Java Remote Method Invocation (RMI)

Distributed Software Systems

Overview

- Java RPC
- Features
  - Integrated with Java language + libraries
    - Security, write once run anywhere, multithreaded
    - Object orientation
  - Can pass "behavior"
    - Mobile code
    - Not possible in CORBA, traditional RPC systems
  - Distributed Garbage Collection
  - *Remoteness of objects **intentionally** not transparent*

Remote Interfaces, Objects, and Methods

- Objects become remote by implementing a remote interface
  - A remote interface extends the interface java.rmi.Remote
  - Each method of the interface declares java.rmi.RemoteException in its `throws` clause in addition to any application-specific clauses

Creating Distributed applications using RMI

1. Define the remote interfaces
2. Implement the remote objects
3. Implement the client (can be done anytime after remote interfaces have been defined)
4. Register the remote object in the name server registry
5. Generate the stub and client using rmic
6. Start the registry
7. Start the server
8. Run the client

Advanced Techniques

- Passing behavior
  - See Java RMI tutorial track example
- Callbacks
- Activation

Remote Interface – `CreditCard.java`

```
package credit;

import java.rmi.*;

public interface CreditCard extends Remote {

    /** This method returns a credit card's credit line. */
    public float getCreditLine() throws RemoteException;

    /** This method allows a card holder to pay all or some
        of a balance. Throws InvalidMoneyException if the
        money param is invalid. */
    public void payTowardsBalance(float money)
        throws RemoteException, InvalidMoneyException;

    /** This method allows the cardholder to make purchases
        against the line of credit. Throws CreditLineExceededException
        if the purchase exceeds available credit. */
    public void makePurchase(float amount, int signature)
        throws RemoteException, InvalidSignatureException,
        CreditLineExceededException;

    /** This method sets the card's personal i.d. signature. */
    public void setSignature(int pin) throws RemoteException;
}
```


Remote Interface – CreditManager.java

```
package credit;

import java.rmi.*;


public interface CreditManager extends java.rmi.Remote {

    /** This method finds an existing credit card for a given customer
        name. If the customer does not have an account, a new card will
        be "issued" with a random personal i.d. signature and a $5000
        starting credit line. */
    public CreditCard findCreditAccount(String Customer)
        throws DuplicateAccountException, RemoteException;


    /** This method creates a new Credit Account with a random
        personal i.d. signature and a $5000 starting credit line. */
    public CreditCard newCreditAccount(String newCustomer)
        throws RemoteException;
}
```

CreditCardImpl.java

```
package credit;

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

/** This class is the remote object that will be referenced by the
skeleton  on the server side and the stub on the client side. */

public class CreditCardImpl
   extends UnicastRemoteObject
   implements CreditCard
{
   private float currentBalance = 0;
   private float creditLine = 5000f;
   private int signature = 0;         // Like a p.i.n. number
   private String accountName;        // Name of owner

   /** Class constructor generates an initial pin.*/
   public CreditCardImpl(String customer)
   throws RemoteException, DuplicateAccountException {
      accountName = customer;
      signature = (int)(Math.random() * 10000);
   }

   /** Returns credit line. */
   public float getCreditLine() throws RemoteException {
      return creditLine;
   }

   /** Pays off some debt. */
   public void payTowardsBalance(float money)
   throws RemoteException, InvalidMoneyException {
      if (money <= 0) {
         throw new InvalidMoneyException ();
      } else {
         currentBalance -= money;
      }
   }
```



```
   /** Changes signature. */
   public void setSignature(int pin) throws RemoteException {
      signature = pin;
   }

   /** Makes a purchase. Makes sure enough credit is available,
      then increments balance and decrements available credit. */
   public void makePurchase(float amount, int signature)
   throws RemoteException, InvalidSignatureException,
   CreditLineExceededException {
      if (signature != this.signature) {
         throw new InvalidSignatureException();
      }
      if (currentBalance+amount > creditLine) {
         throw new CreditLineExceededException();
      } else {
         currentBalance += amount;
         creditLine -= amount;
      }
   }
}
```

