# CONTENTS

## 1. INTRODUCTION

The Phone Operator project simulates a telephone operation station where two ladies work. The station has only two physical wires to connect people on both sides of a city. The project aims to simulate this scenario using Java language and threads. Our goal is to have six people representing one side make phone calls to six mutual friends on the other side. The simulation focuses on a scenario where negotiations take place from one side to the other and end when all connections are made.

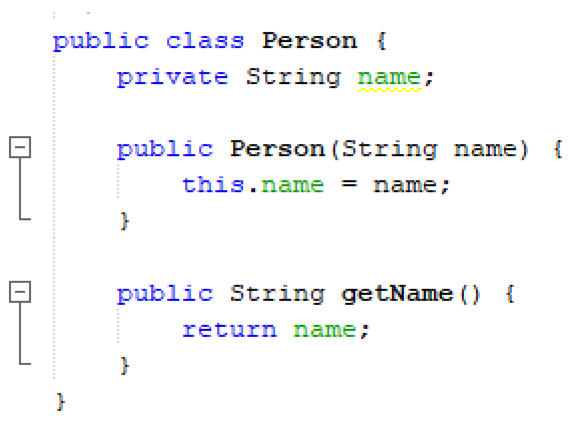
In this project OpLadyand Line classes are created. Next, two lists are created, cityA and cityB, containing objects from the Person class. Then, phone calls are made using OpLady, Line and Person objects according to a specific scenario.

## 2. CLASSES AND CODES

I created Main, Line, Person and OpLady classes.

### 2.1. Person Class

First I created the **Person class**. This class is a struct that represents a person's name. The class has only one property: name is a private string variable that holds the person's name. When creating a person object with the constructor of the class **Person(String name),** the person's name is determined and the name property is set with this name. The **getName()** method returns the name of the person.

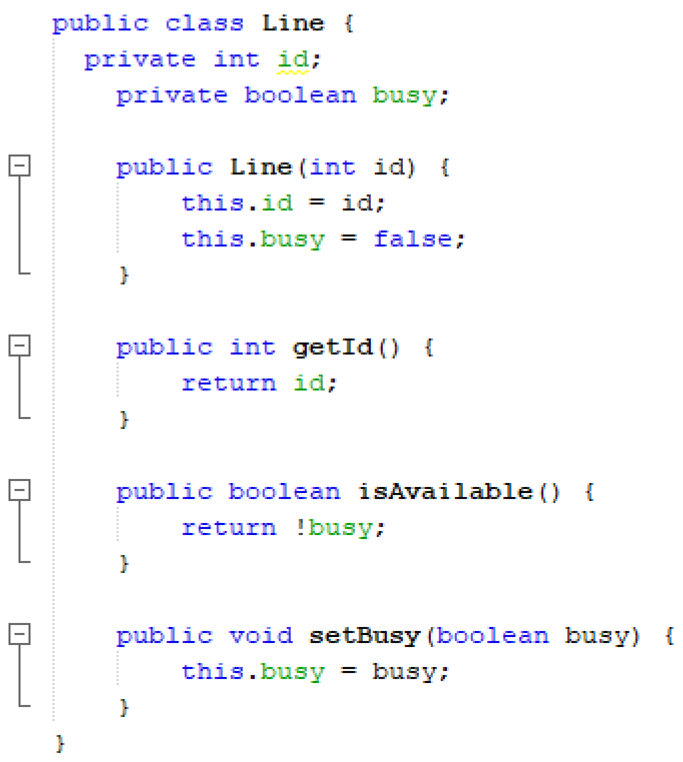


```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

**Figure1:** Person Class Code

## 2.2. Line Class

The **'Line' class** represents a telephone line. Each phone line has an identification number **(`id`)** and busy status **(`busy`).** When creating a phone line object with the class's constructor **`Line(int id)`** the ID number is assigned and by default the busy status of the line is set to **`false`.** The **`getId()`** method returns the ID of the line. The **`isAvailable()`** method checks if the line is busy and returns **`true`** if not, **`false`** otherwise. The **`setBusy(boolean busy)`** method is used to determine the busy status of the line. This class is used to monitor the status of telephone lines and to check lines that are busy or free.

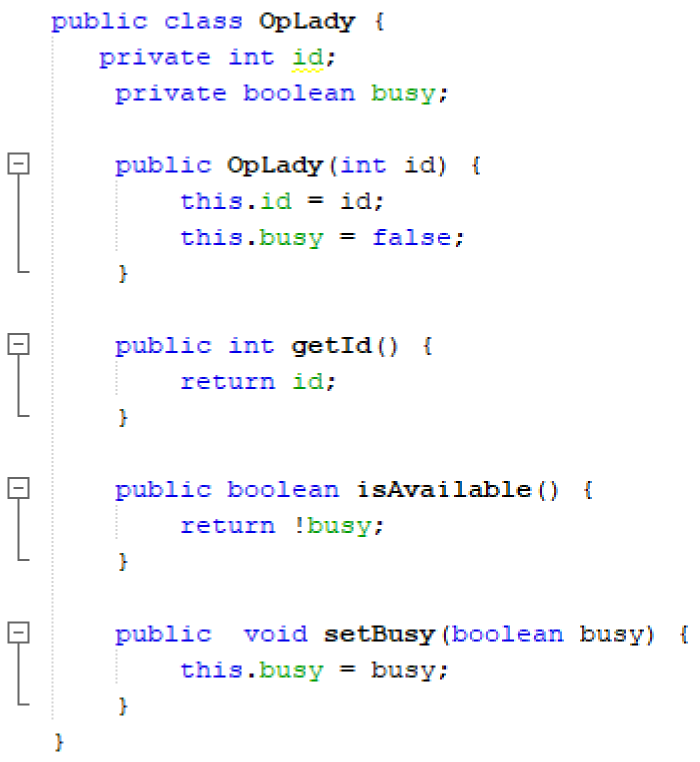


