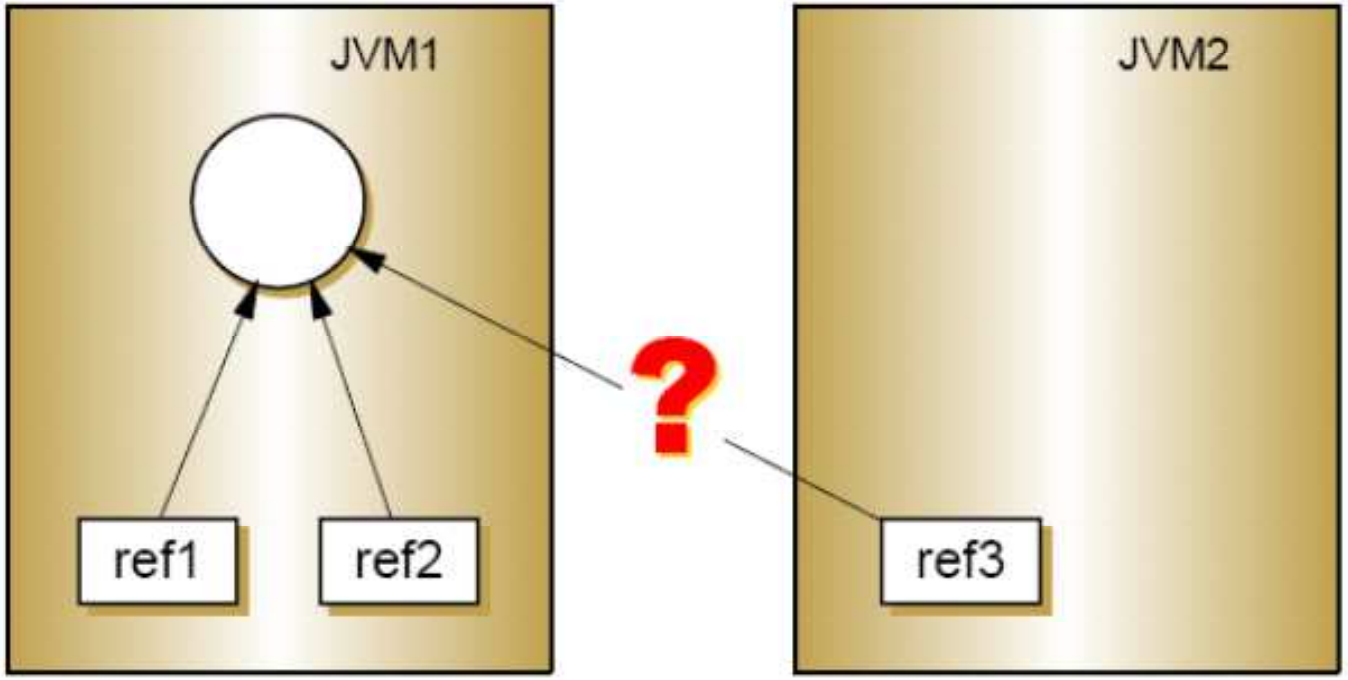
