University of New York Tirana
Faculty of Engineering and Architecture
Rruga e Kavajës, pranë 21 Dhjetorit (Sheshi Ataturk)
Tirane, Shqipëri

Master of Science in Computer Science

Distributed Systems
Manual for Laboratory Practice
PART I - Remote Method Invocation

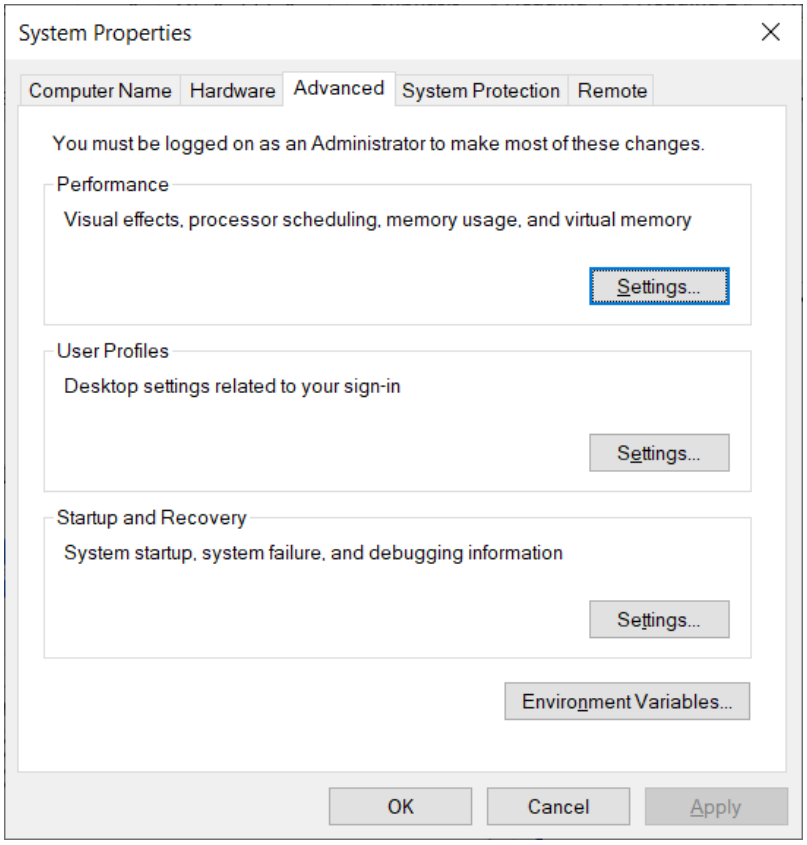
Prof. Dr. Marenglen Biba
Department of Computer Science
E-mail: marenglenbiba@unyt.edu.al

