Synchronization Simulator Project Report

Student Name: Lujain Amjad AbuRajab
Student No: 211045

Overview :

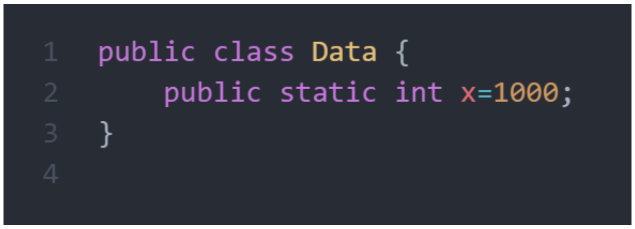
After reviewing the problem, a program was designed to compare the system before and after synchronization as follows:

- **The programming language used:** Java
- **Basic Classes:** Consumer , Producer , Data, OS(main class without Synchronization), OSWithSync.
- **The technique used:** Threads
- **Types of Threads:** Consumer threads , Producer threads

Classes:

Data Class:

Contains Global shared Variable X Between Threads which is the only variable in this case with an initial value equal to 1000.



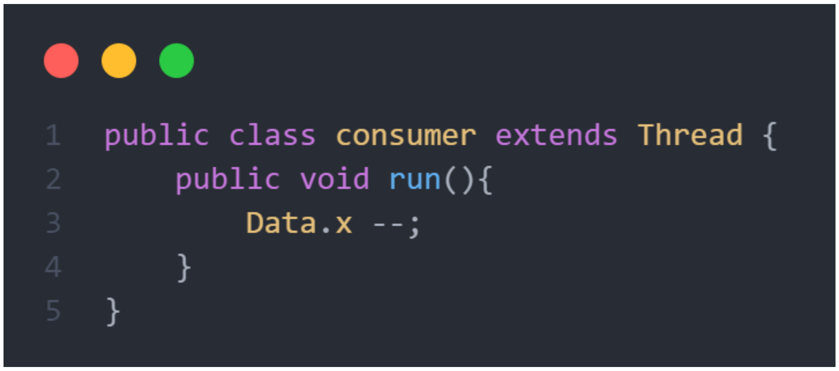
```java
public class Data {
    public static int x=1000;
}
```

Consumer Class:

Consumer Class is Provider of Threads of Consumer type , the main task for this kind of threads is to Consume the value of some data.

Technique Explanation:

As mentioned above , we have here a random number of threads that Produce the value of X , The implementation of this task is by override the method run() Form Thread Class.



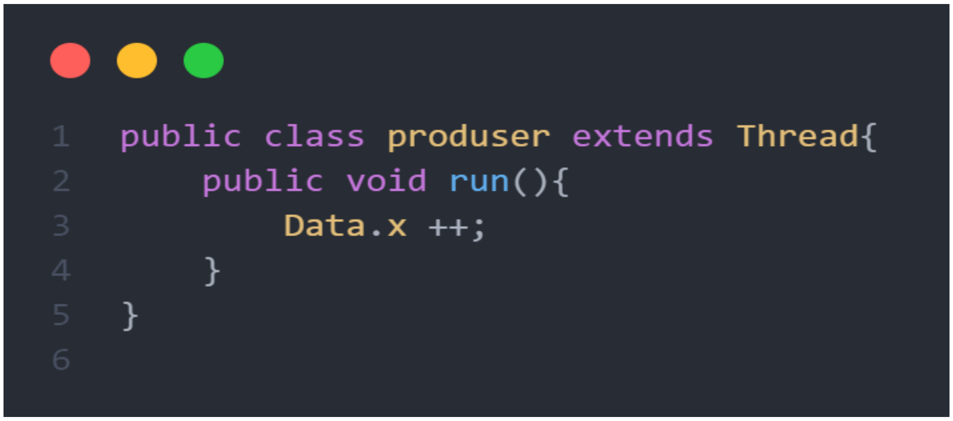
```java
public class consumer extends Thread {
    public void run(){
        Data.x --;
    }
}
```

Producer Class:

Producer Class is Provider of Threads of Producer type , the main task for this kind of threads is to produce the value of some data.