CreditManagerImpl.java

```
package credit;

import java.rmi.*;
import java.rmi.server.*;
import java.util.Hashtable;

public class CreditManagerImpl extends UnicastRemoteObject
implements CreditManager {
   private static transient Hashtable accounts = new Hashtable();

   /** This is the default class constructor that does nothing
```

```
        but implicitly call super(). */
    public CreditManagerImpl() throws RemoteException { }

    /** Creates a new account. Puts the customer name and the
customer's  credit card in the hashtable. */
    public CreditCard newCreditAccount(String customerName)
    throws RemoteException {
        CreditCardImpl newCard = null;
        try {
            newCard = new CreditCardImpl(customerName);
        } catch (DuplicateAccountException e) {
            return null;
        }
        accounts.put(customerName, newCard);
        return newCard;
    }

    /** Searches the hashtable for an existing account. If no account
        for customer name, one is created and added to hashtable.
        Returns the account. */
    public CreditCard findCreditAccount(String customer)
    throws DuplicateAccountException, RemoteException {
        CreditCardImpl account =
(CreditCardImpl)accounts.get(customer);
        if (account != null) {
            return account;    }
        // Create new account. Add credit card to hashtable.
        account = new CreditCardImpl(customer);
        accounts.put(customer, account);
        return account;
    }
}
```


Server – CardBank.java

```
import credit.*;
import java.util.*;
import java.rmi.*;

public class CardBank {

    public static void main (String args[]) {
        //  Create and install a security manager.
        System.setSecurityManager(new RMISecurityManager());

        try {
            // Create an instance of our Credit Manager.
            System.out.println("CardBank: create a CreditManager");
            CreditManagerImpl cmi = new CreditManagerImpl();

            // Bind the object instance to the remote registry. Use the
            // static rebind() method to avoid conflicts.
            System.out.println("CardBank: bind it to a name");
            Naming.rebind("cardManager", cmi);

            System.out.println("CreditManager is now ready");

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
    }
}
```

Client – Shopper.java

```
import credit.*;
import java.rmi.*;


public class Shopper {

    public static void main(String args[]) {

        CreditManager cm = null;
        CreditCard account = null;

        // Check the command line.
        if (args.length < 2) {
            System.err.println("Usage:");
            System.err.println("java Shopper <server> <account name>");
            System.exit (1);
        }

        // Create and install a security manager.
        System.setSecurityManager(new RMISecurityManager());

        // Obtain reference to card manager.
        try {
            String url = new String ("//" + args[0] + "/cardManager");
            System.out.println ("Shopper: lookup cardManager, url = " +
url);
            cm = (CreditManager)Naming.lookup(url);
        } catch (Exception e) {
            System.out.println("Error in getting card manager" + e);

            System.exit(1);
        }

        // Get user's account.
        try {
            account = cm.findCreditAccount(args[1]);
            System.out.println ("Found account for " + args[1]);
```



```
        } catch (Exception e) {
            System.out.println("Error in getting account for " + args[1]);
            System.exit(1);

        }

        // Do some transactions.
        try {
            System.out.println("Available credit is: "
                        + account.getCreditLine());
            System.out.println("Changing pin number for account");
            account.setSignature(1234);
            System.out.println("Buying a new watch for $100");
            account.makePurchase(100.00f, 1234);
            System.out.println("Available credit is now: " +
                         account.getCreditLine());
            System.out.println("Buying a new pair of shoes for $160");
            account.makePurchase(160.00f, 1234);
            System.out.println("CardHolder: Paying off $136 of balance");
            account.payTowardsBalance(136.00f);
            System.out.println("Available credit is now: "+
                         account.getCreditLine());
        } catch (Exception e) {
            System.out.println("Transaction error for " + args[1]);
        }

       System.exit(0);
    }
}
```