1. Document Purpose

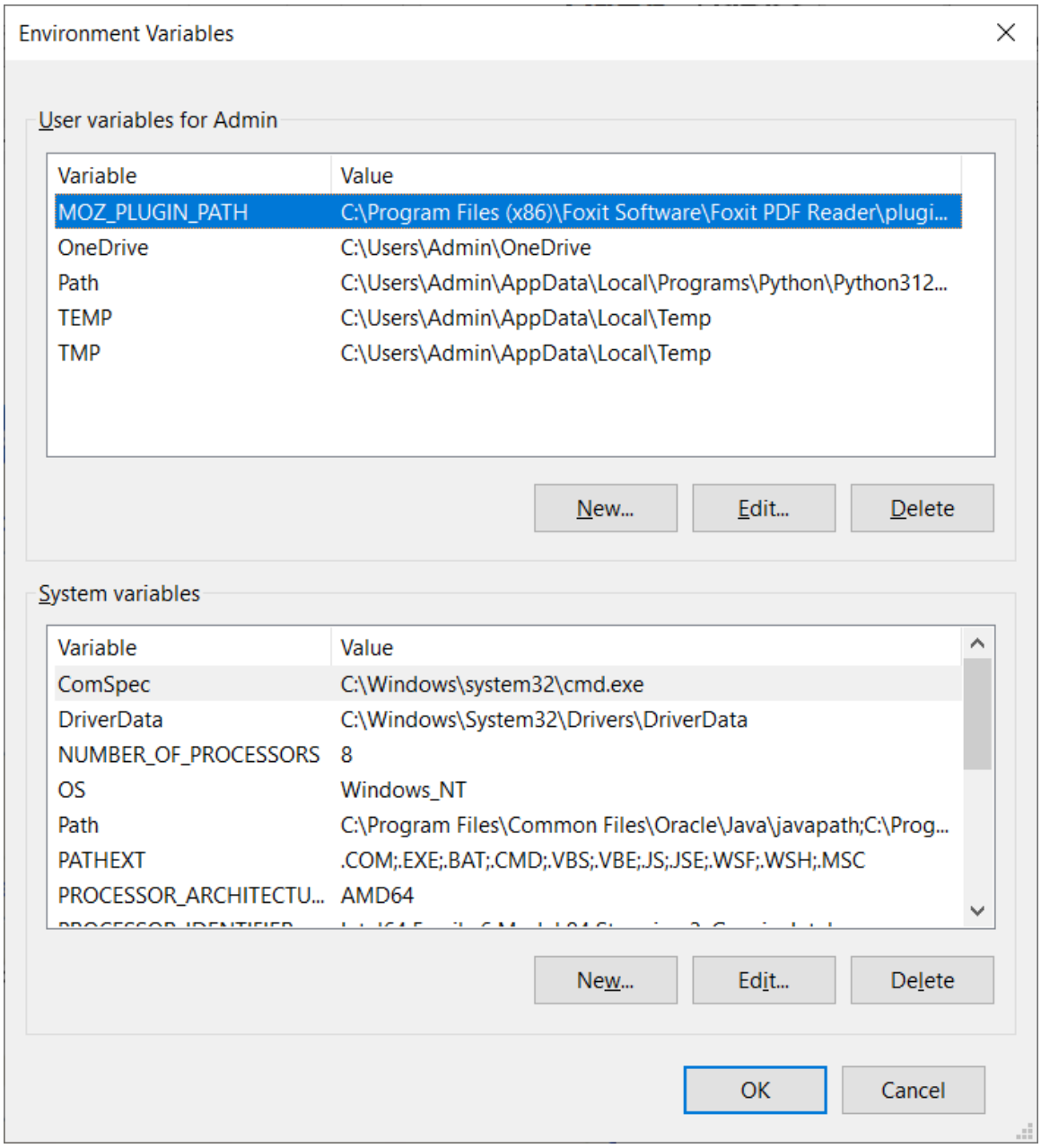
This document contains explanations on how to set up the required software programs and, how to write the lab exercises and how to execute these.

- Install Java SE, JDK 21
- Install Apache Netbeans 21
- Set path variables in the operating system

