Freie Universität Berlin

Prof. Dr. Margarita Esponda

Nichtsequentielle Programmierung, SoeSe 2017

Übungsblatt 9

TutorIn: Lilli Walter
Tutorium 6

Boyan Hristov, Sergelen Gongor

11. Juli 2017

Link zum Git Repository: https://github.com/BoyanH/Freie-Universitaet-Berlin/tree/master/NichtsequentielleUndVerteilteProgrammierung/Solutions/Homework9

Aufgabe 2

1. Kommunikationsmodelle

   a) Asynchrones Modell Prozesse sind entweder im aktiven oder passiven Zustand. Wenn ein Prozess im aktiven Zustand ist, darf er Nachrichte versenden. Prozesse können zu jeder Zeit passiv werden, aber diese können nur dann aktiv werden, wenn sie eine Nachricht bekommen.

   Das Programm terminiert, wenn alle Prozesse pasiv sind UND keine Nachrichten unterwegs sind

   Vorteile:
   - Threads müssen nicht aktiv warten, können weiter arbeiten, bevor sie Antwort bekommen

   Nachteile:
   - Driftrate der Uhrzeit
   - Verzögerung der Versendung von Nachrichten - Unbekannte Ausführungszeit

   b) Synchrones Modell Nachrichten haben keine Laufzeit, nur die Prozesse. Bein Senden einer Nachricht wartet das Prozess, bis er Antwort bekommt.

   Da Prozesse auf Antwort beim Nachrichtenversand warten, terminiert das Programm wenn alle Prozesse passiv werden.

   Vorteile:
   - Die Ausführungszeit der Aktionen ienes Prozesses ist asymptotisch bekannt.
   - Jeder Prozess hat eine lokale Uhr, deren Driftrate-Einschränkungen in Echtzeit bekannt sind, d.h. da der Prozess immer auf Antwort aktiv wartet, kann er die Zeit messen.
   - Die Nachrichte werden über Kanäle innerhalb vorgegebener Zeitschränkungen übertragen

Nachteile:
- Prozesse dürfen nicht weiter arbeiten, bevor diese Antwort auf einem Nachrichversand bekommen

c) Atommodell Die Nachrichten haben eine Laufzeit, die Prozesse aber nicht.

Terminiert, wenn keine Nachrichten unterwegs sind.

2. Arten von Transparenz

a) Ortstransparenz Ressourcen werden nur über Namen identifiziert, der Ort, wo diese enthalten / gespeichert sind ist nicht relevant. Dadurch wird eine ganz tolle Abstraktionsschicht hergestellt, der Entwickler muss sich keine Sorgen machen, wo die Ressourcen zu finden sind, hauptsache die sind alle verwendbar. Kann man bei RMI betrachten, da werden Objekte in dem Registry registriert, welches Thread diese hergestellt hat und wo die gespeichert sind ist irrelevant, man kann diese aber überall benutzen.

b) Zugriffstransparenz Zugriffsformen sind vom Betriebssystem und Ort der Ressourcen unabhängig. Wieder ein Abstraktionsschicht, man kann die selbe Logik über unterschiedliche Betriebssysteme benutzen. Diese Art von Transparenz ist eine Erweiterung von Ortstransparenz, eine Kombination aus Ortstransparenz und Transparenz der Zugriffsmethode.

c) Replikationstransparenz
Der Benutzer merkt nicht, ob er mit dem Original eines Objekts arbeitet oder mit einer Kopie. Weiteres Abstraktionsschicht, der Entwickler muss sich keine Sorgen machen, ob alles richtig gespeichert wird und selber die Laufzeit durch lokale Kopies optimieren. Dadurch wird Effizienz erreciht ohne die zusätzliche Komplexität.

d) Relokationstransparenz
Ressourcen werden zur Laufzeit auf einem anderen Ort verschoben. Das wird wieder deswegen gemacht, damit höhere Effizienz erreicht wird, oder falls es in dem Speichermedium kein Platz mehr gibt. Das ist aber Betriebssystem- und Rechner- abhängig. Durch dem Transparenz werden wieder Sorgen des Entwicklers entfernt.

e) Migrationstransparenz
Ressourcen werden von einem Rechner zu anderen verlagert, ohne dass der Benutzer es merkt. Vorteile sind äquivalent zu diese von Relokationstransparenz.

f) Fehlertransparenz
Systemfelher und dessen Behebung sind vom Benutzer verborgen. Ist wieder für die Logik der Anwendung nich relevant und muss vom Entwickler des verteilten Systems nicht behandelt werden, also ein weiteres Abstraktionsschicht.

g) Nebenläufigkeitstransparenz
Die Synchronisation des Zugriffs aus gemeinsame Ressourcen wird vom Benutzer verborgen. Vorteile äquivalent zu diese der Fehlertransparenz

Aufgabe 3

1. Was ist das Middleware? Welche Dienste soll die Middleware bereitstellen?
Middleware ist eine Sammlung von APIs, die ein Abstraktionsschicht für die Entwicklung von Client-/Server-Systemen, wobei die Entwicklung viel leichter wird.