```
public class Line {
  private int id;
    private boolean busy;

    public Line(int id) {
        this.id = id;
        this.busy = false;
    }

    public int getId() {
        return id;
    }

    public boolean isAvailable() {
        return !busy;
    }

    public void setBusy(boolean busy) {
        this.busy = busy;
    }
}
```

**Figure2:** Line Class Code

## 2.3. OpLady Class

The **`OpLady` class** represents a telephone operator. Each operator carries an identification number **(`id`)** and busy status **(`busy`).** When creating an operator object with the constructor of the class **`OpLady(int id)`** the ID number is assigned and by default the busy state of the operator is set to **`false`.** The **`getId()`** method returns the operator's **ID** number. The **`isAvailable()`** method checks if the operator is busy and returns **`true`** if not, **`false`** otherwise. The **`setBusy(boolean busy)`** method is used to determine the busy state of the operator. This class can be used to monitor the status of operators and check operators that are busy or free.

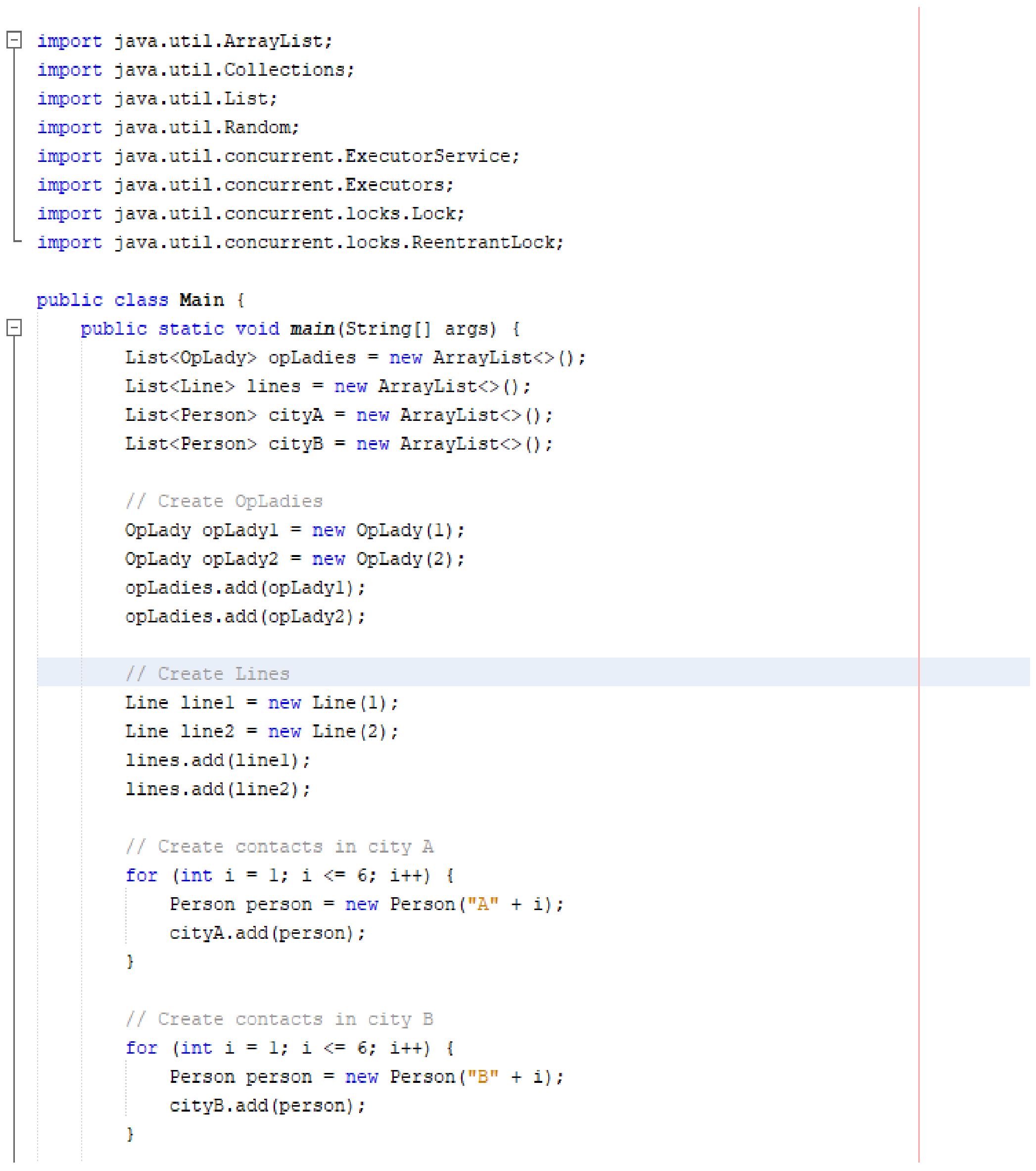