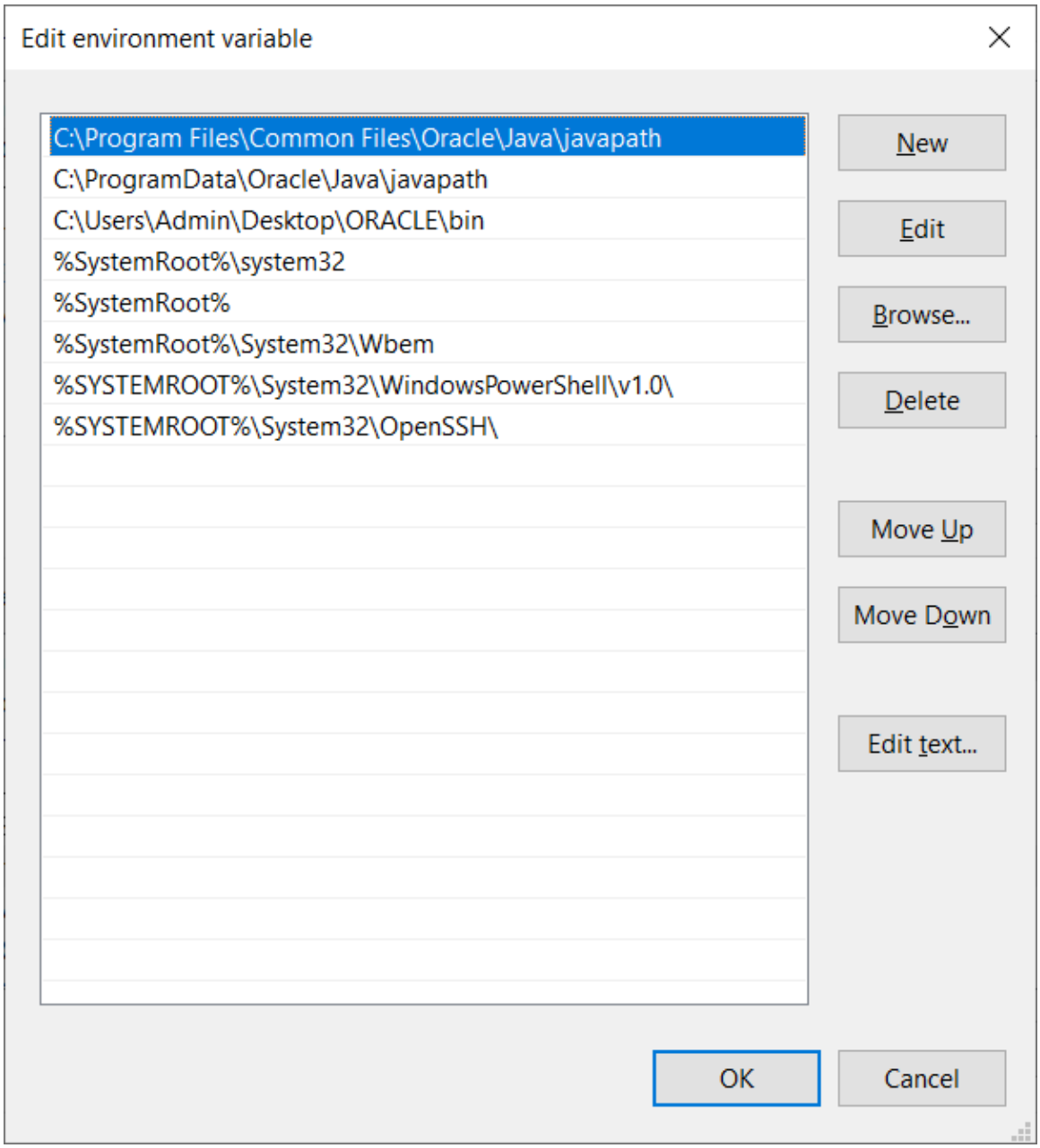
For running the programs, a correct configuration of the running environment is necessary (path variable).



Click on Environment Variables.



