Java key APIs and features
ICPOO-II.2304

Yousra Chabchoub

Outline

- Persistency
  - Serialization
  - JDBC
- Distributed programming with Java
  - Socket API
  - RMI

Serialization

Yousra Chabchoub

Serialization (1/2)

- Java provides a core native mechanism called serialization, which allows for transforming Java objects into stream of bytes
- Objects are serialized
  - For object transmission through the network
  - For simple object storage to achieve persistency
- In order to be serialized, an objet must implement the java.io.Serializable interface
- All attributes of the object are serialized if
  - If they are serializable or primitive types
  - If they are not declared with keyword static or transient
- When serialized, all the object state is encoded into bytes
  - It includes all the instance fields (including the superclasses)
  - It also writes a reference to the class (id + version)

Serialization (2/2)

- To save an object to a file

```
Personne p = new Personne("Dupont", "Jean", 36);
FileOutputStream fos = new FileOutputStream("personne.serial");
ObjectOutputStream oos= new ObjectOutputStream(fos);
oos.writeObject(p);
```

- Personne.serial is a file containing binary data that describes the object p

- To load an object from a file

```
FileInputStream fis = new FileInputStream("personne.serial");
ObjectInputStream ois= new ObjectInputStream(fis);
p = (Personne) ois.readObject();
```

Serialization Netbeans Project (On Moodle) Code sample (1/3)

Example of a Serializable class « Personne »

```
/*Personne.java class*/

import java.io.Serializable;

public class Personne implements Serializable {
     static private final long serialVersionUID = 6L;
     private String nom;
     private String prenom;
     private Integer age;

     public Personne(String nom, String prenom, Integer age) {
          this.nom = nom;
          this.prenom = prenom;
          this.age = age;
     }

     public String toString() {
          return nom + " " + nom + " " + age + " ans";
     }
}
```

Code sample (2/3)

Serialization of the class « Personne »

```
package javaserialization;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;

public class Main {

     static public void main(String[] argv) throws Exception {
               // instanciation of Personne
               Personne p = new Personne("Dupont", "Jean", 36);
               System.out.println("new instance : " + p);
               // opening a stream related to the file "personne.serial"
               FileOutputStream fos = new FileOutputStream("personne.serial");
               // creating a stream of objects related to the file stream
               ObjectOutputStream oos= new ObjectOutputStream(fos);
               oos.writeObject(new java.util.Date());
               // serialization of p
               oos.writeObject(p);
               String[] tableau = {"Au", "revoir!"};
               oos.writeObject(tableau);
               oos.flush();
               System.out.println(p + " a ete serialise");
               //closing the streams
               oos.close();
               fos.close();
               }
}
```

Comments

- If we forget « implements Serializable » in the class « Personne »
  - No compilation error
  - NotSerializableException error at the execution of `oos.writeObject(p)`
- It is possible to serialize the class **java.util.Date** because it implements the interface serializable
- Any array of serializable objects can be serialized (ex: array of strings, cf Date and String classes in java API)

Code sample (3/3)

Deserialization of the serialized objects

```
package javaserialization;

import java.io.FileInputStream;
import java.io.ObjectInputStream;

public class deserializationMain {

      static public void main(String[] argv)  throws Exception{
            Personne p = null;
            // opening a file inputStream
            FileInputStream fis = new FileInputStream("personne.serial");
            // opening an object Input stream related to the file
            ObjectInputStream ois= new ObjectInputStream(fis);
            //deserialization of the Date
            System.out.println(ois.readObject());
            // deserialization of the object personne
            p = (Personne) ois.readObject();
            System.out.println(p);
            //deserialization of the String Array
            String[]tableau = (String[])ois.readObject();
            System.out.println(tableau[0]+" "+tableau[1]);
            // closing the streams
           is.close();
           fis.close();
           if(p != null) {
                 System.out.println(p + " a ete deserialise");
           }
      }
}
```

readObject() returns an Object, one has to specify the real class if necessary (ex: `p = (Personne) ois.readObject();`)

Persistency with JDBC

Yousra Chabchoub

Data bases access in Java

- There are many techniques to access data stored in a data base:
  - JDBC (Java Database Connectivity)
  - javax.sql (DataSource)
  - JNDI (Java Naming and Directory Interface)
  - ODBC (Open Database Connectivity)
- JDBC offers a set of methods to
  - Connect to a data base
  - Execute SQL queries (SELECT, DELETE…)
- JDBC deals with many kinds of data bases (oracle, mysql…). The compatible driver is needed.