Ein Middleware bietet Abstraktion von der Netzprogrammierung, asynchronen Austausch von Nachrichten, Erweiterung der Netzverbindung (optimierung von Laufzeit, Felherbehandlungen usw.,

alles was bein Arten von Transparenz schon erwähnt wurde), synchrone Verbindung, einheitlichen Zugriff auf DBS.

2. Welche Middleware Kategorien kennen Sie?
   Anwendugsorientierte(CORA, JEE, .NET usw.), Kommunikationsorientierte(RPC,RMI, Web Service usw.) und Nachrichtenorientierte (MOM, JMS usw.)

3. Was verstehen Sie unter MOM - Message Oriented Middleware?
   Das ist ein Nachrichteorientiertes Middleware. Es gibt ein Vermittler (der MOM Server) und eine Warteschlange von Nachrichten, welche den Nachrichtenaustausch steuern. Serven und Empfänger sind dabei unabhängig von einander und Nachrichten können zusammengebundelt (mehrere Nachrichte) abgeholt werden. Also der MOM-Server steht zwischen Server und Client und steuert die ganze Kommunikation, wobei die feste Kopplung zwischen den Beiden entfernt wird.

4. Welche sind die wichtigsten Schnittstellen einer JMS-Anwendung?

   a) ConnectionFactory
      Baut die Verbindungen zu einem JMS-Provider auf

   b) Connection
      Wird von ConnectionFactory erstellt und erstellt ein Kommunikationskanal zu JMS-Provider

   c) Session
      Stellt den kontext, in dem Nachrichten, Sender und Empfänger erzeugt werden, dar

   d) Message
      Eine Nachricht, mit dem dazugehörigen Meta-Daten

   e) Destination
      Eine Warteschlange des JMS-Providers, wo eine Nachricht landet

   f) MessageProducer

   g) MessageConsumer

5. Was versteht man unter SOA - Service Oriented Architectures?
   Das ist eine Architektur, die als Grundprinzip die Verteilung von Dienste hat. Dienste sind stark gekapselt und können auf mehrere Rechnern sich befinden. Die Architektur hat als Kommunikationsmedium MOM, führt das Client/Server Prinzip weiter und hat als Kern Stardtechnologien wie XML, SOAP, WSDL und ist von spezifischen Betriebssystem und Datenbanken entkoppelt.

6. Was ist ein dynamischer Web-Server.
   Konkrete Definition wurde in der Vorlesung nicht erwähnt. Es gibt zwei Antwortsmöglichkeiten

   a) Im Gegenteil vom statischen Web-Server
      Ein Web-Server, der Methodenaufrufe und Logik hat und nicht nur die einzelne Möglichkeit anbietet, Dateien zurückzugeben

   b) Schätzung im Kontext vom verteilte Systemen
      Ein Web-Server mit verteilten Dienste, also Dienste die sich auf mehreren Rechner befinden.

7. Erläutern Sie die SOAP und WSDL Protokolle.

   a) SOAP
      Ein von Microsoft, IBM und Lotus entwickeltes XML basiertes Kommunikationsprotokoll. Das Protokoll ist Platform unabhängig, kann mit Firewalls umgehen und ist leicht erweiterbar. Es basiert sich auf HTTP, HTTPS und XML. Eine Nachricht wird mit SOAP wie ein Briefumschlag weitergegeben, wobei es die Struktur der Nachricht und wie es bearbeitet werden soll darauf steht.