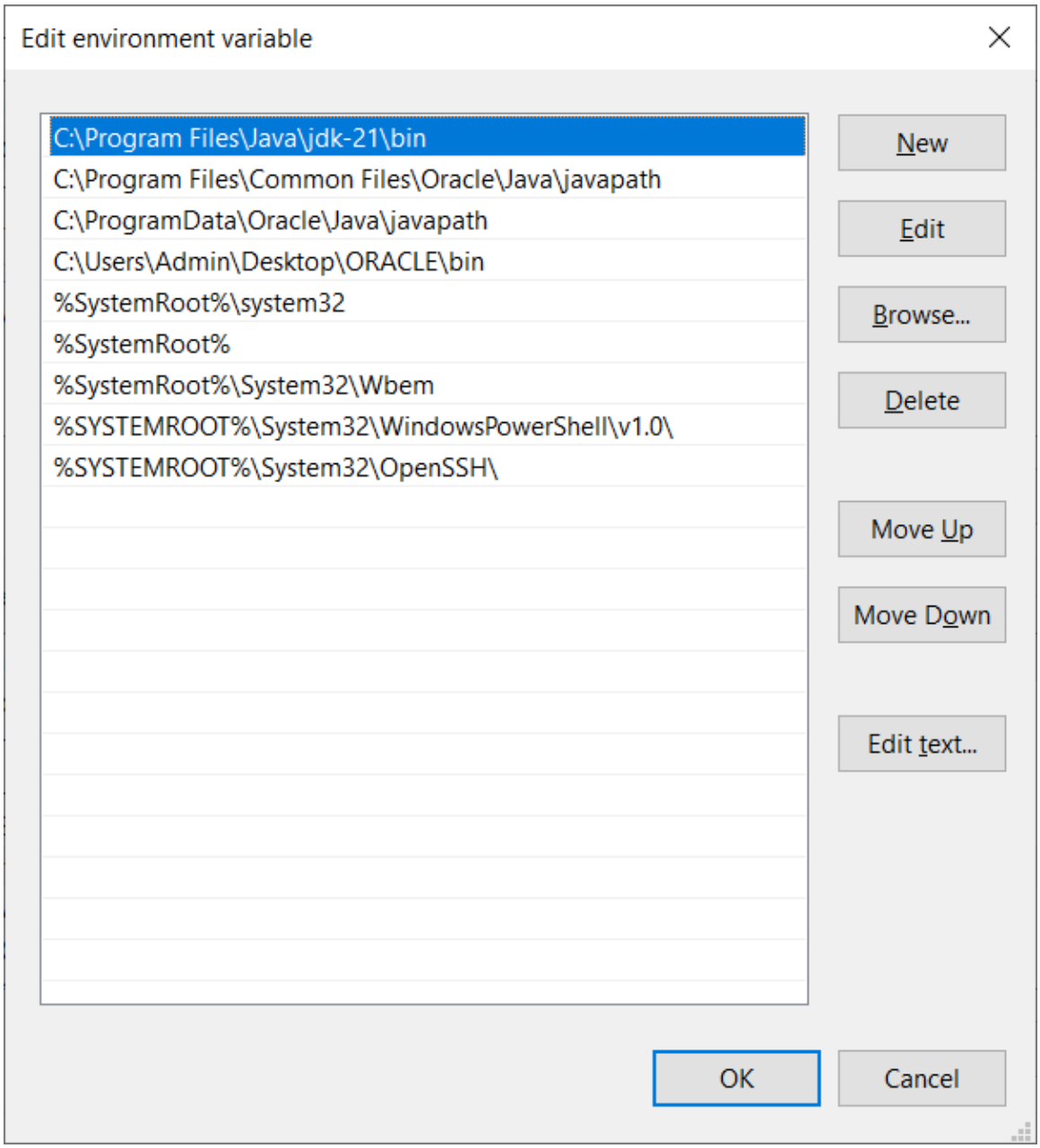
Find the Path system variable and click Edit.



Select New and set the value of the variable to the directory where you have installed Java, for example:

C:\Program Files\Java\jdk-21\bin

Move the item up as follows:



2. Developing the RMI Client and Server

When you finish this exercise, you will have run your first RMI system.

It consists of three major parts:

- The **RMI Registry** that hold references to the remote services.
- The **RMI host server** program that creates the remote services, registers them with the registry and waits for client requests.

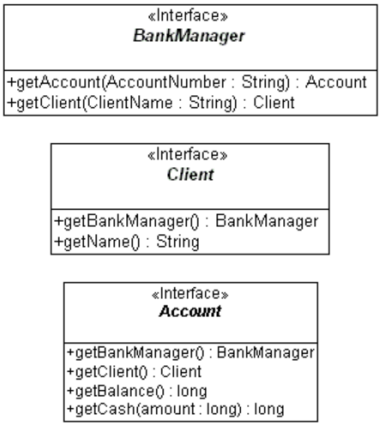
- The **RMI client program.** A program that obtains references to remote service object from the RMI registry and then uses those services.

2.1 Fundamentals of RMI

This exercise will introduce you to the definition of RMI remote services using Java interfaces.

Educational goals:

- Introduce the UML Description of a banking system
- Complete the Java source code for the system interfaces



These are the interfaces to develop in the project.

