CLOUD COMPUTING PRACTICAL#02

OBJECTIVE: TO BECOME FAMILIAR WITH RMI OBJECT SERIALIZATION

REMOTE METHOD INVOCATION

- RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.

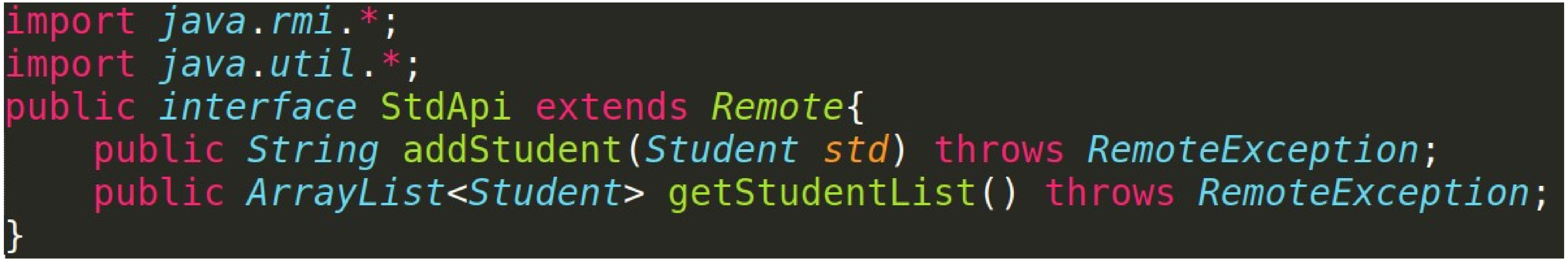
SERIALIZATION

- *Serialization* is the process of converting a set of object instances that contain references to each other into a linear stream of bytes, which can then be sent through a socket, stored to a file, or simply manipulated as a stream of data.
- Serialization is the mechanism used by RMI to pass objects between JVMs, either as arguments in a method invocation from a client to a server or as return values from a method invocation.

NOW WE WILL SEE CODE FOR PASSING OBJECTS THROUGH RMI

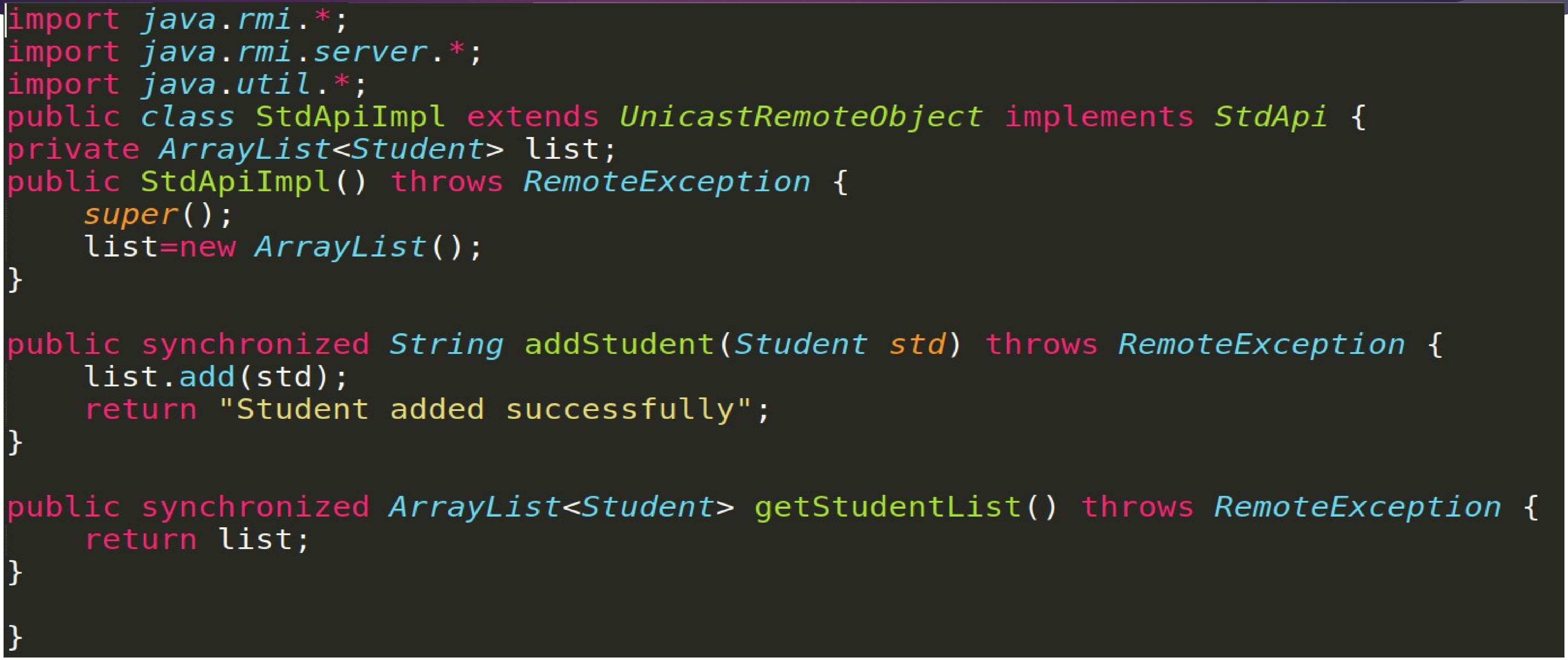
Our scenario will have a server sharing an object via RMI and a client calling the shared instance.

