Distributed Systems Lab
Car Hire Booking Distributed System

Team Members:
Tania Rajabally (2017130047)
Shruti Rampure(2017130048)
Rahul Ramteke(2017130049)

Batch: C

Problem Statement:
The purpose of this system is to monitor and control the bookings of cars in a distributed environment. All the features of a typical Car Hire Booking System are discussed here by considering a distributed system.

DATA REPLICATION

Data replication is the process in which the data is copied at multiple locations (Different computers or servers) to improve the availability of data.
Data replication is done with an aim to:

- Increase the availability of data.
- Speed up the query evaluation.

There are two types of data replication:

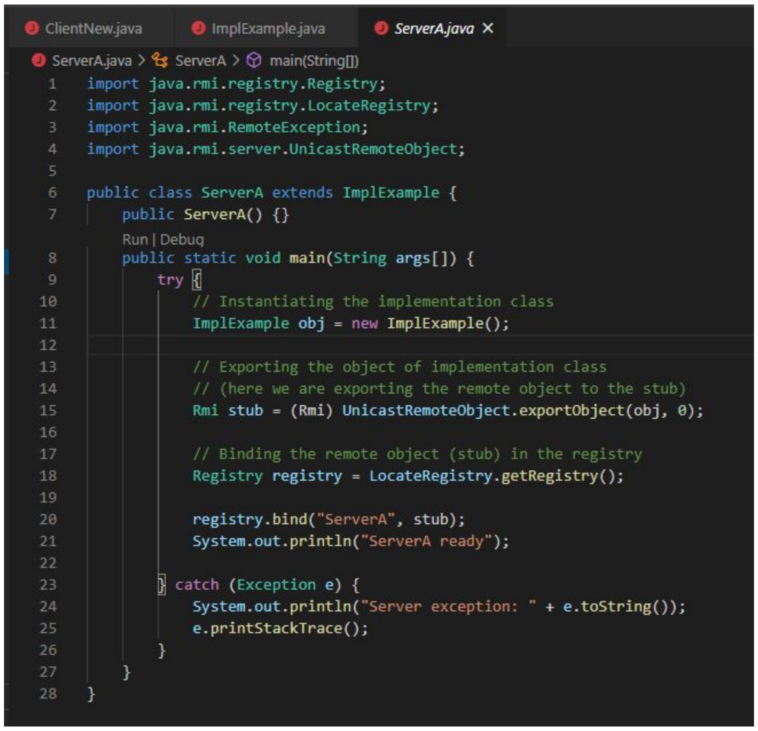
- Synchronous Replication:
  In synchronous replication, the replica will be modified immediately after some changes are made in the relation table. So there is no difference between original data and replica.
- Asynchronous replication:
  In asynchronous replication, the replica will be modified after commit is fired on to the database.

Data Replication is generally performed to:

- To provide a consistent copy of data across all the database nodes.
- To increase the availability of data.
- The reliability of data is increased through data replication.
- Data Replication supports multiple users and gives high performance.
- To remove any data redundancy,the databases are merged and slave databases are updated with outdated or incomplete data.
- Since replicas are created there are chances that the data is found itself where the transaction is executing which reduces the data movement.
- To perform faster execution of queries.

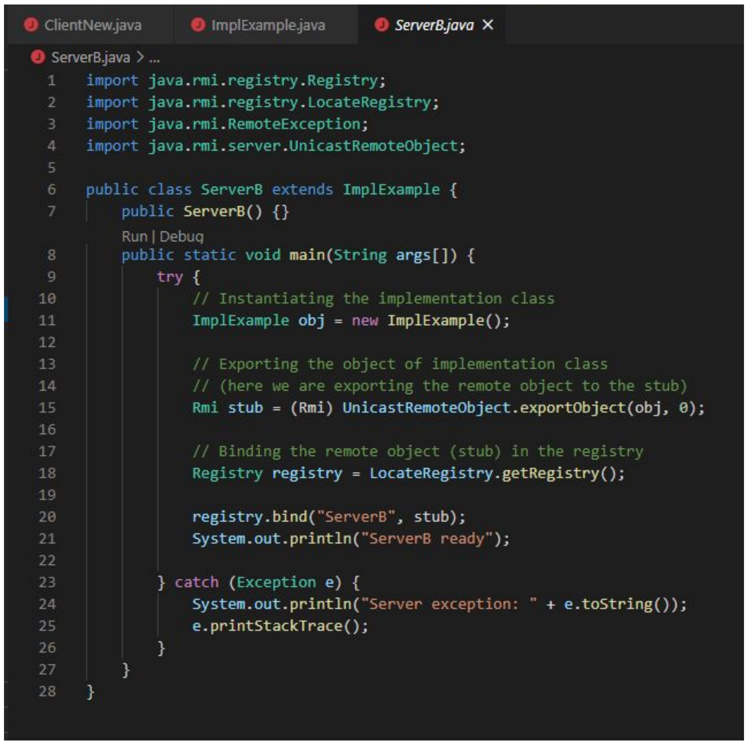
Code:

ServerA.java



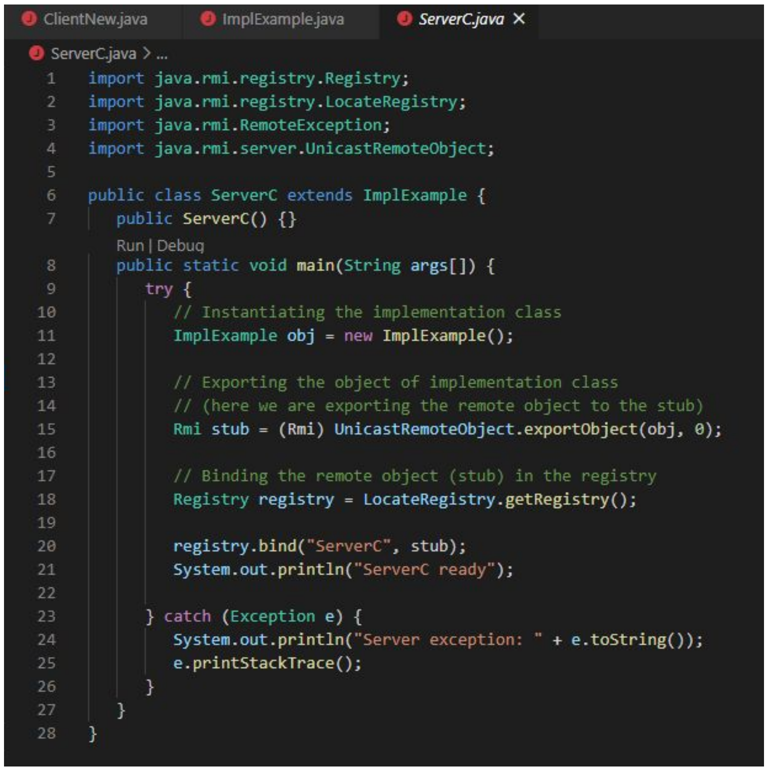
```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ServerA extends ImplExample {
    public ServerA() {}
    public static void main(String args[]) {
        try {
            // Instantiating the implementation class
            ImplExample obj = new ImplExample();

            // Exporting the object of implementation class
            // (here we are exporting the remote object to the stub)
            Rmi stub = (Rmi) UnicastRemoteObject.exportObject(obj, 0);

            // Binding the remote object (stub) in the registry
            Registry registry = LocateRegistry.getRegistry();

            registry.bind("ServerA", stub);
            System.out.println("ServerA ready");

        } catch (Exception e) {
            System.out.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

ServerB.java



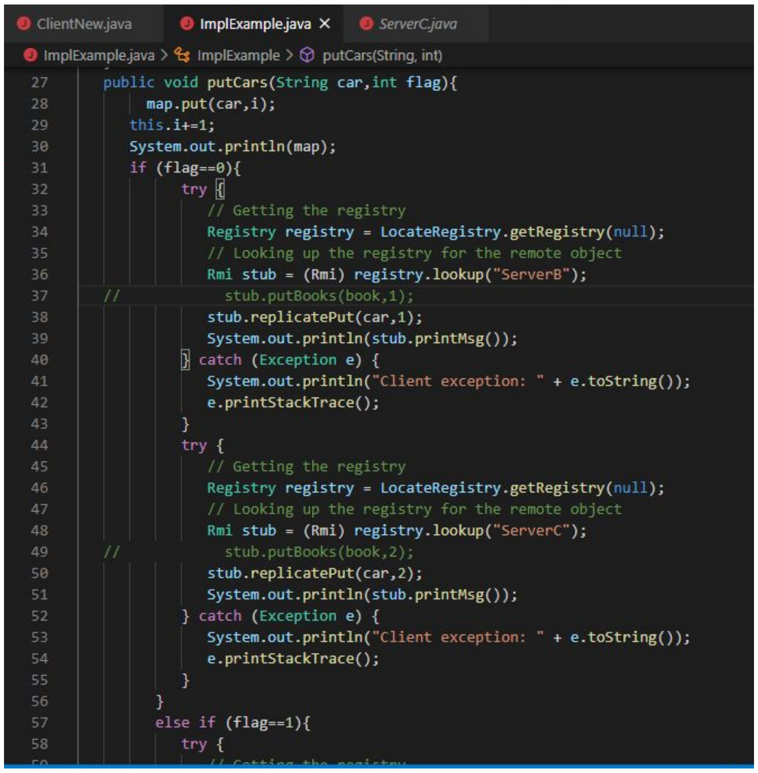
```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ServerB extends ImplExample {
    public ServerB() {}
    public static void main(String args[]) {
        try {
            // Instantiating the implementation class
            ImplExample obj = new ImplExample();