File Account.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Account extends Remote
   {
   // Add method to return master BankManager

   // Add method to return Client of this account

   // Add method to return balance of this account

   // Add method to withdraw cash from this account
   }
```

File BankManager.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;



public interface BankManager extends Remote
   {
   // Add method to return an Account service

   // Add method to return a Client service
   }
```

File Client.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Client extends Remote
   {
   // Add method to return master BankManager

   // Add method to return the name of this client
   }
```

2.2 Development of a Simple Banking System

In this part you will run your first RMI system. It is based on the Banking System that you started in the previous exercise.

Educational goals:

- Learn to run the RMI Registry as a separate process
- Run a server that support remote RMI objects
- Implement an RMI client that uses remote services.

The **RMI Registry** manages the publication of the RMI remote services. You have to run a server program that creates the actual remote services, and finally, finish coding the program BankUser, which will use the RMI remote services.

Step 1: Code Development

Develop the following code in Netbeans:

Account.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Account extends Remote {
  // Add method to return master BankManager
  public BankManager getBankManager()
      throws RemoteException;

  // Add method to return Client of this account
  public Client getClient()
      throws RemoteException;

  // Add method to return balance of this account
  public long getBalance()
      throws RemoteException;

  // Add method to withdraw cash from this account
  public long getCash (long amount)
      throws NoCashAvailableException, RemoteException;
}
```

BankManager.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BankManager extends Remote {

  // Add method to return an Account service
  public Account getAccount(String accountNumber)
      throws RemoteException;

  // Add method to return a Client service
  public Client getClient(String clientName)
      throws RemoteException;
}
```

Client.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Client extends Remote {

  // Add method to return master BankManager
  public BankManager getBankManager()
      throws RemoteException;

  // Add method to return the name of this client
  public String getName()
      throws RemoteException;
}
```

AccountImpl.java

```
import java.rmi.RemoteException;
import java.io.Serializable;

public class AccountImpl implements Account, Serializable {

  private BankManager bankManager;
  private Client      client;
  private long        balance;
  private String      accountNumber;

  // public constructor
  public AccountImpl (
     BankManager bankManager,
     Client client,
     String accountNumber) {
   this.bankManager = bankManager;
   this.client      = client;
   this.balance     = 0;
   this.accountNumber = accountNumber;
  }

  public void deposit(long amount) {
   balance += amount;
  }

  public BankManager getBankManager()
```

```
    throws RemoteException {
  return bankManager;
 }

 public Client getClient()
    throws RemoteException {
  return client;
 }

 public long getBalance()
    throws RemoteException {
  return balance;
 }

 public long getCash(long amount)
    throws NoCashAvailableException, RemoteException {
  if (amount > balance) {
    throw new NoCashAvailableException();
  }
  balance = balance - amount;
  return amount;
 }
}
```

BankManagerImpl.java

```
import java.io.Serializable;
import java.util.HashMap;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class BankManagerImpl implements BankManager, Serializable {
 private HashMap accounts = new HashMap(20);;
 private HashMap clients = new HashMap(10);;

 // public No-argument constructor
 public BankManagerImpl()
    throws java.rmi.RemoteException {
  initialize();
 }

 @Override
 public Account getAccount(String accountNumber)
    throws RemoteException {
  AccountImpl account = (AccountImpl)accounts.get(accountNumber);
  return account;
```

```
  }

  @Override
  public Client getClient(String clientName)
      throws RemoteException {
    ClientImpl client = (ClientImpl)clients.get(clientName);
    return client;
  }

  public void initialize()
      throws java.rmi.RemoteException {
    Client clientCharlie = new ClientImpl(this, "Charlie");
    Client clientShannon = new ClientImpl(this, "Shannon");
    clients.put("Charlie", clientCharlie);
    clients.put("Shannon", clientShannon);
//
//    // Create accounts:
//    //     * put them into the hashtable
//    //     * associate them with clients
    Account account;
    account = new AccountImpl(this, clientCharlie, "4434");
    ((AccountImpl)account).deposit(500);
    accounts.put("4434", account);
    account = new AccountImpl(this, clientCharlie, "4461");
    ((AccountImpl)account).deposit(600);
    accounts.put("4461", account);
    account = new AccountImpl(this, clientShannon, "6678");
    ((AccountImpl)account).deposit(700);
    accounts.put("6678", account);
  }
}
```