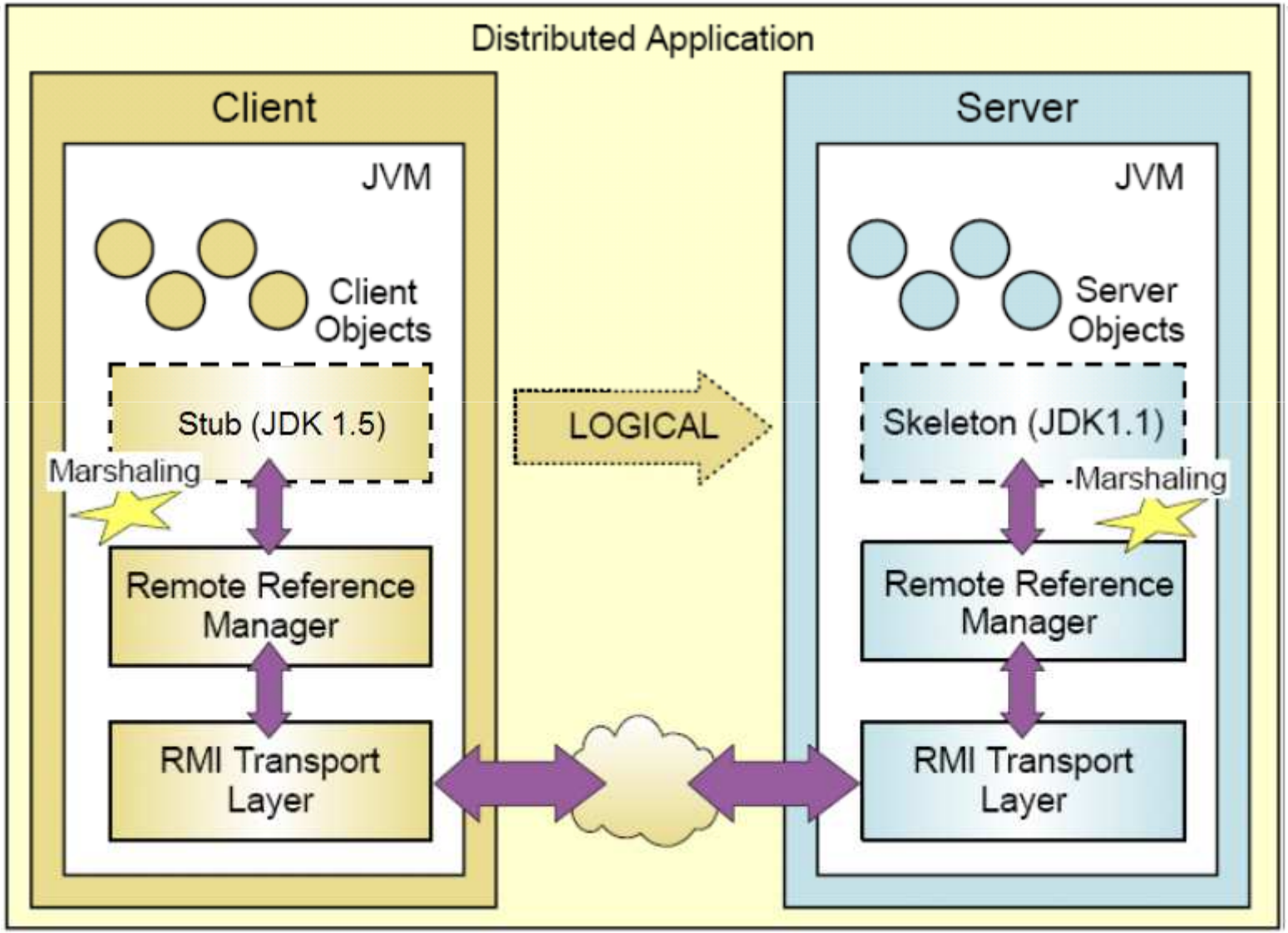
Java - RMI

