.Main

personIds and **friendIds** list with integer values 1, 2, 3, 4, 5, and 6 using the **Arrays.asList()** method. It creates an ArrayList instance based on the provided values. Collections.shuffle() shuffles the order of elements in the **personIds** and **friendIds** list randomly. Inside the loop, a new **clonedFriendIds** list is created by making a copy of the **friendIds** list. This ensures that each person has their own separate list of friend IDs.

The start() method is called on each Person object. This method starts the execution of the thread associated with the Person object, causing the run() method of the Person class to be invoked.

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Integer> personIds = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6));
        List<Integer> friendIds = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6));

        Collections.shuffle(personIds);
        Collections.shuffle(friendIds);


        PhoneStationManager phoneStationManager = new PhoneStationManager();

        for (int personId : personIds) {
            List<Integer> clonedFriendIds = new ArrayList<>(friendIds);
            Person person = new Person(personId, phoneStationManager, clonedFriendIds);
            person.start();
        }
    }
}
```

Output:

Each person can call their friends using line 1 and line 2 at the same time. It can be used by another or same person after you have finished calling. The output will change each time the program is running , but there will always be 36 calls in total.

Note: The photo below does not show the whole output.

```
Person 6 is making a call to Friend 6 on side B using Line 2
Person 3 is making a call to Friend 1 on side B using Line 1
Person 3 has finished the call on Line 1
Person 6 has finished the call on Line 2
Person 1 is making a call to Friend 2 on side B using Line 1
Person 5 is making a call to Friend 5 on side B using Line 2
Person 5 has finished the call on Line 2
Person 1 has finished the call on Line 1
Person 2 is making a call to Friend 1 on side B using Line 2
Person 3 is making a call to Friend 3 on side B using Line 1
Person 2 has finished the call on Line 2
Person 6 is making a call to Friend 4 on side B using Line 2
Person 3 has finished the call on Line 1
Person 5 is making a call to Friend 2 on side B using Line 1
Person 6 has finished the call on Line 2
Person 1 is making a call to Friend 5 on side B using Line 2
Person 5 has finished the call on Line 1
Person 4 is making a call to Friend 1 on side B using Line 1
Person 1 has finished the call on Line 2
Person 2 is making a call to Friend 6 on side B using Line 2
Person 4 has finished the call on Line 1
Person 3 is making a call to Friend 4 on side B using Line 1
Person 2 has finished the call on Line 2
Person 6 is making a call to Friend 5 on side B using Line 2
Person 3 has finished the call on Line 1
Person 5 is making a call to Friend 6 on side B using Line 1
Person 6 has finished the call on Line 2
Person 5 has finished the call on Line 1
Person 1 is making a call to Friend 4 on side B using Line 1
Person 4 is making a call to Friend 3 on side B using Line 2
Person 1 has finished the call on Line 1
Person 3 is making a call to Friend 5 on side B using Line 1
Person 4 has finished the call on Line 2
Person 5 is making a call to Friend 3 on side B using Line 2
Person 5 has finished the call on Line 2
Person 3 has finished the call on Line 1
```