b) WSDL
WSDL ist kein Protokoll, sondern eine Sprache!!! (Von Englisch Web Services Description Language).
Das ist eine Sprache, die die WebDienste beschreibt, wie technische Details (Transportprotokoll, Adresse der Dienste) und Spezifikation der Funktionsschnittstellen. Das ganze wird in XML beschrieben. Die Hauptelemente von WSDL sind:

i. Types
Datentypdefinitionen

ii. Message
Typisierte Definition der kommunizierten Daten

iii. Operation
Beschreibt was ein Dienst eigentlich macht

iv. Port Type
Alle Dienste, die auf dem Port zu finden sind

v. Binding
Spezifiziert das Protokoll und Datenformat für ein Port Type

vi. Port
Endpunkt

vii. Service
Eine Menge zusammengehöriger Endpunkte

Aufgabe 4

1. RPC

a) Vorlauf

i. Der Client ruft den Client Stub auf (das Lokale Pointer von dem externen Dienst)

ii. Marshalling. Der Client Stub verpackt die Parameter für den externen Prezeduraufruf und macht ein Systemcall für den Nachritversand

iii. Betriebssystem schickt die Nachricht vom Klientenrechner zu dem Serverrechner

iv. Das Betriebssystem vom Server bearbeitet die Packete und übergibt die zum Server Stub

v. Unmarshalling. Die verpackte Parameter werden geparsed.

vi. Die Prozedur wird auf dem Server aufgerufen. Danach werden die gleiche Schritte im Gegenrichtung ausgeführt

b) Aus Entwicklungsschicht

i. Jede Klasse, die externe Prozeduren anbietet wird implementiert

ii. Der Server erstellt ein neuse Objekt aus der Klasse PropertyHandlerMapping, dass als ein Register dient, wo alle mögliche Dienste registriert werden müssen

iii. Alle Dienste, also Klassen mit externen Prozeduren, werden in dem Registry hinzugefügt

iv. Es wird ein WebServer auf einem bestimmten Port hergestellt

v. Darauf wird ein XmlRpcServer hergestellt

vi. zu dem XmlRpcServer wir der Register gegeben

vii. Am Ende wird der Server gestartte

viii. Der Client erstellt ein XmlRpcClient

ix. Der Client erstellt eine XmlRpcClientConfigImpl, also eine Konfiguration, wo definiert wird, wo sich der XmlRpcServer befindet

x. Der Client führt ein RPC aus mit XmlRpcClient(instance).execute('<registeredClass.method>', <params>)

xi. Dabei wird alles in XML verpackt und über dem Netz geschickt wie oben beschrieben

2. Marshalling
   Das ist das Prozess, in dem ein Objekt / Parametern von dem üblichen Form (wie diese in Java zu finden ung gespeichrt sind) in einem andere Form transformiert werden, wenn diese anders gespeichert werden müssen oder transportiert werden müssen. Zum Beispiel wenn ale Parameter für ein RPC transportiert werden müssen, werden diese in XML transformiert.

3. Art von Parameterübergabe in RPC
   Es wird Call-by-value benutzt (ist auch im XML offensichtlich). Das Problem mit call-by-reference und RPC ist, dass die unterschiedliche Rechner keinen gemeinsamen Speicherplatz haben, kann also keine richtige Referenz übergegeben werden. Call-by-reference könnte durch Copy-restore (google it) implementiert werden, dabei kann aber eine Menge von Issues entstehen, wird also nicht gemacht.

4. Was muss gemacht werden, um Objekte an einen entfernten Server als Parameter einer RMI senden zu können?

   Diese Objekte müssen speziell bezeichnet werden, damit die konkrete RMI Implementierung wissen kann, dass eine Kopie übergeben werden soll und dass die Methodenaufrufe auf dem Objekt dann extern sein müssen. Im Java muss man von der UnicastRemoteObject Klasse erben.

5. Welche sind die wichtigen Softwarekomponenten, die in einem XML-RPC teilnehmen?

   a) Betriebssystem

   b) XMLRPCServer

   c) PropertyHandlerMapping (Der Register, wo alle mögliche Dienste registriert werden)

Aufgabe 1

Wir haben uns für die 2. Aufgabe entschieden, also ein Kalender zu bauen. Dabei haben wir gesehen, dass die Nutzer nur Events sehen können, in denen diese teilnehmen. Wir haben davon ausgegangen, dass nur die Nutzer, die an einem Event teilnehmen, weitere Nutzer einfügen können. Diese müssen also die Events updaten.

Die Funktionalität kann man testen, in dem man die in ClientInputThread definiert Befehle benutzt (oder beim Bedarf weitere einfügt. Diese sollen aber die erforderte Funktionalität anbieten). Bein next-Event kann man sehen, dass egal was in der Console geschrieben wird, gibt es kein Antwort bis das nächste Event angefangen ist, also der Call wird blockiert, damit der Client auch (das haben wir bewusst gemacht).