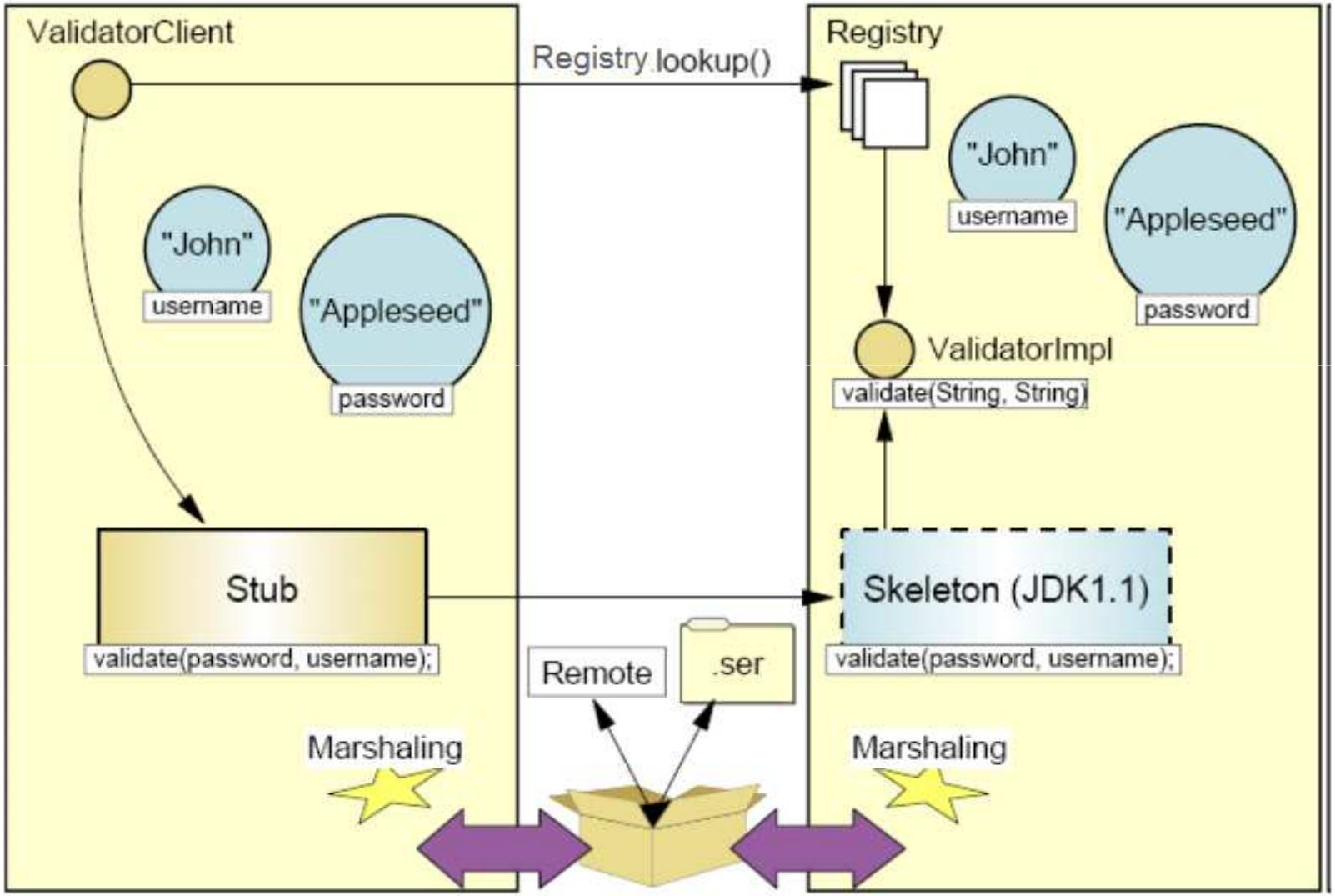
Промена парадигме



RMI


Distributed Application
Client
JVM
Client Objects
Stub (JDK 1.5)
Marshaling
Remote Reference Manager
RMI Transport Layer
LOGICAL
Server
JVM
Server Objects
Skeleton (JDK1.1)
Marshaling
Remote Reference Manager
RMI Transport Layer

Комуникација са удаљеним објектом



Корак 1: Креирање интерфејса

За све објекте којима се приступати удаљено мора се креирати интерфејс преко кога им се приступа.