Interface



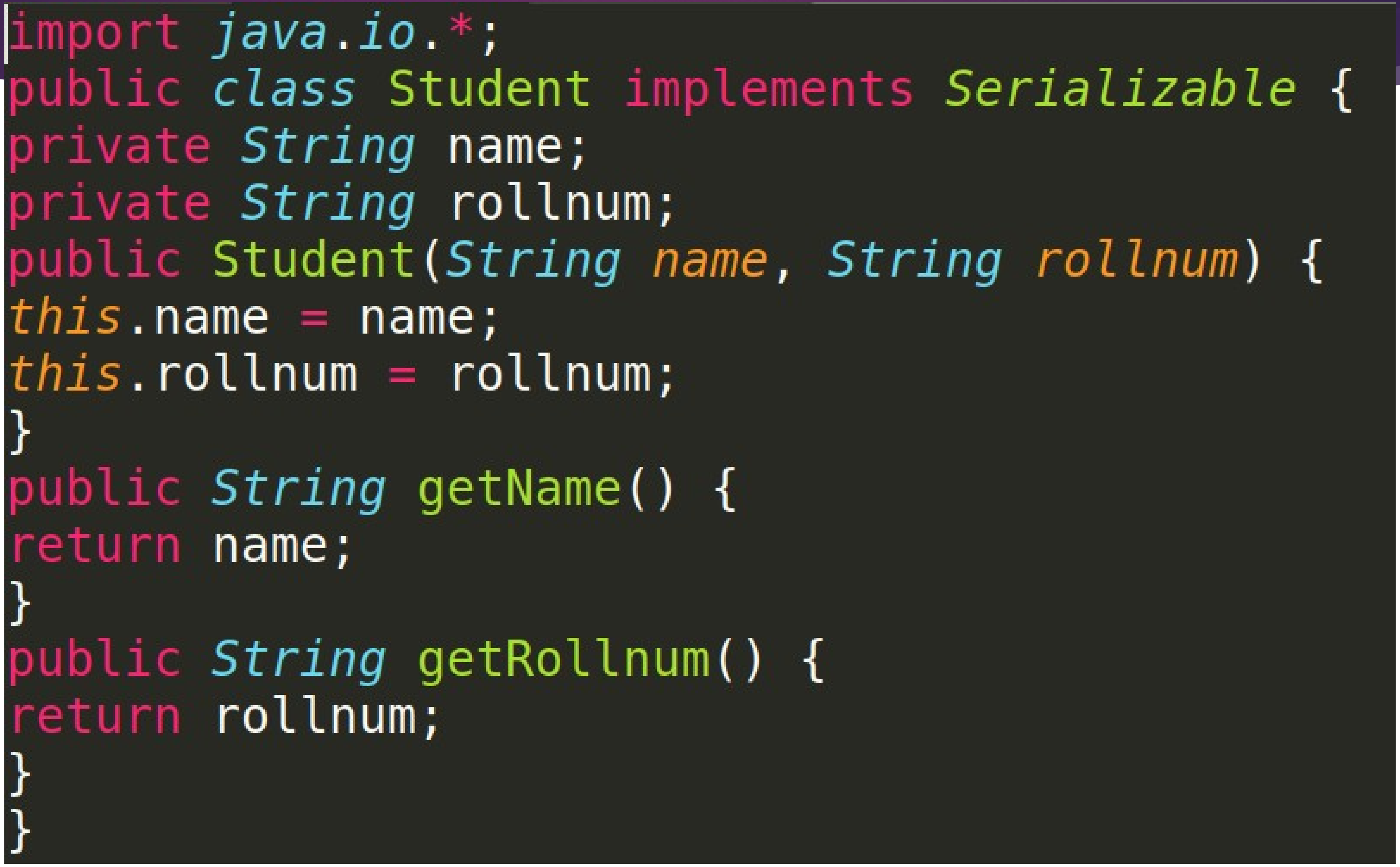
```java
import java.rmi.*;
import java.util.*;
public interface StdApi extends Remote{
    public String addStudent(Student std) throws RemoteException;
    public ArrayList<Student> getStudentList() throws RemoteException;
}
```