BankUser.java

```
import java.rmi.*;
import java.net.MalformedURLException;
import java.util.Locale;
import java.text.NumberFormat;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class BankUser {

  public static void main(String[] args) {

    try {
    Registry registry = LocateRegistry.getRegistry("localhost", 1099);
```

```
    BankManager bm = (BankManager) registry.lookup("BankManagerImpl");


      // Lookup account 4461
      Account account = bm.getAccount("4461");
      Client client = account.getClient();
      String name = client.getName();
      long cash = account.getBalance();
      NumberFormat currencyFormat =
        NumberFormat.getCurrencyInstance(Locale.US);
      String balanceString = currencyFormat.format(cash);
      System.out.println(name + "'s account has " + balanceString);

    } catch (NotBoundException notBoundException) {
      System.err.println("Not Bound: " + notBoundException);
     } catch (RemoteException remoteException) {
      System.err.println("Remote Exception: " + remoteException);
     }
   }
}
```

ClientImpl.java

```
import java.rmi.RemoteException;
import java.io.Serializable;

public class ClientImpl implements Client, Serializable {

  private BankManager bankManager;
  private String      clientName;

  // public constructor
  public ClientImpl(BankManager bm, String name) {
    this.bankManager = bm;
    this.clientName  = name;
  }

  public BankManager getBankManager()
      throws RemoteException {
    return bankManager;
  }

  public String getName()
      throws RemoteException {
    return clientName;
```

```
  }
}
```

NoCashAvailableException.java

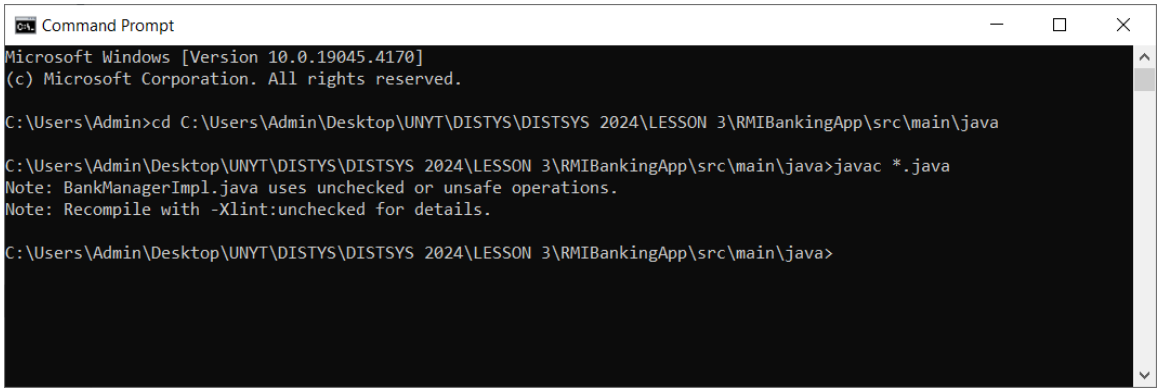
```
public class NoCashAvailableException extends Exception {
}
```

