Category “rmi 2010/2011”

Q1 phone book interface (20 pts)

After exercise **4.2.(c)**, cut and paste to the box below the complete content of your file `PhoneBookInterface.java`.

```
package part1;

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

public interface PhoneBookInterface extends Remote
{
    /**
     * The default rmi registry name of a PhoneBook instance.
     */
    static final String DEFAULT_SERVER_NAME = "PhoneBookServer";

    /**
     * @param name -
     *            The name of the contact to be looked up in the phone book
     * @return a Contact object for the given name
     */
    Contact getContact(String name) throws RemoteException;

    /**
     * @param name -
     *            The name of the contact to be deleted from phone book
     */
    void delete(String name) throws RemoteException;

    /**
     * @return a String array of all the contact names in the phone book
     */
    String[] getNames() throws RemoteException;

    /**
     * @param contact -
     *            An contact record to add to the phone book.
     */
    void add(Contact contact) throws RemoteException;
}
```

Q2 which serialisable (19 pts)

In the box below, type the **name** of the class that you made `Serializable` in exercise **4.2.(d)**.

(100) matches regexp (case sensitive):

```
[ ]*Contact[ ]*
```

(100) matches regexp (case sensitive):

```
[ ]*contact[ ]*
```

`Contact` *is the class whose objects are passed remotely and therefore it needs to be made serialisable.*

Q3 phone book main (20 pts)

After exercise **4.2.(f)**, cut and paste to the box below your method `main` from file `PhoneBook.java`.

```
    public static void main(String[] args)
    {
        System.out.printf("setting security manager...");
        System.setSecurityManager(new RMISecurityManager());
        System.out.printf("ok\n");

        try
        {
            // create application:
            System.out.printf("creating application...");
            PhoneBook serverObject = new PhoneBook();
            System.out.printf("ok\n");

            // create rmi registry:
            System.out.printf("creating rmi registry...");
            LocateRegistry.createRegistry(1099);
            System.out.printf("ok\n");

            // bind server name to this application:
            System.out.printf("rebinding...");
            Naming.rebind("MyFriends", serverObject);
            System.out.printf("ok\n");
        }
        catch (Exception e)
        {
            System.out.println("An exception occured while creating server");
            e.printStackTrace();
        }
    }
```

Q4 client display names (20 pts)

After exercise **4.2.(g)**, cut and paste to the box below your method `displayPhoneBookNames` from file `PhoneBookClient.java`.

```
    private static void displayPhoneBookNames(PhoneBookInterface phoneBook)
       throws RemoteException
    {
        for (String name : phoneBook.getNames())
        {
            System.out.println(name);
        }
    }
```

Q5 client server - PhoneBook (7 pts)

Suppose that the phone book client and server are to be packaged as separate Eclipse projects so that they can be distributed separately.

Indicate whether the source code of the **PhoneBook** class can be safely omitted from the client or from the server or whether it has to be present on both client and server.

(100) ☐ can be omitted on client

(0) ☐ can be omitted on server

(0) ☐ has to be present on both client and server

The `PhoneBook` class provides the inner working of the remote server object which is not required on the client. The client only needs to know the `PhoneBookInterface`.

Q6 client server - PhoneBookInterface (7 pts)

Answer the previous question, but for `PhoneBookInterface`.

(0) ☐ can be omitted on client

(0) ☐ can be omitted on server

(100) ☐ has to be present on both client and server

The interface is needed on the client to be able to create a proxy to the remote phone book object and call its methods. The interface is also needed on the server so that its JVM knows which methods of the phone book object are remotely accessible.

Q7 client server - Product (7 pts)

Answer the previous question, but for the `Contact` class.

(0) ☐ can be omitted on client

(50) ☐ can be omitted on server

(100) ☐ has to be present on both client and server

The class `Contact` *is currently used by both client and server, so it has to be on both.*

Nevertheless, in principle it could be omitted from the server if the server code would treat `Contact` *instances it receives as if they were instances of* `Contact`*'s parent class* `Object`*. If that were so, one would need to arrange that the compiled code of the* `Product` *class could be automatically downloaded by the server from the client when it first receives a* `Contact` *instance so that it knows how the class* `Contact` *has overridden the inherited* `Object` *methods.*

Total of 100 pt.