```
public class OpLady {
    private int id;
     private boolean busy;

     public OpLady(int id) {
         this.id = id;
         this.busy = false;
     }

     public int getId() {
         return id;
     }

     public boolean isAvailable() {
         return !busy;
     }

     public  void setBusy(boolean busy) {
         this.busy = busy;
     }
}
```

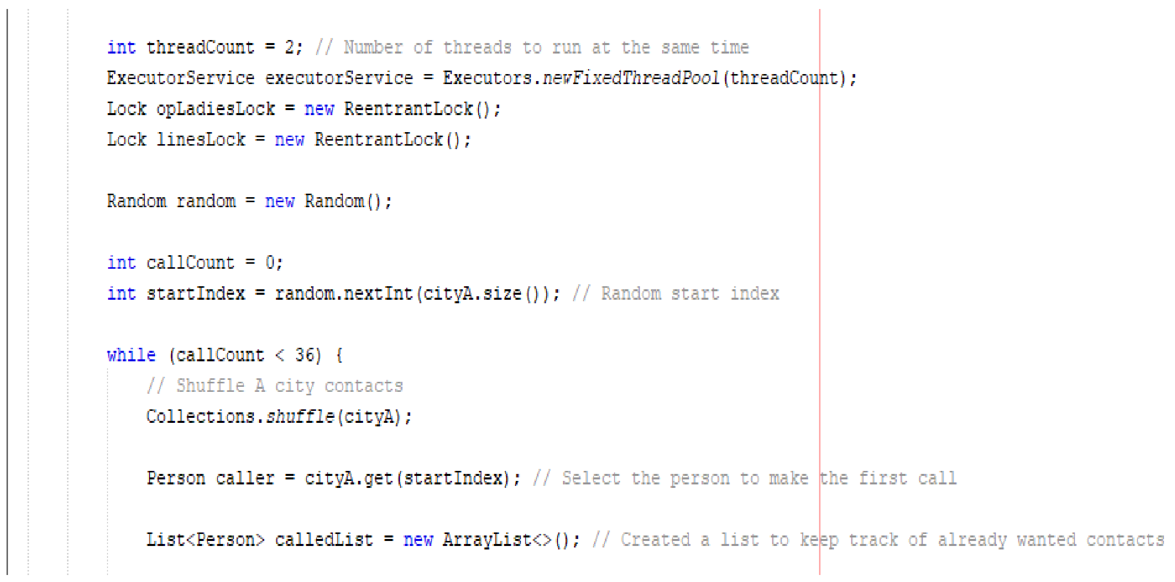
**Figure3:** OpLady Class Code

## 2.4. Main Class

The **`Main` class** represents a program that simulates the telephone operation station. The program handles phone calls between two telephone operators, two telephone lines, and people in two cities. Up to two simultaneous phone calls can be made on the station. The program allows 6 people in city A to talk to their 6 mutual friends in city B. Phone calls are always made from A to B. Threads are used to run processes synchronously. The status of operators and the status of telephone lines are securely controlled using locks. The program runs until 36 phone calls are made and then closes the thread pool.

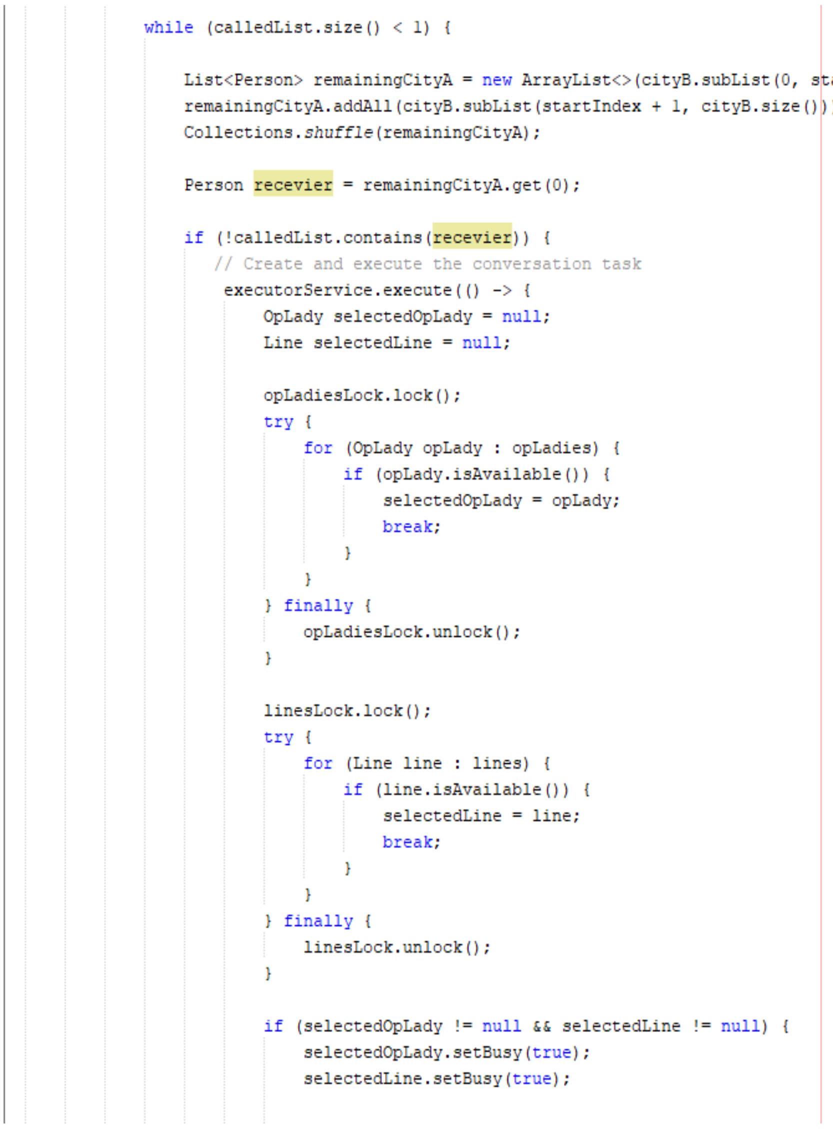