Example

```
import java.sql.*;
public class JDBCExample {
  Connection conn;

  public void runExample() throws Exception {
    try {
      // Loads the driver for MySQL
      // (MYSQL connector must be in the class path!)
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      String url = "jdbc:mysql://localhost/testdb";
      conn = DriverManager.getConnection(url, "username", "password");
      Statement st = conn.createStatement();
      ResultSet rs = st.executeQuery("SELECT name, age FROM users");
      while (rs.next())
      {
        String s = rs.getString("name");
        float n = rs.getFloat("age");
        System.out.println(s + " is " + n);
      }
    } finally {
      conn.close();
      conn = null;
    }
  }
```

Application

- Database Example Netbeans project available on moodle
- JDBC driver for MYSQL fichier is available on moodle (to add to the project Libraries)
- Needed Software
  - MySQL Server provided by MAMP (WAMP or LAMP, depending on your OS)
- Connection parameters
  - jdbc:mysql://localhost/coffeebreak
  - username=root, password=isep
- Insert the two following coffees into the table COFFEES:
  - Chocolat 5euros
  - Capuccino 6euros
- Add 1euros to the current price
- Delete coffees having a price >6
- Delete all coffees

Distributed Programming with Java

Yousra Chabchoub

Sockets

- Java provides a standard API (java.net) for TCP and UDP sockets
  - TCP sockets need to be connected (client-server) while UDP sockets can be used to send UDP packets (Datagrams) to any receiving socket (including multicasting)
  - TCP supports control-flow protocols ensure error recovery, packet ordering, and equity in bandwith
  - java.net.Socket is the root superclass for all sockets
    - Socket.getInputStream() and Socket.getOutputStream() allow to read and write from the socket (with TCP)
- ServerSocket defines a TCP socket to listen to TCP connections
  - ServerSocket.accept():Socket will wait until a connection is made, the returned socket will be used for the client-server communication
  - See the example code given with the course
- DatagramSocket.send(DatagramPacket) is used for UDP communication (no connection with accept required)
  - MulticastSocket (extends DatagramSocket) will be used for multicasting

Sockets (TCP client/server Project on Moodle)

- Server
  - serverSocket = new ServerSocket(port);
  - Socket serverSocket.accept();
  - PrintWriter out_socket;
  - out_socket = new PrintWriter(socket.getOutputStream(), true);
  - BufferedReader in_socket;
  - in_socket = new BufferedReader(new InputStreamReader (socket.getInputStream()));
  - in_socket.readLine();
  - out_socket.println("Hello");
  - socket.close();
- Client
  - InetAddress address = InetAddress.getByName("IP address ")
  - Socket socket = new Socket(InetAddress, port);
  - out_socket
  - in_socket
  - ….

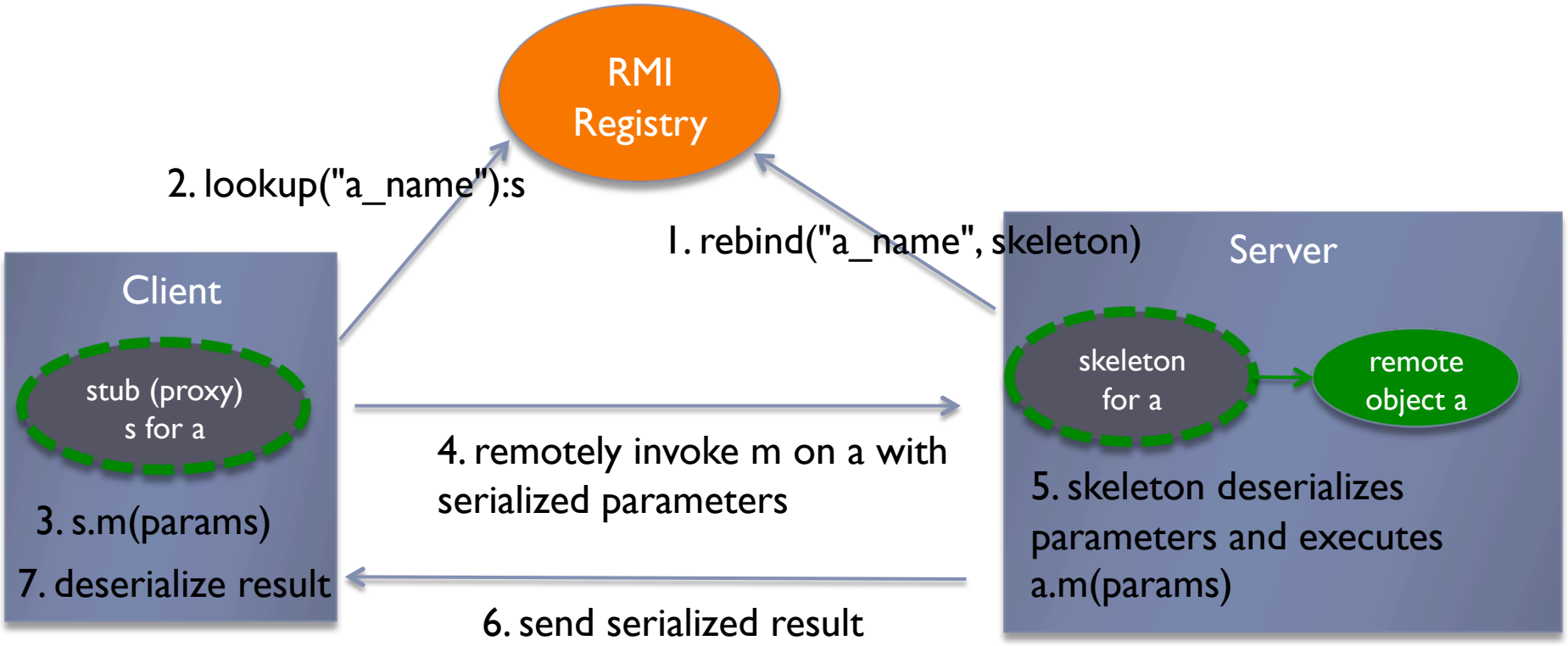
Sockets (TCP client/server Project on Moodle)