            // Exporting the object of implementation class
            // (here we are exporting the remote object to the stub)
            Rmi stub = (Rmi) UnicastRemoteObject.exportObject(obj, 0);

            // Binding the remote object (stub) in the registry
            Registry registry = LocateRegistry.getRegistry();

            registry.bind("ServerB", stub);
            System.out.println("ServerB ready");

        } catch (Exception e) {
            System.out.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

ServerC.java



```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ServerC extends ImplExample {
    public ServerC() {}

    public static void main(String args[]) {
        try {
            // Instantiating the implementation class
            ImplExample obj = new ImplExample();

            // Exporting the object of implementation class
            // (here we are exporting the remote object to the stub)
            Rmi stub = (Rmi) UnicastRemoteObject.exportObject(obj, 0);

            // Binding the remote object (stub) in the registry
            Registry registry = LocateRegistry.getRegistry();

            registry.bind("ServerC", stub);
            System.out.println("ServerC ready");

        } catch (Exception e) {
            System.out.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Implementation.java



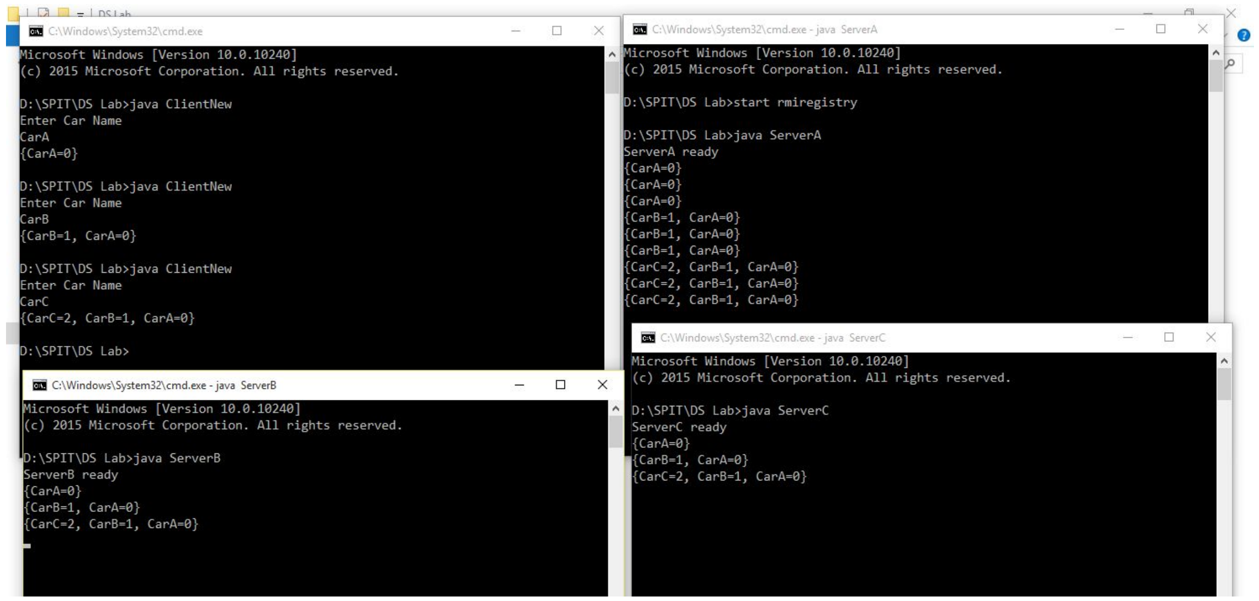
```java
public void putCars(String car,int flag){
     map.put(car,i);
   this.i+=1;
   System.out.println(map);
   if (flag==0){
         try {
            // Getting the registry
            Registry registry = LocateRegistry.getRegistry(null);
            // Looking up the registry for the remote object
            Rmi stub = (Rmi) registry.lookup("ServerB");
//                stub.putBooks(book,1);
            stub.replicatePut(car,1);
            System.out.println(stub.printMsg());
         } catch (Exception e) {
            System.out.println("Client exception: " + e.toString());
            e.printStackTrace();
         }
         try {
            // Getting the registry
            Registry registry = LocateRegistry.getRegistry(null);
            // Looking up the registry for the remote object
            Rmi stub = (Rmi) registry.lookup("ServerC");
//                stub.putBooks(book,2);
            stub.replicatePut(car,2);
            System.out.println(stub.printMsg());
         } catch (Exception e) {
            System.out.println("Client exception: " + e.toString());