```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Main {
    public static void main(String[] args) {
        List<OpLady> opLadies = new ArrayList<>();
        List<Line> lines = new ArrayList<>();
        List<Person> cityA = new ArrayList<>();
        List<Person> cityB = new ArrayList<>();

        // Create OpLadies
        OpLady opLady1 = new OpLady(1);
        OpLady opLady2 = new OpLady(2);
        opLadies.add(opLady1);
        opLadies.add(opLady2);

        // Create Lines
        Line line1 = new Line(1);
        Line line2 = new Line(2);
        lines.add(line1);
        lines.add(line2);

        // Create contacts in city A
        for (int i = 1; i <= 6; i++) {
            Person person = new Person("A" + i);
            cityA.add(person);
        }

        // Create contacts in city B
        for (int i = 1; i <= 6; i++) {
            Person person = new Person("B" + i);
            cityB.add(person);
        }
```

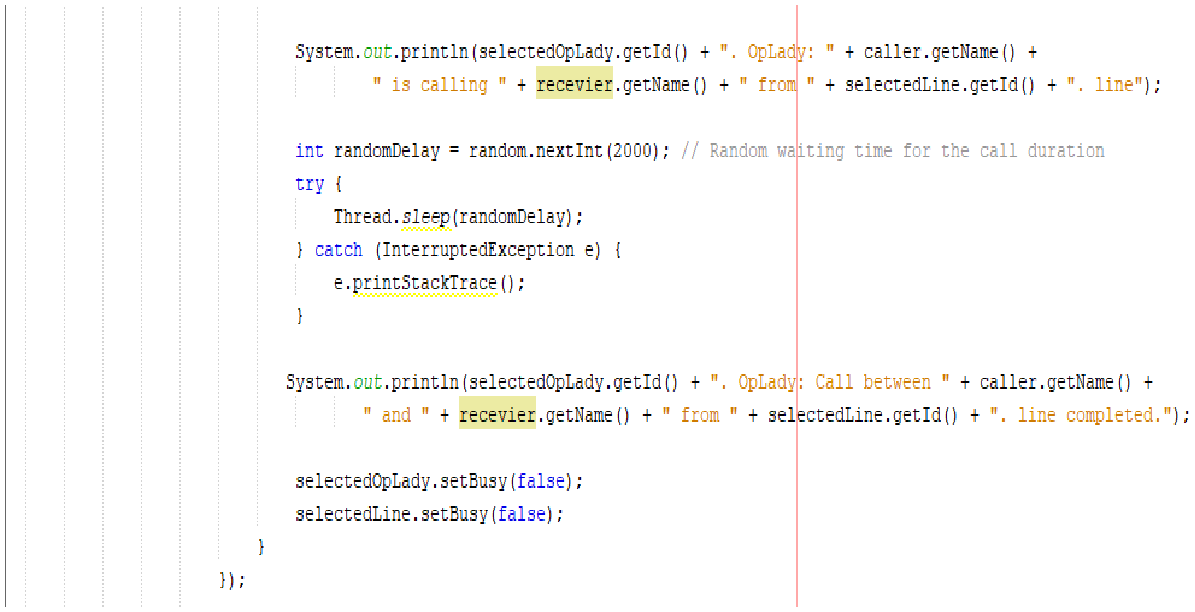


```java
        int threadCount = 2; // Number of threads to run at the same time
        ExecutorService executorService = Executors.newFixedThreadPool(threadCount);
        Lock opLadiesLock = new ReentrantLock();
        Lock linesLock = new ReentrantLock();

        Random random = new Random();

        int callCount = 0;
        int startIndex = random.nextInt(cityA.size()); // Random start index

        while (callCount < 36) {
            // Shuffle A city contacts
            Collections.shuffle(cityA);

            Person caller = cityA.get(startIndex); // Select the person to make the first call

            List<Person> calledList = new ArrayList<>(); // Created a list to keep track of already wanted contacts
```

**Figure4.1:** Main Class Code

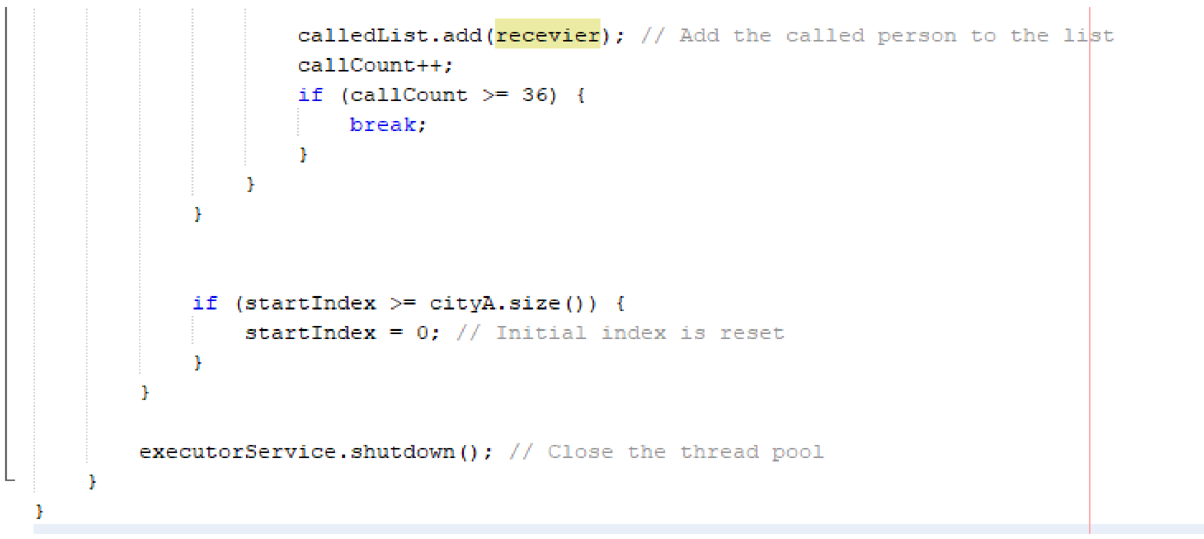


```
while (calledList.size() < 1) {

    List<Person> remainingCityA = new ArrayList<>(cityB.subList(0, startIndex));
    remainingCityA.addAll(cityB.subList(startIndex + 1, cityB.size()));
    Collections.shuffle(remainingCityA);