Remote Object Class



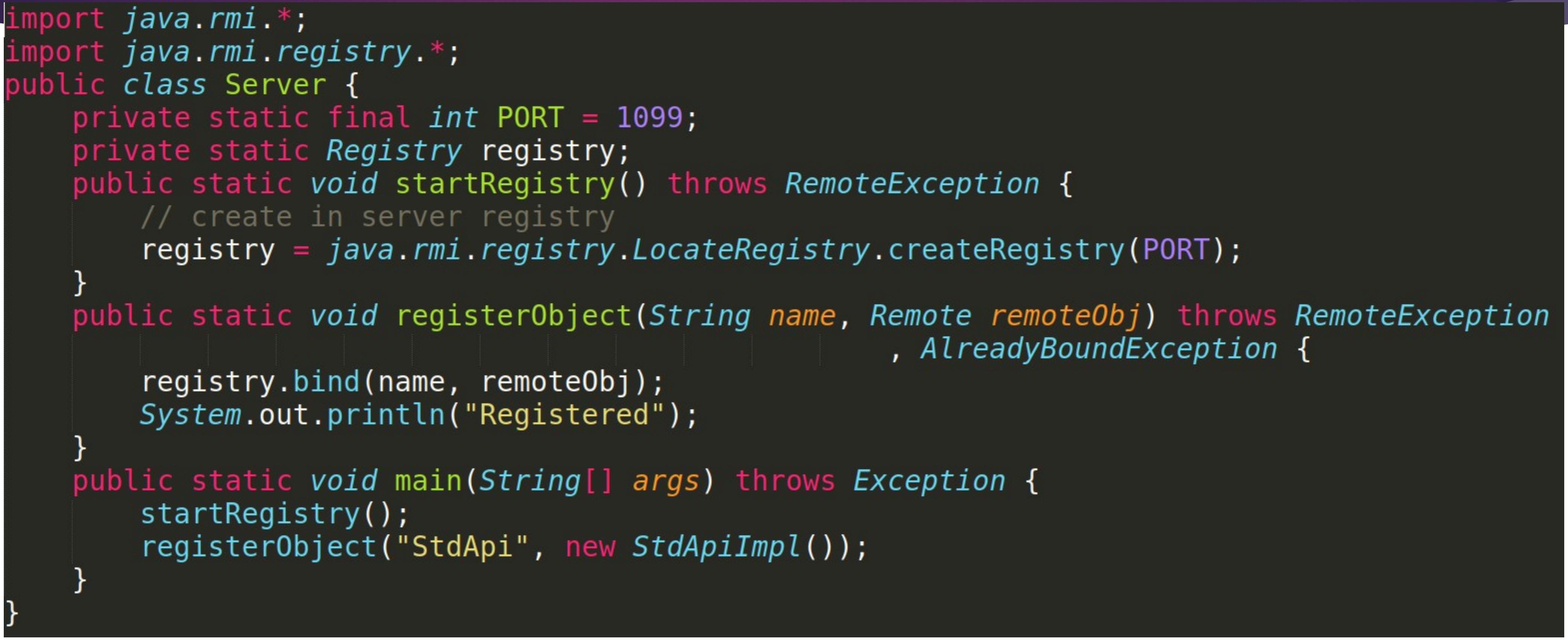
```java
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
public class StdApiImpl extends UnicastRemoteObject implements StdApi {
private ArrayList<Student> list;
public StdApiImpl() throws RemoteException {
    super();
    list=new ArrayList();
}

public synchronized String addStudent(Student std) throws RemoteException {
    list.add(std);
    return "Student added successfully";
}

public synchronized ArrayList<Student> getStudentList() throws RemoteException {
    return list;
}


}
```