Интерфејс мора да:

- проширује интерфејс Remote
- свака методе декларише да емитује RemoteException

Корак 1: Креирање интерфејса

```
import java.rmi.*;

public interface Validator extends Remote {
    String validate(String username, String password)
            throws RemoteException;
}
```

Корак 2: Имплементација интерфејса

Да би неком објекту могло удаљено да се приступа минимално је потребно:

- Декларисати да се имплементира удаљени интерфејс
- Дефинисати конструктор за удаљени објекат
- Направити имплементације за све методе из удаљеног интерфејса

Корак 2: Имплементација интерфејса

```
import java.util.*;

public class ValidatorImpl implements Validator {
    Map<String, String> memberMap;

    public ValidatorImpl() {
        memberMap = new HashMap<String, String>();
        memberMap.put("John", convert("JohnsPassword"));
    }

    public String validate(String username, String password) {
        if (getMemberMap().containsKey(username)
        && getMemberMap().get(username).equals(convert(password)))
            return "Welcome: " + username;
        return "Invalid username or password";
    }

    public Map<String, String> getMemberMap() {
        return memberMap;
    }
}
```

Корак 3: Упис удаљеног објекта

Како би неки објекат постао доступан за удаљени приступ потребно је урадити следеће:

- Креирање и инсталација сигурносне контроле (security manager)
- Креирање и извоз удаљених објеката
- Регистрација удаљеног објекта у одговарајући регистар

Корак 3: Упис удаљеног објекта

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
public class LoginServer {
    public static int port = 8080;
    public static void main(String args[]) {
        try {
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());
            }
            Validator validator = new ValidatorImpl();
            Validator stub = (Validator)
                    UnicastRemoteObject.exportObject(validator, 0);
            String urlString = "/Validator";
            Registry registry = LocateRegistry.createRegistry(port);
//          Registry registry = LocateRegistry.getRegistry(port);
            registry.rebind(urlString, stub);
            System.out.println("Login server is on");
        } catch (RemoteException e) {
                    e.printStackTrace();
        }
    }
}
```

Корак 3: UnicastRemoteObject

Methods

Modifier and Type	Method and Description
`Object`	`clone()` Returns a clone of the remote object that is distinct from the original.
`static RemoteStub`	`exportObject(Remote obj)` Exports the remote object to make it available to receive incoming calls using an anonymous port.
`static Remote`	`exportObject(Remote obj, int port)` Exports the remote object to make it available to receive incoming calls, using the particular supplied port.
`static Remote`	`exportObject(Remote obj, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf)` Exports the remote object to make it available to receive incoming calls, using a transport specified by the given socket factory.
`static boolean`	`unexportObject(Remote obj, boolean force)` Removes the remote object, obj, from the RMI runtime.

Корак 3: LocateRegistry

Methods

Modifier and Type	Method and Description
static Registry	**createRegistry**(int port) Creates and exports a Registry instance on the local host that accepts requests on the specified port.
static Registry	**createRegistry**(int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) Creates and exports a Registry instance on the local host that uses custom socket factories for communication with that instance.
static Registry	**getRegistry**() Returns a reference to the the remote object Registry for the local host on the default registry port of 1099.
static Registry	**getRegistry**(int port) Returns a reference to the the remote object Registry for the local host on the specified port.
static Registry	**getRegistry**(String host) Returns a reference to the remote object Registry on the specified host on the default registry port of 1099.
static Registry	**getRegistry**(String host, int port) Returns a reference to the remote object Registry on the specified host and port.
static Registry	**getRegistry**(String host, int port, RMIClientSocketFactory csf) Returns a locally created remote reference to the remote object Registry on the specified host and port.

Корак 3: Registry

Methods

Modifier and Type	Method and Description
void	**bind**(String name, Remote obj) Binds a remote reference to the specified name in this registry.
String[]	**list**() Returns an array of the names bound in this registry.
Remote	**lookup**(String name) Returns the remote reference bound to the specified name in this registry.
void	**rebind**(String name, Remote obj) Replaces the binding for the specified name in this registry with the supplied remote reference.
void	**unbind**(String name) Removes the binding for the specified name in this registry.

Корак 4: Креирање клијента

Како би се приступило неком удаљеном објекту који је уписан у регистар доступних објеката потребно је урадити следеће:

- Креирање и инсталација сигурносне контроле (security manager)
- Приступити регистру у коме је регистрован удаљени објекат
- На основу задатог имена из регистра дохватити удаљену референцу објеката

Корак 4: Креирање клијента