    Person recevier = remainingCityA.get(0);

    if (!calledList.contains(recevier)) {
       // Create and execute the conversation task
        executorService.execute(() -> {
            OpLady selectedOpLady = null;
            Line selectedLine = null;

            opLadiesLock.lock();
            try {
                for (OpLady opLady : opLadies) {
                    if (opLady.isAvailable()) {
                        selectedOpLady = opLady;
                        break;
                    }
                }
            } finally {
                opLadiesLock.unlock();
            }

            linesLock.lock();
            try {
                for (Line line : lines) {
                    if (line.isAvailable()) {
                        selectedLine = line;
                        break;
                    }
                }
            } finally {
                linesLock.unlock();
            }

            if (selectedOpLady != null && selectedLine != null) {
                selectedOpLady.setBusy(true);
                selectedLine.setBusy(true);
```



```
                System.out.println(selectedOpLady.getId() + ". OpLady: " + caller.getName() +
                        " is calling " + recevier.getName() + " from " + selectedLine.getId() + ". line");

                int randomDelay = random.nextInt(2000); // Random waiting time for the call duration
                try {
                    Thread.sleep(randomDelay);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

               System.out.println(selectedOpLady.getId() + ". OpLady: Call between " + caller.getName() +
                       " and " + recevier.getName() + " from " + selectedLine.getId() + ". line completed.");

                selectedOpLady.setBusy(false);
                selectedLine.setBusy(false);
            }
        });
```

**Figure4.2:** Main Class Code

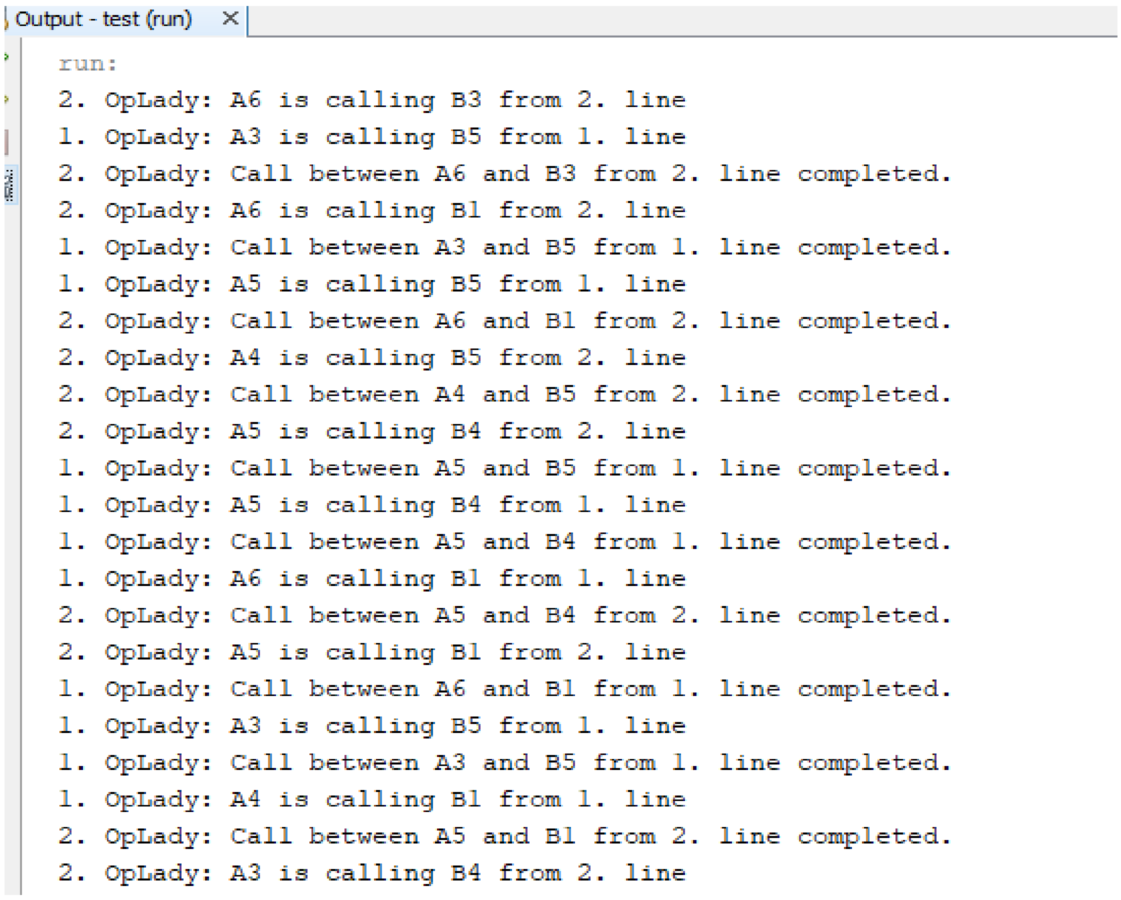