BankSystemServer.java

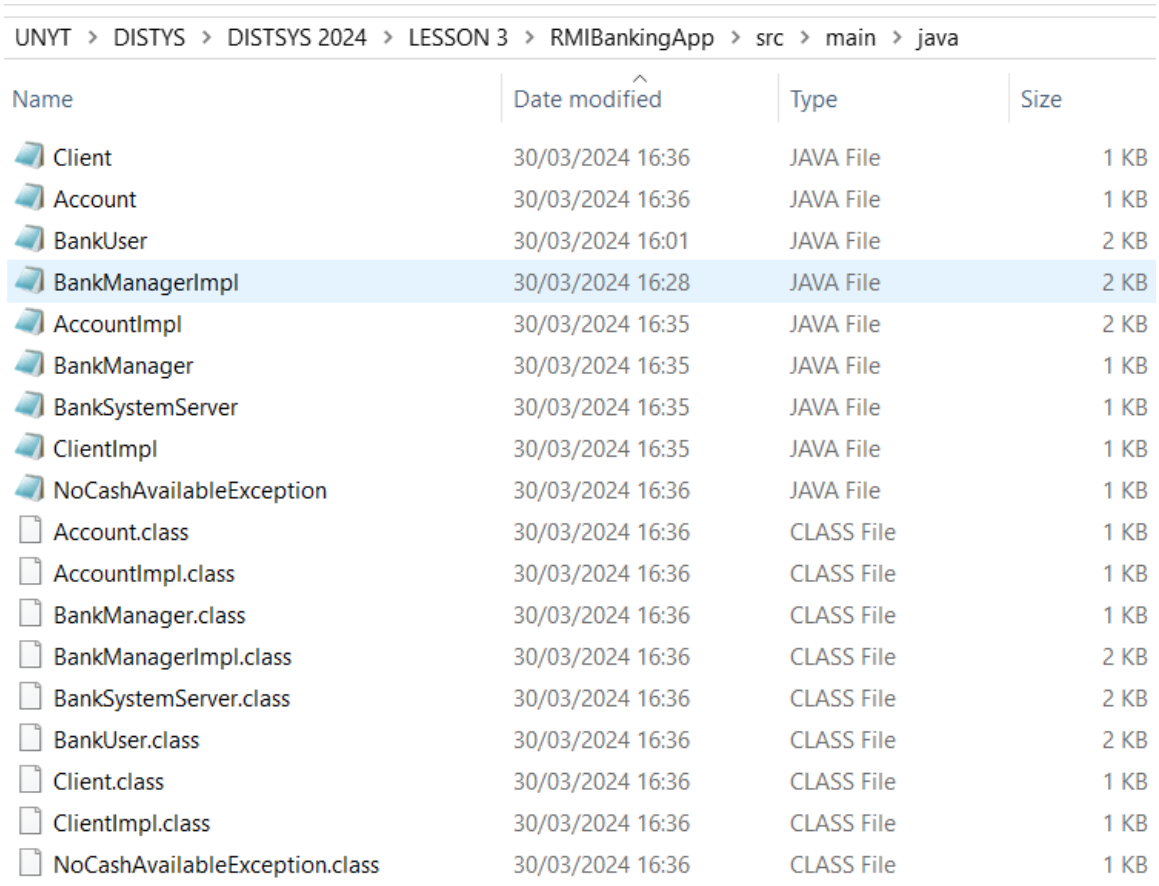
```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class BankSystemServer {

  public static void main(String args[]) {

    try {

       Registry registry = LocateRegistry.createRegistry(1099);
       registry.rebind("BankManagerImpl", new BankManagerImpl());

    } catch (RemoteException remoteException) {
      System.err.println(
        "Failure during object export to RMI: " +
         remoteException);
    }

    System.out.println("Server started.");
    System.out.println("Enter <CR> to end.");
    try {
       int i = System.in.read();
    } catch (Exception exception) {
    }
    System.exit(0);
  }
}
```

Step 2: Compile and run the code following these steps:

All the steps should be performed on the same directory where you have your .java files.



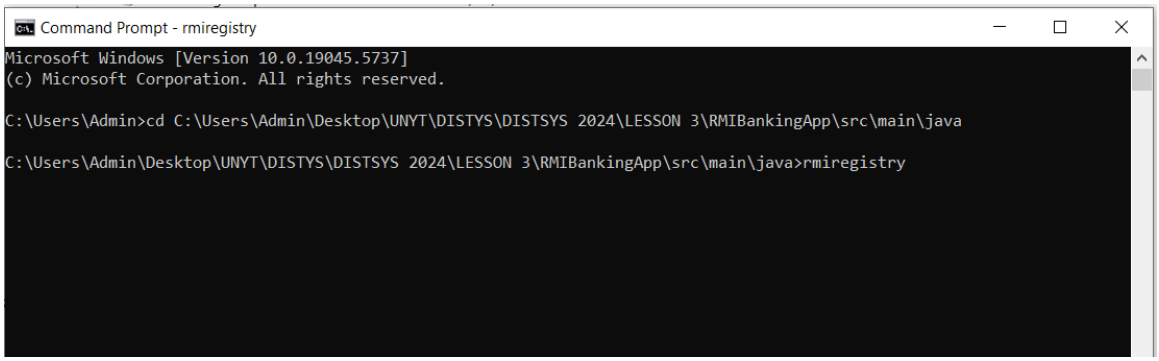
Observe the content of the folder. You will find all the .class files.