Technique Explanation:

As mentioned above , we have here a random number of threads that Produce the value of X , The implementation of this task is by override the method run() Form Thread Class.



```
public class produser extends Thread{
    public void run(){
        Data.x ++;
    }
}

```

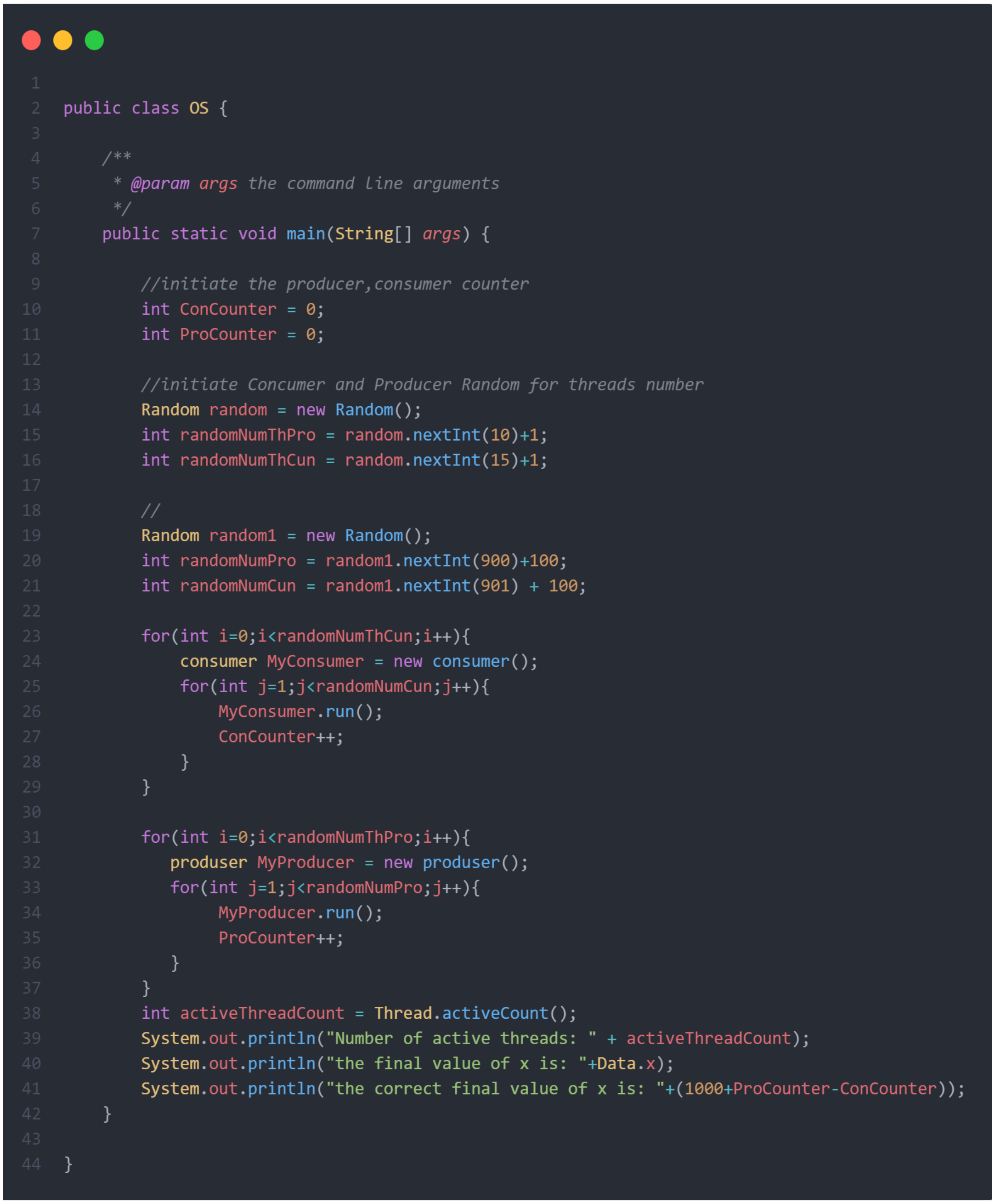
OS Class:

OS Class Provider random number of Threads of Producer and Consumer type with difference range for each then Execute it’s task (if it’s producer will add , Otherwise it Will consume), in additinal there are a random number of produce or consume for each thread.

There are 2 main output in here, the first one will show us the result without synchronization, and the second one will show us the true result with Synchronization.

Note:

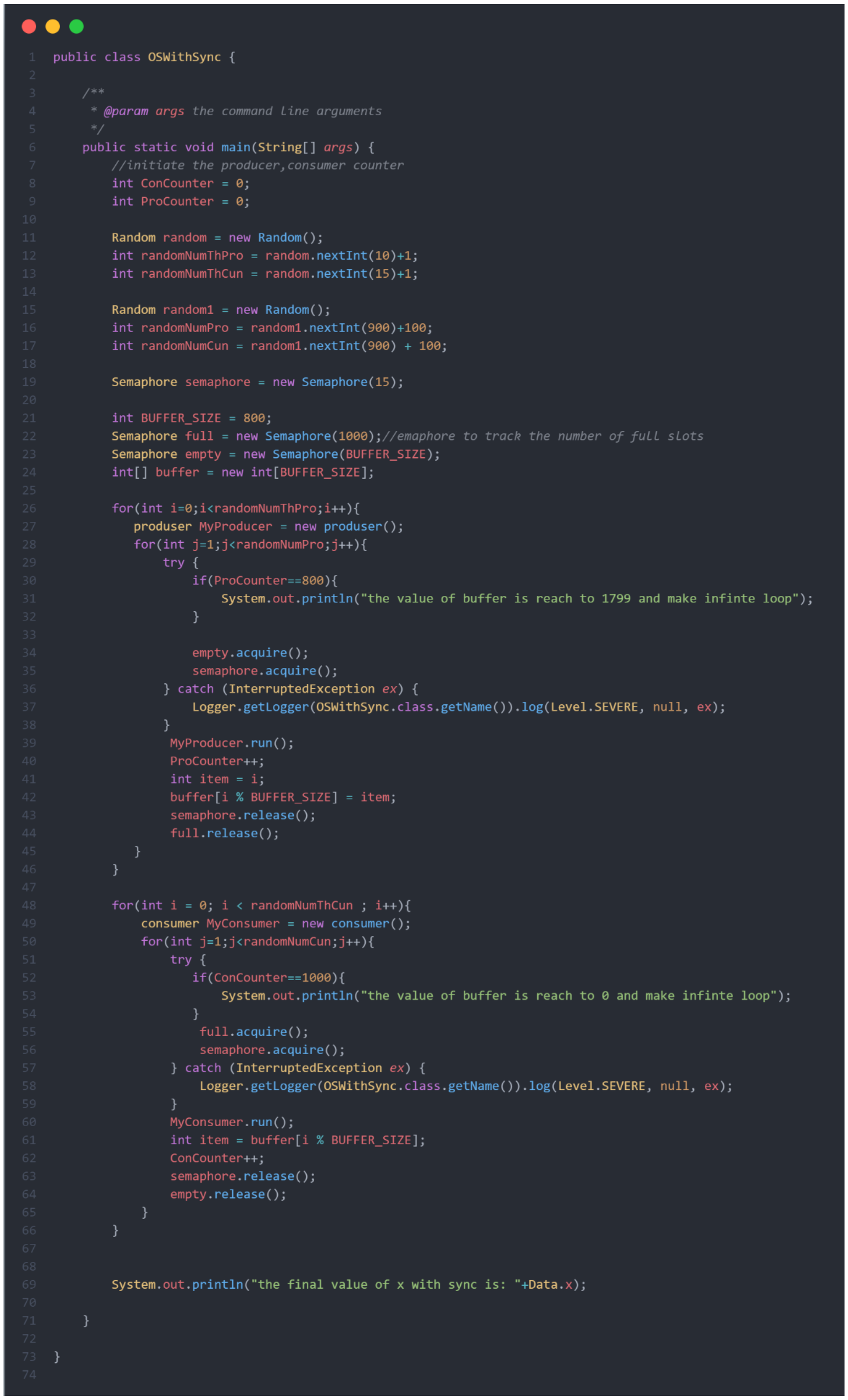
The implementation here doesn't work as expected because “OSWithSync” Class work on the same core with this file so the OS will execute both of them with Synchronization.



```java

public class OS {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        //initiate the producer,consumer counter
        int ConCounter = 0;
        int ProCounter = 0;

        //initiate Concumer and Producer Random for threads number
        Random random = new Random();
        int randomNumThPro = random.nextInt(10)+1;
        int randomNumThCun = random.nextInt(15)+1;

        //
        Random random1 = new Random();
        int randomNumPro = random1.nextInt(900)+100;
        int randomNumCun = random1.nextInt(901) + 100;

        for(int i=0;i<randomNumThCun;i++){
            consumer MyConsumer = new consumer();
            for(int j=1;j<randomNumCun;j++){
                MyConsumer.run();
                ConCounter++;
            }
        }

        for(int i=0;i<randomNumThPro;i++){
           produser MyProducer = new produser();
           for(int j=1;j<randomNumPro;j++){
                MyProducer.run();
                ProCounter++;
           }
        }
        int activeThreadCount = Thread.activeCount();
        System.out.println("Number of active threads: " + activeThreadCount);
        System.out.println("the final value of x is: "+Data.x);
        System.out.println("the correct final value of x is: "+(1000+ProCounter-ConCounter));
    }

}
```

OSWithSync Class:

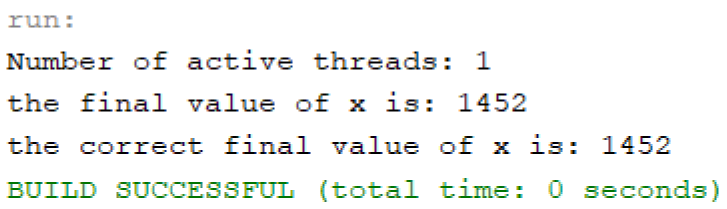
The diffraction between this Class and the previous one that we implemented the Semaphore method to made the synchronization with acquire and release methods which give us the correct final result for the varable X, with taking into consideration The condition on the variable on its value.



```
public class OSWithSync {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //initiate the producer,consumer counter
        int ConCounter = 0;
        int ProCounter = 0;

        Random random = new Random();
        int randomNumThPro = random.nextInt(10)+1;
        int randomNumThCun = random.nextInt(15)+1;

        Random random1 = new Random();
        int randomNumPro = random1.nextInt(900)+100;
        int randomNumCun = random1.nextInt(900) + 100;

        Semaphore semaphore = new Semaphore(15);

        int BUFFER_SIZE = 800;
        Semaphore full = new Semaphore(1000);//emaphore to track the number of full slots
        Semaphore empty = new Semaphore(BUFFER_SIZE);
        int[] buffer = new int[BUFFER_SIZE];

        for(int i=0;i<randomNumThPro;i++){
           produser MyProducer = new produser();
           for(int j=1;j<randomNumPro;j++){
               try {
                   if(ProCounter==800){
                       System.out.println("the value of buffer is reach to 1799 and make infinte loop");
                   }

                   empty.acquire();
                   semaphore.acquire();
               } catch (InterruptedException ex) {
                   Logger.getLogger(OSWithSync.class.getName()).log(Level.SEVERE, null, ex);
               }
                MyProducer.run();
                ProCounter++;
                int item = i;
                buffer[i % BUFFER_SIZE] = item;
                semaphore.release();
                full.release();
           }
        }

        for(int i = 0; i < randomNumThCun ; i++){
            consumer MyConsumer = new consumer();
            for(int j=1;j<randomNumCun;j++){
                try {
                   if(ConCounter==1000){
                       System.out.println("the value of buffer is reach to 0 and make infinte loop");
                   }
                    full.acquire();
                    semaphore.acquire();
                } catch (InterruptedException ex) {
                    Logger.getLogger(OSWithSync.class.getName()).log(Level.SEVERE, null, ex);
                }
                MyConsumer.run();
                int item = buffer[i % BUFFER_SIZE];
                ConCounter++;
                semaphore.release();
                empty.release();
            }
        }


        System.out.println("the final value of x with sync is: "+Data.x);

    }

}
```

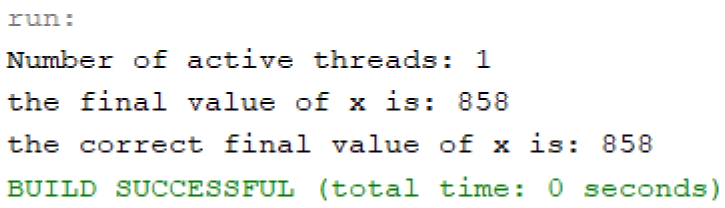
The Result of This Code (without sync)

Result 1:



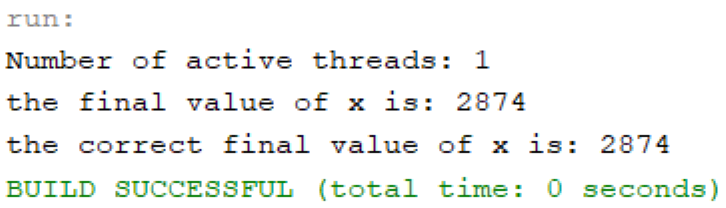
```
run:
Number of active threads: 1
the final value of x is: 1452
the correct final value of x is: 1452
BUILD SUCCESSFUL (total time: 0 seconds)
```

Result 2:



```
run:
Number of active threads: 1
the final value of x is: 858
the correct final value of x is: 858
BUILD SUCCESSFUL (total time: 0 seconds)
```

Result 3:

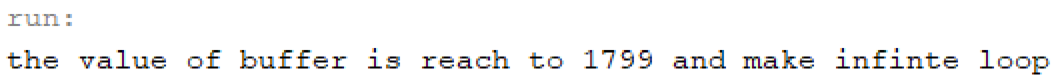


```
run:
Number of active threads: 1
the final value of x is: 2874
the correct final value of x is: 2874
BUILD SUCCESSFUL (total time: 0 seconds)
```

As we can see, the value of the shared data depends on the order of who accesses this data first (Actually it should be)

—

The Result of This Code (with sync)



```
run:
the value of buffer is reach to 1799 and make infinte loop
```

And it will stay the same for different number of threads because we have condition for the final value of X.

Thank You