- Application
  - Modify the given example as follows : the server chooses a random number between 0 and 5 and the client has to guess. The server can help the client by answering "+" or "–" to each attempt.

Problems with the socket API

- Low-level API
  - Communication protocols and synchronization must be implemented manually (e.g. waiting for a response)
  - Serialization of data must be implemented manually (e.g. with the serialization API – see last lecture)
  - IP-dependant (one must know the IP addresses or names of the servers, which may change depending on the deployment environment)
- Java provides a mechanism to abstract the distributed communication protocol
  - Based on RPC (Remote Procedure Call)
  - But for a method call (Remote Method Invocation – RMI)
- With RMI, one can call a remote method on a remote object almost exactly like if it was a local object
  - Transparently deal with: naming, lookup, serialization, etc
- NOTE: RMI is Java to Java

RMI basics



- Object a, Skeleton, and Stub all implement the same interface that must be implementing java.rmi.Remote

RMI Netbeans Project (on Moodle)
Example/Account interface

```
package rmicommon;

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

public interface Account extends Remote {

    float credit(float amount) throws RemoteException;
    float debit(float amount) throws RemoteException;

}
```

```
package rmiserver;

import rmicommon.Account;

public class AccountImpl implements Account {

    float balance;

    public float credit(float amount) {
        System.out.println("client is
crediting "+amount);
        balance +=amount;
        return balance;
    }

    public float debit(float amount) {
        System.out.println("client is
debiting "+amount);
        balance -=amount;
        return balance;
    }


}
```

Example/server

```
package rmiserver;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import rmicommon.Account;

public class Main {

    static Registry registry;
    static Account account;

    public static void main(String[] args) throws Exception {
        System.out.println("creating the RMI registry...");
        //creates a Registry instance on the local host on port number 1099
        registry = LocateRegistry.createRegistry(1099);
        System.out.println("creating the core account (referenced to avoid potential
GC)...");
        account = new AccountImpl();
        System.out.println("wrapping the account into a remote object skeleton and bind
it...");
//exportObject(account,0):Exports the remote object account to make it available to receive incoming calls,
using port 0
//rebind: Replaces name « account » in this registry with the supplied remote reference
        registry.rebind("account", (Account) UnicastRemoteObject.exportObject(account,0));
        System.out.println("account server up and listening.");
    }
}
```

Example/client

```
package rmiclient;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import rmicommon.Account;

public class Main {

    public static void main(String[] args) throws Exception {
        //Returns a reference to the remote object Registry on the specified host and port
        Registry registry = LocateRegistry.getRegistry("localhost", 1099);
        //Returns the remote reference bound to the specified name in this registry
        Account account = (Account)registry.lookup("account");
        System.out.println("crediting 1000...");
        float result = account.credit(1000);
        System.out.println("credit operation returned "+result);
    }

}
```

Application

- Add and test a new method « transfert » in the Main class to transfert a given amount from account1 to account2

Notes on RMI

- The RMI architecture is a typical architecture of a distributed system: all of them have a repository/naming/ lookup service, which provide stubs to clients to access remote objects or resources
  - CORBA example (very similar to RMI, but cross-language)
  - WEB Services repository architecture for SOA