Wichtig bei der Ausführung sind die Parameter. Da das Betriebssystem und JVM einige Ports und Hosts schützen, haben wir localhost und port 8080 benutzt. Dafür muss man die zwei Programme folgendermasen starten

```
java -jar CalenderRMI.jar client 127.0.0.1 8080

java -jar CalenderRMI.jar server 8080
```

```
package alpv.calendar;

import java.lang.reflect.Array;
import java.rmi.RemoteException;
import java.util.*;

public class CalendarServerImpl implements CalendarServer {

    private Dictionary<Long, Event> eventsById;
    private Dictionary<Long, EventWaiterThread> waiterForEvent;
    private Object eventsLock;
    private static long eventsCount = 0;
    private Dictionary<String, List<EventCallback>> callbacksForUser;

    public CalendarServerImpl() {
        this.eventsById = new Hashtable<>();
        this.waiterForEvent = new Hashtable<>();
        this.eventsLock = new Object();
        this.callbacksForUser = new Hashtable<>();
    }

    @Override
    public long addEvent(Event e) throws RemoteException {
        e.setId(++eventsCount);

        synchronized (eventsLock) {
            long id = e.getId();
            EventWaiterThread eventWaiter = new EventWaiterThread(e, this);
            this.eventsById.put(id, e);
            this.waiterForEvent.put(id, eventWaiter);
            eventWaiter.start();
            return id;
        }
    }

    @Override
    public boolean removeEvent(long id) throws RemoteException {
        synchronized (eventsLock) {
            Event registeredEventWithId = this.eventsById.remove(id);
            EventWaiterThread waiterForEvent = this.waiterForEvent.remove(id);

            if (waiterForEvent != null) {
                waiterForEvent.interrupt();
            }

            return registeredEventWithId != null;
        }
    }

    @Override
    public boolean updateEvent(long id, Event e) throws RemoteException {
        synchronized (eventsLock) {
            Event registeredEventWithId = this.eventsById.get(id);
            EventWaiterThread waiterForEvent = this.waiterForEvent.get(id);

            /**
             * If the date has changed, we need to restart the waiter thread as well, so
    just
             * add a new event with the same id, remove the current one
```

```
59              *
60              * Not the best implementation ever, but keeps our timeout logic clean and
    simple
61              */
62             if (registeredEventWithId == null) {
63                 return false;
64             } else if (registeredEventWithId.getBegin().compareTo(e.getBegin()) == 0) {
66                 registeredEventWithId.setUser(e.getUser());
67                 registeredEventWithId.setName(e.getName());
69             } else {
70                 e.setId(registeredEventWithId.getId());
71                 this.removeEvent(id);
72                 this.addEvent(e);
73                 return true;
74             }
75         }
77         return false;
78     }
80     @Override
81     public List<Event> listEvents(String user) throws RemoteException {
82         synchronized (eventsLock) {
83             List<Event> eventsList = new LinkedList<>();
84             Enumeration<Event> events = this.eventsById.elements();
85             Event currentEvent = null;
87             try {
88                 do {
89                     currentEvent = events.nextElement();
91                     if (currentEvent != null && Arrays.asList(currentEvent.getUser()).
    indexOf(user) > -1) {
92                         eventsList.add(currentEvent);
93                     }
94                 }
95                 while (currentEvent != null);
96             } catch (NoSuchElementException e) {
97                 // dictionary has no further items
98             }
101            return eventsList;
102        }
103    }
105    @Override
106    public Event getNextEvent(String user) throws RemoteException {
107        Event closestEvent;
108        long timeBeforeNextEvent;
109        Date currentDate = new Date();
111        synchronized (eventsLock) {
112            List<Event> userEvents = this.listEvents(user);
113            closestEvent = userEvents.get(0);
115            for (Event currentEvent : userEvents) {
116                // if the event is closer in the future, but still after the current date
117                if (currentEvent.compareTo(closestEvent) < 0 && currentDate.compareTo(
    currentDate) < 0) {
118                    closestEvent = currentEvent;
119                }
120            }
121        }
123        if (closestEvent != null) {
```

```
124             timeBeforeNextEvent = closestEvent.getBegin().getTime() - new Date().getTime();
125             try {
126                 Thread.sleep(timeBeforeNextEvent);
127                 return closestEvent;
128             } catch (InterruptedException e) {
129                 // interrupted, don't return anything
130                 return null;
131             }
132         }
134         return null;
135     }
137     @Override
138     public void RegisterCallback(EventCallback ec, String user) throws RemoteException {
139         List<EventCallback> callbacksForGivenUser = this.callbacksForUser.get(user);
141         if (callbacksForGivenUser == null) {
142             List newList = new LinkedList<>();
143             newList.add(ec);
144             this.callbacksForUser.put(user, newList);
145         } else {
146             callbacksForGivenUser.add(ec);
147         }
148     }
150     /**
151      * Why the fuck is the user not given here but was given in RegisterCallback?
152      * ...consistency is for the weak...no worries, I'll find it from all callbacks...
153      */
154     @Override
155     public void UnregisterCallback(EventCallback ec) throws RemoteException {