```
                    calledList.add(recevier); // Add the called person to the list
                    callCount++;
                    if (callCount >= 36) {
                        break;
                    }
                }
            }


            if (startIndex >= cityA.size()) {
                startIndex = 0; // Initial index is reset
            }
        }

        executorService.shutdown(); // Close the thread pool
    }
}
```

**Figure4.3:** Main Class Code

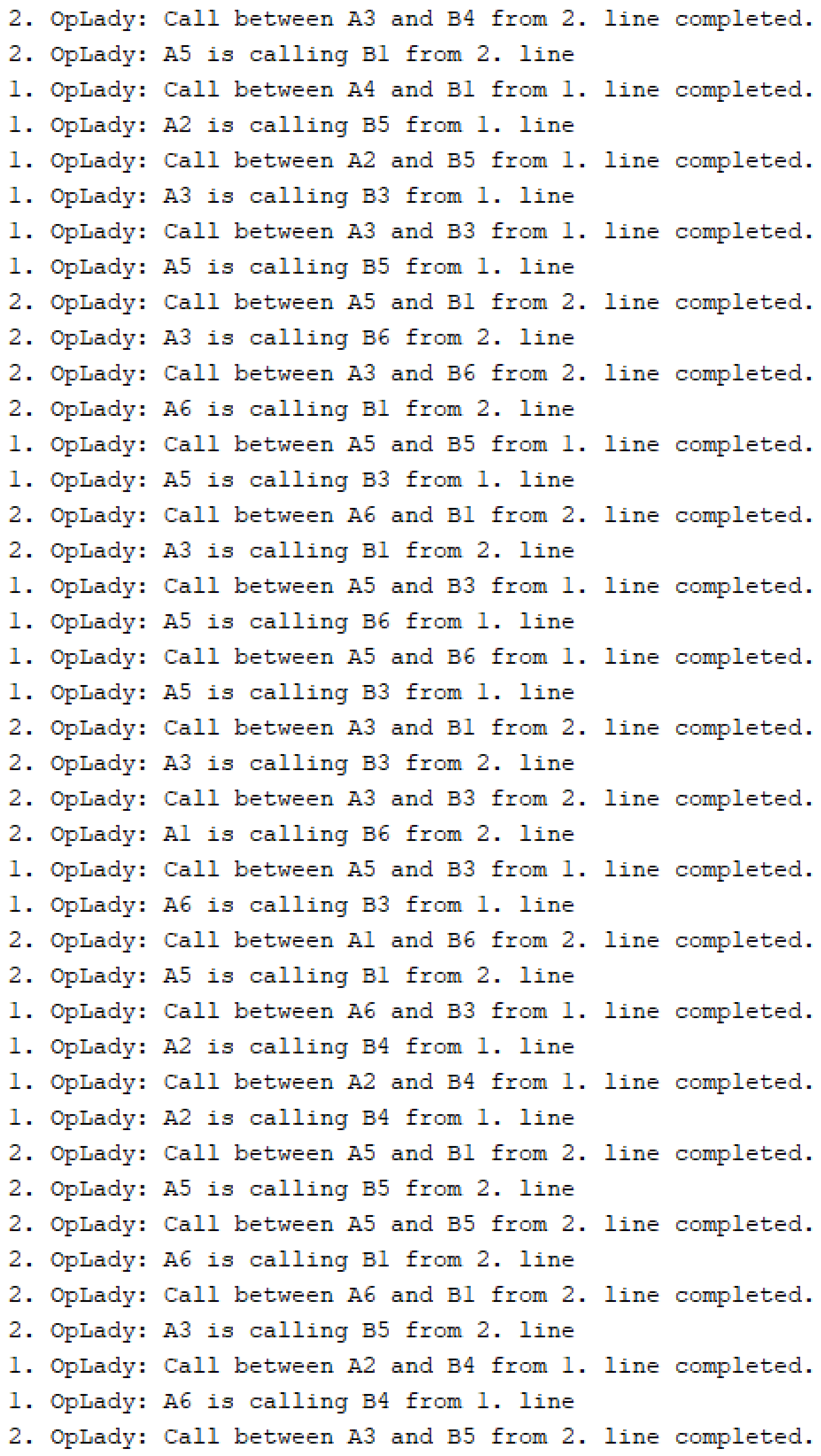
## 3. PROGRAM OUTPUT

**Total Output Number:** 36



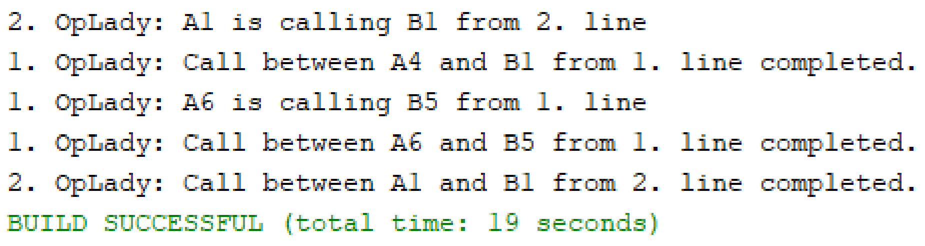
```
run:
2. OpLady: A6 is calling B3 from 2. line
1. OpLady: A3 is calling B5 from 1. line
2. OpLady: Call between A6 and B3 from 2. line completed.
2. OpLady: A6 is calling B1 from 2. line
1. OpLady: Call between A3 and B5 from 1. line completed.
1. OpLady: A5 is calling B5 from 1. line
2. OpLady: Call between A6 and B1 from 2. line completed.
2. OpLady: A4 is calling B5 from 2. line
2. OpLady: Call between A4 and B5 from 2. line completed.
2. OpLady: A5 is calling B4 from 2. line
1. OpLady: Call between A5 and B5 from 1. line completed.
1. OpLady: A5 is calling B4 from 1. line
1. OpLady: Call between A5 and B4 from 1. line completed.
1. OpLady: A6 is calling B1 from 1. line
2. OpLady: Call between A5 and B4 from 2. line completed.
2. OpLady: A5 is calling B1 from 2. line
1. OpLady: Call between A6 and B1 from 1. line completed.
1. OpLady: A3 is calling B5 from 1. line
1. OpLady: Call between A3 and B5 from 1. line completed.
1. OpLady: A4 is calling B1 from 1. line
2. OpLady: Call between A5 and B1 from 2. line completed.
2. OpLady: A3 is calling B4 from 2. line
```

**Figure5.1:** Program Output