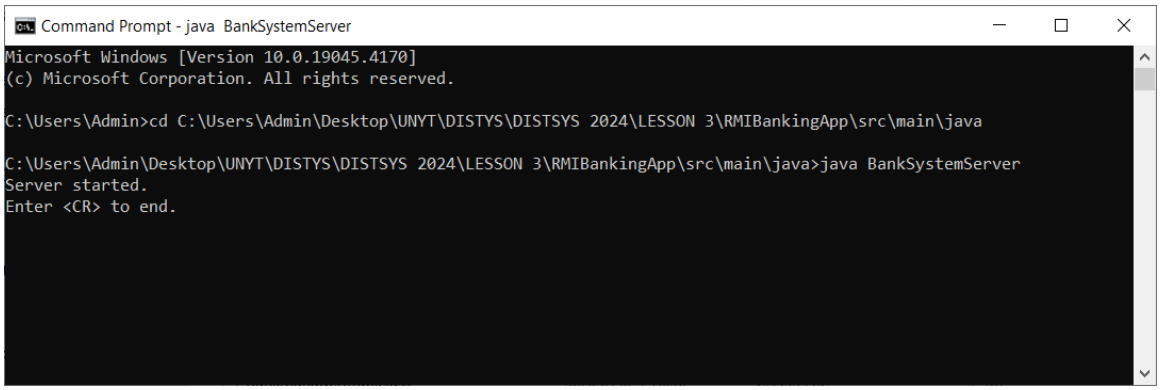
3. Run the server and the client

3.1 Run the server RMI Registry program.

Open a command line and move to the directory that will contain your code from this exercise.



From the same location, run the server with the following command:

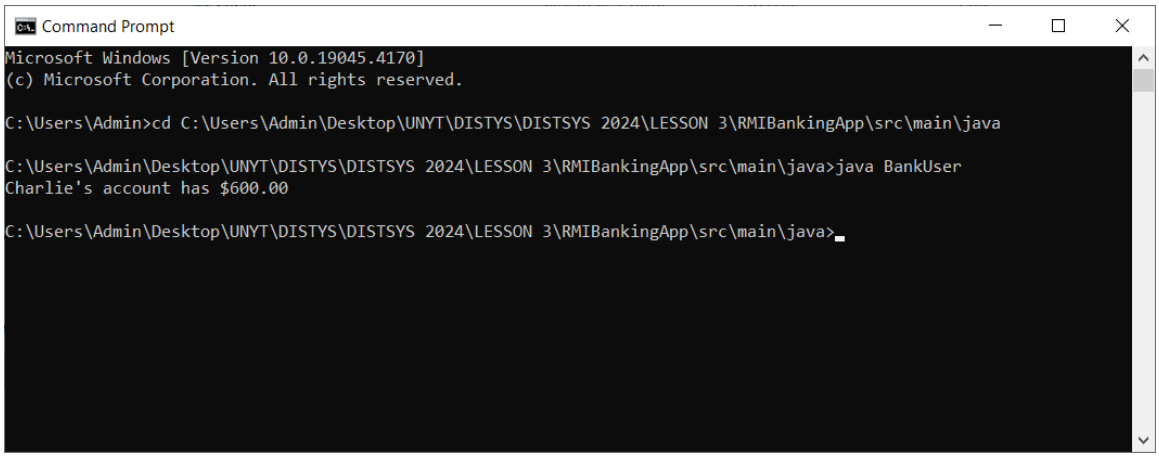


Do not close this window!

3.2 Run a program that will use the exported RMI services.

You simply need to start the client program, BankUser.

This is accomplished by opening a new command line, moving to the directory which containts your code from this exercise and running the following program: java BankUser.


Command Prompt
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Admin>cd C:\Users\Admin\Desktop\UNYT\DISTYS\DISTSYS 2024\LESSON 3\RMIBankingApp\src\main\java
C:\Users\Admin\Desktop\UNYT\DISTYS\DISTSYS 2024\LESSON 3\RMIBankingApp\src\main\java>java BankUser
Charlie's account has $600.00
C:\Users\Admin\Desktop\UNYT\DISTYS\DISTSYS 2024\LESSON 3\RMIBankingApp\src\main\java>

Notes

This tutorial was prepared based on the following resource:

https://www.oracle.com/java/technologies/