            e.printStackTrace();
         }
      }
      else if (flag==1){
         try {
            // Getting the registry
```

Explanation of Implementation:
Here, we have created three servers for replication. ServerA is the main server. Everytime a change is made in serverA, we lookup for the other two servers and the respective changes will be made in ServerB and ServerC. In this way if one of the servers crashes, the data can be retrieved from the other servers and no data is lost. As we can see in the screenshots below, each server is started on a new window. Every time the client enters a new car name, it is updated in all the servers.
First, clientA is updated and then the hashmap in the other two servers are updated. In this way all the data is stored and replicas are available.

Steps to run:

1. Start rmiregistry
2. Compile the files : javac *.java
3. Run the command:
   java ServerA so that the serverA will be ready.
   In a new window run :
   Java ServerB so that serverB is ready.
   In a new window run :
   Java ServerC so that serverC is ready.
4. In a new window, run the command: java ClientNew to run the client side program. The client can then add new cars.

Screenshots:


C:\Windows\System32\cmd.exe
C:\Windows\System32\cmd.exe - java ServerA
C:\Windows\System32\cmd.exe - java ServerB
C:\Windows\System32\cmd.exe - java ServerC

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\DS Lab>java ClientNew
Enter Car Name
CarA
{CarA=0}

D:\SPIT\DS Lab>java ClientNew
Enter Car Name
CarB
{CarB=1, CarA=0}

D:\SPIT\DS Lab>java ClientNew
Enter Car Name
CarC
{CarC=2, CarB=1, CarA=0}

D:\SPIT\DS Lab>
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\DS Lab>start rmiregistry

D:\SPIT\DS Lab>java ServerA
ServerA ready
{CarA=0}
{CarA=0}
{CarA=0}
{CarB=1, CarA=0}
{CarB=1, CarA=0}
{CarB=1, CarA=0}
{CarC=2, CarB=1, CarA=0}
{CarC=2, CarB=1, CarA=0}
{CarC=2, CarB=1, CarA=0}
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\DS Lab>java ServerB
ServerB ready
{CarA=0}
{CarB=1, CarA=0}
{CarC=2, CarB=1, CarA=0}
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\DS Lab>java ServerC
ServerC ready
{CarA=0}
{CarB=1, CarA=0}
{CarC=2, CarB=1, CarA=0}
```

The data is getting replicated in all three servers.

Conclusion:
Data replication was implemented to increase the reliability and availability of the data. We implemented this via 3 servers which store the data. Thus our car booking system has 3 replication servers.