```
2. OpLady: Call between A3 and B4 from 2. line completed.
2. OpLady: A5 is calling B1 from 2. line
1. OpLady: Call between A4 and B1 from 1. line completed.
1. OpLady: A2 is calling B5 from 1. line
1. OpLady: Call between A2 and B5 from 1. line completed.
1. OpLady: A3 is calling B3 from 1. line
1. OpLady: Call between A3 and B3 from 1. line completed.
1. OpLady: A5 is calling B5 from 1. line
2. OpLady: Call between A5 and B1 from 2. line completed.
2. OpLady: A3 is calling B6 from 2. line
2. OpLady: Call between A3 and B6 from 2. line completed.
2. OpLady: A6 is calling B1 from 2. line
1. OpLady: Call between A5 and B5 from 1. line completed.
1. OpLady: A5 is calling B3 from 1. line
2. OpLady: Call between A6 and B1 from 2. line completed.
2. OpLady: A3 is calling B1 from 2. line
1. OpLady: Call between A5 and B3 from 1. line completed.
1. OpLady: A5 is calling B6 from 1. line
1. OpLady: Call between A5 and B6 from 1. line completed.
1. OpLady: A5 is calling B3 from 1. line
2. OpLady: Call between A3 and B1 from 2. line completed.
2. OpLady: A3 is calling B3 from 2. line
2. OpLady: Call between A3 and B3 from 2. line completed.
2. OpLady: A1 is calling B6 from 2. line
1. OpLady: Call between A5 and B3 from 1. line completed.
1. OpLady: A6 is calling B3 from 1. line
2. OpLady: Call between A1 and B6 from 2. line completed.
2. OpLady: A5 is calling B1 from 2. line
1. OpLady: Call between A6 and B3 from 1. line completed.
1. OpLady: A2 is calling B4 from 1. line
1. OpLady: Call between A2 and B4 from 1. line completed.
1. OpLady: A2 is calling B4 from 1. line
2. OpLady: Call between A5 and B1 from 2. line completed.
2. OpLady: A5 is calling B5 from 2. line
2. OpLady: Call between A5 and B5 from 2. line completed.
2. OpLady: A6 is calling B1 from 2. line
2. OpLady: Call between A6 and B1 from 2. line completed.
2. OpLady: A3 is calling B5 from 2. line
1. OpLady: Call between A2 and B4 from 1. line completed.
1. OpLady: A6 is calling B4 from 1. line
2. OpLady: Call between A3 and B5 from 2. line completed.
```

**Figure5.2:** Program Output



```
2. OpLady: A1 is calling B1 from 2. line
1. OpLady: Call between A4 and B1 from 1. line completed.
1. OpLady: A6 is calling B5 from 1. line
1. OpLady: Call between A6 and B5 from 1. line completed.
2. OpLady: Call between A1 and B1 from 2. line completed.
BUILD SUCCESSFUL (total time: 19 seconds)
```

**Figure5.3:** Program Output