```
import java.rmi.registry.*;
public class LoginClient {
    public static void main(String args[]) {
        try {
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());
            }
            String host = args[0];
            int port = Integer.parseInt(args[1]);
            String urlString = "/Validator";
            Registry registry = LocateRegistry.getRegistry(host, port);
            Validator reply = (Validator) registry.lookup(urlString);
            System.out.println(reply.validate(args[2], args[3]));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Привилегије

java.home/lib/security/java.security (Solaris/Linux)
java.home\lib\security\java.security (Windows)

Подразумевана датотека са привилегијама

Коме се привилегија додељује

Која се привилегија додељује

grant
signedBy "*signer_names*", **codeBase** "*URL*",
principal *principal_class_name* "*principal_name*",
principal *principal_class_name* "*principal_name*", ...
{

permission *permission_class_name* "*target_name*", "*action*",
signedBy "*signer_names*";
permission *permission_class_name* "*target_name*", "*action*",
signedBy "*signer_names*"; ...

};

Структура датотеке са привилегијама

Привилегије

java.policy ← Пример имена датотеке са привилегијама

```
grant {
    // Allow everything for now
    permission java.security.AllPermission;
};
```

Пример датотеке са привилегијама

Коришћење

За верзије јаве пре верзије 1.5 је неопходно посебно превести дељене класе: **rmic** ValidatorImpl

Покретање:

Уколико је на серверској страни приступ регистру био са getRegistry потребно је покренути посебан програм који ради као регистар: **rmiregistry [-J<runtime flag>][*port*]**

Апликација:

java -Djava.security.policy=path_to_java.policy_file [-Djava.rmi.server.codebase=file:\path/] Class_With_Main

-Djava.security.policy=path_to_java.policy_file даје путању до фајла са за коришћену полису,

-Djava.rmi.server.codebase=file:\path/ даје путању до места где се налазе дељене класе којима приступа регистар, подешава се само ако се регистар независно покреће

Коришћење

Покретање:

Уколико је на серверској страни приступ регистру био са getRegistry потребно је покренути посебан програм:
rmiregistry -J-Djava.rmi.server.useCodebaseOnly=false

Серверска страна:

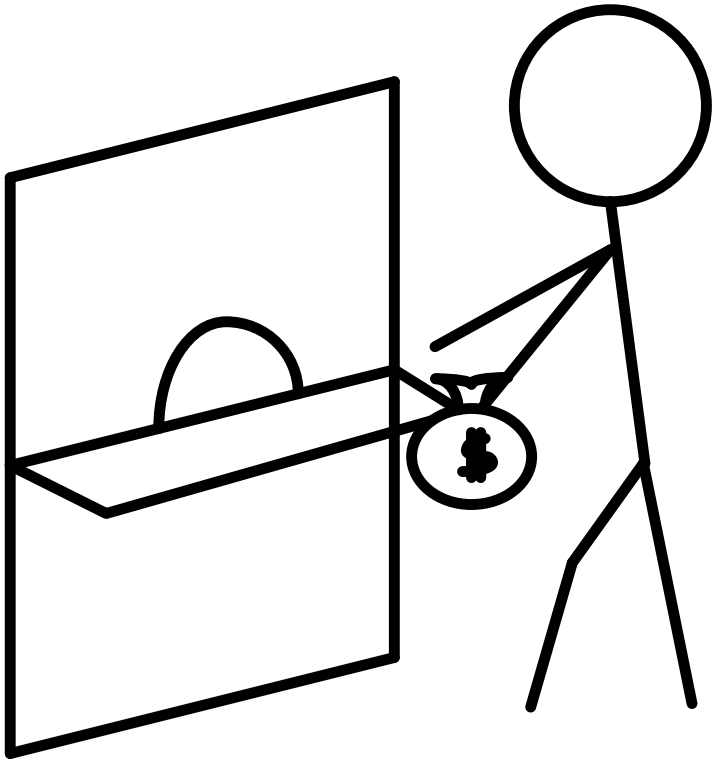
java -Djava.security.policy=java.policy
-Djava.rmi.server.codebase=file:\path/ LoginServer

Клијентска страна:

java -Djava.security.policy=java.policy
-Djava.rmi.server.codebase=file:\path/ LoginClient host port username password

Задаци

The Savings Account problem



The Savings Account problem

Рачун у банци може да дели више корисника. Сваки корисник може да уплаћује и подиже новац са рачуна под условом да салдо на рачуну никада не буде негативан, као и да види тренутно стање рачуна. Уколико нема новца на рачуну корисник чека док се новац не уплати на рачун. Решити проблем користећи удаљене позиве метода у Јави.