156         Enumeration<List<EventCallback>> callbacks = this.callbacksForUser.elements();
157         List<EventCallback> currentUserCallbacks;
159         try {
160             do {
161                 currentUserCallbacks = callbacks.nextElement();
163                 // if the callback was found in the current list of callbacks, stop
    searching
164                 // it was successfully removed
165                 if (currentUserCallbacks != null && currentUserCallbacks.remove(ec)) {
166                     break;
167                 }
169             } while (currentUserCallbacks != null);
170         } catch(NoSuchElementException e) {
171             // dictionary has no further elements
172         }
174     }
176     public void eventStarted(long id) {
177         Event startedEvent = this.eventsById.get(id);
178         String[] users = startedEvent.getUser();
180         for (String user : users) {
182             for (EventCallback ec : this.callbacksForUser.get(user)) {
183                 try {
184                     ec.call(startedEvent);
185                 } catch (RemoteException e) {
186                     e.printStackTrace();
187                 }
188             }
189         }
190     }
```

```
}
```

```
package alpv.calendar;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.List;

public class ClientInputThread extends Thread {

    private CalendarServer stub;
    private BufferedReader readBuffer;
    private EventCallback ec;
    String userName;

    public ClientInputThread(CalendarServer stub) {
        this.stub = stub;
        this.readBuffer = new BufferedReader(new InputStreamReader(System.in));;
    }

    @Override
    public void run () {

        while(true) {

            try {
                String newLine = this.readBuffer.readLine();
                this.parseCommand(newLine);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }

    private void parseCommand(String line) {
        String eventName;
        Long milliseconds;

        try {
            if (this.userName == null && !line.equals("setName")) {
                System.out.println("Cannot execute any commands without setting your name first. Execute \"setName\" first ");
            }

            switch (line) {
                case "newEvent":
                    System.out.print("Enter event name: ");
                    eventName = this.readBuffer.readLine();
                    System.out.print("After how many seconds is the event happening: ");
                    milliseconds = Long.parseLong(this.readBuffer.readLine()) * 1000;
                    String[] user = {this.userName};

                    stub.addEvent(new Event(eventName, user, new Date(new Date().getTime() + milliseconds)));
                    break;
                case "setName":
                    System.out.print("Enter your name: ");
                    this.userName = this.readBuffer.readLine();

                    if (this.ec != null) {
                        stub.UnregisterCallback(this.ec);
                        this.ec = new EventCallbackImpl(this.userName);
                        stub.RegisterCallback(ec, this.userName);
                    }
```

```
                    break;
                case "addUserToEvent":
                    List<Event> eventsList = stub.listEvents(this.userName);
                    for (Event e : eventsList) {
                        System.out.printf("#%s \"%s\" starting on %s\n", e.getId(), e.
    getName(), e.getBegin());
                    }
                    System.out.print("Which event id? : ");
                    long eventId = Long.parseLong(this.readBuffer.readLine());
                    System.out.print("Which user? : ");
                    String newUser = this.readBuffer.readLine();

                    for (Event e : eventsList) {
                        if (e.getId() == eventId) {
                            Event newEvent = new Event(e.getName(), e.getUser(), e.getBegin
    ());
                            String[] newUsers = new String[newEvent.getUser().length];

                            for (int i = 0; i < newEvent.getUser().length; i++) {
                                newUsers[i] = newEvent.getUser()[i];
                            }
                            newUsers[newUsers.length-1] = newUser;
                            stub.updateEvent(newEvent.getId(), newEvent);
                            break;
                        }
                    }

                    break;
                case "subscribeForEvents":
                    this.ec = new EventCallbackImpl(this.userName);
                    stub.RegisterCallback(ec, this.userName);
                    break;
                case "unsubscribe":
                    stub.UnregisterCallback(this.ec);
                    this.ec = null;
                    break;
                case "removeEvent":
                    List<Event> eventList = stub.listEvents(this.userName);
                    for (Event e : eventList) {
                        System.out.printf("#%s \"%s\" starting on %s\n", e.getId(), e.
    getName(), e.getBegin());
                    }
                    System.out.print("Which event id? : ");
                    long eId = Long.parseLong(this.readBuffer.readLine());

                    stub.removeEvent(eId);
                    break;
                case "nextEvent":
                    Event next = stub.getNextEvent(this.userName);
                    System.out.printf("Next event awaited, event name: %s\n", next.getName
    ());
                default:
                    System.out.println("Command not recognized. Check ClientInputThread.
    java for more info");

            }

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