Serializable Class



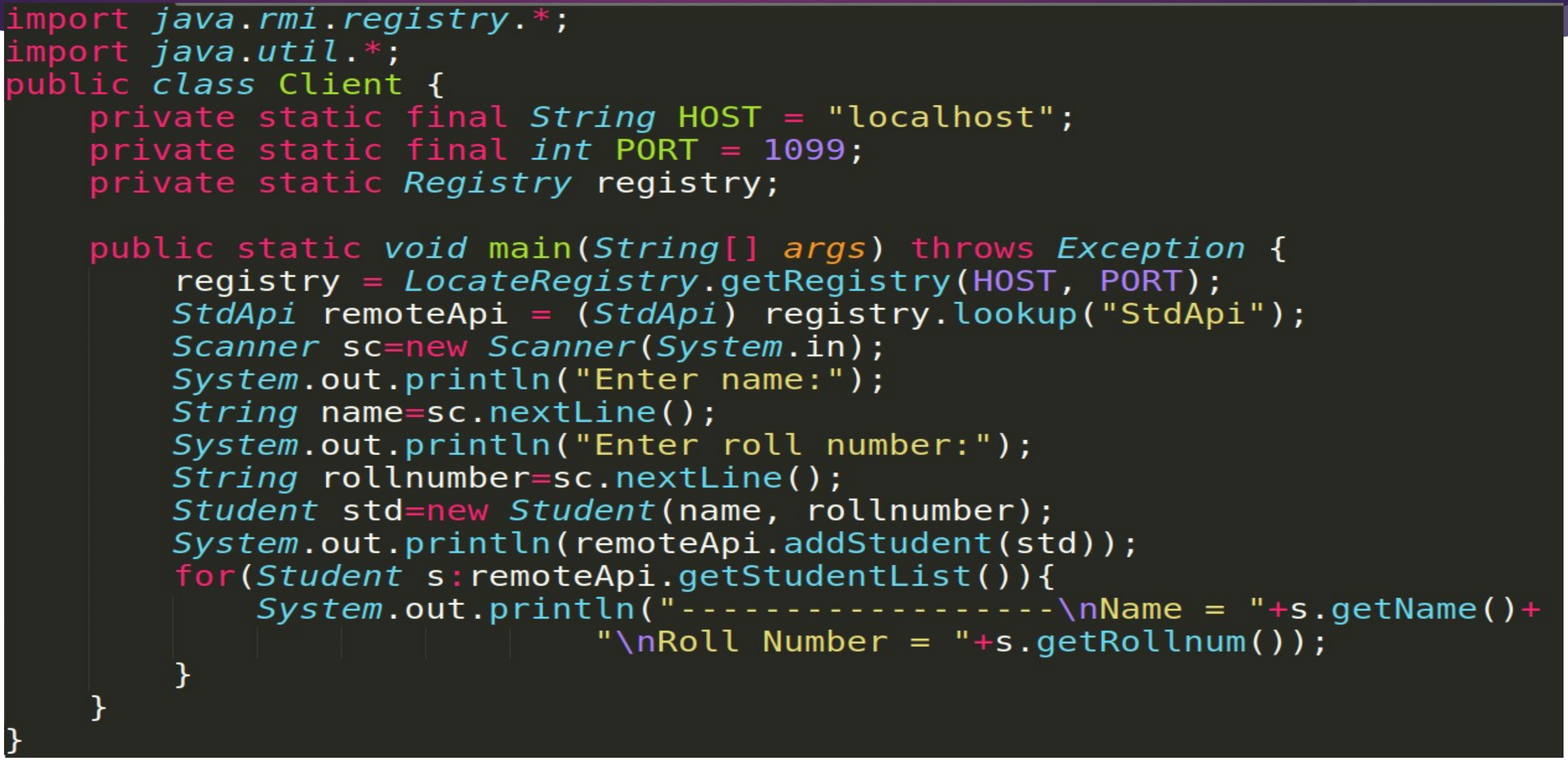
```
import java.io.*;
public class Student implements Serializable {
private String name;
private String rollnum;
public Student(String name, String rollnum) {
this.name = name;
this.rollnum = rollnum;
}
public String getName() {
return name;
}
public String getRollnum() {
return rollnum;
}
}
```

Server Class



```
import java.rmi.*;
import java.rmi.registry.*;
public class Server {
    private static final int PORT = 1099;
    private static Registry registry;
    public static void startRegistry() throws RemoteException {
        // create in server registry
        registry = java.rmi.registry.LocateRegistry.createRegistry(PORT);
    }
    public static void registerObject(String name, Remote remoteObj) throws RemoteException
                                                , AlreadyBoundException {
        registry.bind(name, remoteObj);
        System.out.println("Registered");
    }
    public static void main(String[] args) throws Exception {
        startRegistry();
        registerObject("StdApi", new StdApiImpl());
    }
}
```

Client Class



```
import java.rmi.registry.*;
import java.util.*;
public class Client {
    private static final String HOST = "localhost";
    private static final int PORT = 1099;
    private static Registry registry;

    public static void main(String[] args) throws Exception {
        registry = LocateRegistry.getRegistry(HOST, PORT);
        StdApi remoteApi = (StdApi) registry.lookup("StdApi");
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter name:");
        String name=sc.nextLine();
        System.out.println("Enter roll number:");
        String rollnumber=sc.nextLine();
        Student std=new Student(name, rollnumber);
        System.out.println(remoteApi.addStudent(std));
        for(Student s:remoteApi.getStudentList()){
            System.out.println("------------------\nName = "+s.getName()+
                                "\nRoll Number = "+s.getRollnum());
        }
    }
}
```

TASK:

- Develop any simple application to demonstrate object passing in RMI. (Expl: Make a class of your own and use its objects as arguments and return types for RMI)