The Savings Account problem

```
import java.rmi.*;

public interface Bank extends Remote {

    public UserAccount getUserAccount(String name)
            throws RemoteException;
}

import java.rmi.*;

public interface UserAccount extends Remote {

    public float getStatus() throws RemoteException;

    public void transaction(float value) throws RemoteException;

}
```

The Savings Account problem

```
import java.rmi.*;
import java.util.*;
public class BankImpl implements Bank {
    private static final long serialVersionUID = 1L;
    private static transient Map<String, UserAccount> users;
    public BankImpl() {
        users = new HashMap<String, UserAccount>();
    }
    public synchronized UserAccount getUserAccount(String name) {
        UserAccount user = users.get(name);
        if (user != null) {
            return user;
        }
        try {
            user = new UserAccountImpl(name);
        } catch (RemoteException e) {
        }
        users.put(name, user);
        return user;
    }
}
```

The Savings Account problem

Удаљени приступ објекту који
није уписан у регистар

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class UserAccountImpl extends UnicastRemoteObject
        implements UserAccount, Serializable {

    private static final long serialVersionUID = 1L;
    private float status;
    private String name;

    public UserAccountImpl(String name) throws RemoteException {
        this.status = 0;
        this.name = name;
    }

    private void work() {
        try {
            Thread.sleep(500 + (int) (Math.random() * 1000));
        } catch (InterruptedException e) {
        }
    }
```

The Savings Account problem

```
        public synchronized float getStatus() {
                work();
                return status;
        }

        public synchronized void transaction(float value) {
                work();
                while (status + value < 0) {
                        try {
                                wait();
                        } catch (Exception ex) {
                        }
                }
                status += value;
        notifyAll();
}
```

The Savings Account problem

```
import java.rmi.registry.*;
import java.rmi.server.*;
public class Server {
    public static final int port = ...;
    public static final String server = ...;

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            Bank bank = new BankImpl();
            Bank stub = (Bank)
                UnicastRemoteObject.exportObject(bank, 0);
            String urlString = "/Bank";
            Registry registry = LocateRegistry.createRegistry(port);
            registry.rebind(urlString, stub);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The Savings Account problem

```
import java.rmi.registry.*;

public class Client {
    public static final String host = ...
    public static final int port = ...;
    public static void main(String[] args) {
        try {
            Bank bank = null;
            UserAccount userAcount = null;
            String name = args[0];

            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());
            }

            String urlString = "/Bank";

            Registry registry = LocateRegistry.getRegistry(host,port);
            bank = (Bank) registry.lookup(urlString);

            userAcount = bank.getUserAccount(name);
```

The Savings Account problem

```
    for (int m = 0; m < 100; m++) {
        float nstatus = (float) (50 + m - (int) (Math.random() * 100));
        try {
            System.out.println("Status: " + userAcount.getStatus());
            System.out.println("Promena statusa za " + nstatus);
            userAcount.transaction(nstatus);
            System.out.println("Novi status: " +
                    userAcount.getStatus());
        } catch (Exception e) {
            System.err.println("Greska pri transakciji za " + name);
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
```

The Savings Account problem

java.policy

```
grant {
    // Allow everything for now
    permission java.security.AllPermission;
};
```

Покретање:

```
//start rmiregistry

java -Djava.security.policy=java.policy Server

java -Djava.security.policy=java.policy Client pera
```

Питања?

Захарије Радивојевић
Електротехнички Факултет
Универзитет у Београду
zaki@etf.rs