```
package alpv.calendar;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
```

```
public class EventCallbackImpl extends UnicastRemoteObject implements  EventCallback {

    public String uName;

    public EventCallbackImpl(String name) throws RemoteException {
        this.uName = name;
    }

    @Override
    public void call(Event e) throws RemoteException {
        System.out.printf("%s event started! Whohoo; In callback for user %s \n", e.getName
    (), this.uName);
    }
}
```

```
package alpv.calendar;

import java.util.Date;

public class EventWaiterThread extends Thread {

    private Event event;
    private CalendarServerImpl server;

    public EventWaiterThread(Event e, CalendarServerImpl server) {
        this.event = e;
        this.server = server;
    }

    @Override
    public void run() {
        Date currentDate = new Date();
        System.out.println(this.event.getBegin());
        long millisecondsToBegin = this.event.getBegin().getTime() - currentDate.getTime();
        try {
            EventWaiterThread.sleep(millisecondsToBegin);
            this.server.eventStarted(this.event.getId());
        } catch (InterruptedException e) {
            // interrupted, just end
            // e.printStackTrace();
        }

    }
}
```

```
package alpv.calendar;

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

public class Main {
    private static final String USAGE   = String.format("usage: java -jar UB%%X_%%NAMEN
    server PORT%n" +
                                                        "         (to start a server)%n" +
                                                        "or:    java -jar UB%%X_%%NAMEN
    client SERVERIPADDRESS SERVERPORT%n" +
                                                        "         (to start a client)");

    private static CalendarServer calendarServerStub;

    /**
     * Starts a server/client according to the given arguments.
     * @param args
     */
    public static void main(String[] args) {
        try {
```

```
21              int i = 0;
22
23              if(args[i].equals("server")) {
24                  try {
25
26                      /**
27                       * Set the java's rmi server's hostname to 127.0.0.1 AKA localhost, we
    don't want a global server
28                       */
29                      System.setProperty("java.rmi.server.hostname","127.0.0.1");
30
31                      // Bind the remote object's stub in the registry
32                      Registry registry = LocateRegistry.createRegistry(Integer.parseInt(args
    [1]));
33
34                      /**
35                       * It's important to always keep a reference to the class registered in
     the LocateRegistry,
36                       * otherwise it could get garbage-collected before it was successfully
    registered
37                       */
38                      CalendarServerImpl calendarServerImpl = new CalendarServerImpl();
39                      calendarServerStub = (CalendarServer) UnicastRemoteObject.exportObject(
    calendarServerImpl, Integer.parseInt(args[1]));
40                      registry.bind("CalendarServer", calendarServerStub);
41
42                      System.err.println("Server ready");
43                  } catch (Exception e) {
44                      System.err.println("Server exception: " + e.toString());
45                      e.printStackTrace();
46                  }
47              }
48              else if(args[i].equals("client")) {
49                  try {
50                      Registry registry = LocateRegistry.getRegistry(args[1], Integer.
    parseInt(args[2])
51                      );
52                      CalendarServer stub = (CalendarServer) registry.lookup("CalendarServer"
    );
53                      new ClientInputThread(stub).start();
54
55                  } catch (Exception e) {
56                      System.err.println("Client exception: " + e.toString());
57                      e.printStackTrace();
58                  }
59              }
60              else
61                  throw new IllegalArgumentException();
62          }
63          catch(ArrayIndexOutOfBoundsException e) {
64              System.err.println(USAGE);
65          }
66          catch(NumberFormatException e) {
67              System.err.println(USAGE);
68          }
69          catch(IllegalArgumentException e) {
70              System.err.println(USAGE);
71          }
72      }
73  }
```