Praktikum Verteilte Anwendungen 1

Gruppenaufteilung

Das Praktikum findet in zwei Gruppen statt. Beachten Sie bitte, dass Sie nur an dem Ihnen zugewiesenen Termin teilnehmen können.

Anwesenheitspflicht

Für das Praktikum besteht Anwesenheitspflicht. Tragen Sie sich bitte an jedem Termin in die Anwesenheitsliste ein. Die regelmäßige Anwesenheit ist eine Voraussetzung für das Bestehen des Praktikums.

Einzelarbeit

Die Veranstaltung gilt nur dann als besucht, wenn Sie einzeln am Rechner arbeiten. Gruppenarbeit ist nicht zulässig. Das gemeinsame Arbeiten vor einem Rechner wird so gewertet, als ob keiner der beiden die Veranstaltung besucht hat. Das gemeinsame Abgeben von Lösungen wird so bewertet, als ob keiner der beiden die Lösung abgegeben hat.

Abgabe

Zu jedem Praktikumstermin gibt es ein Aufgabenblatt mit mehreren Aufgaben. Zum Bestehen des Praktikums müssen Sie diese selbstständig und erfolgreich bearbeiten. Schicken Sie allen von Ihnen erstellten Programmcode an: Thomas.Mendelssohn@fh-furtwangen.de. Schicken Sie pro Aufgabe eine E-Mail mit folgendem Aufbau:

- **To (An):** Thomas.Mendelssohn@hs-furtwangen.de
- **Subject (Betreff):** Geben Sie im Subject hintereinander die Nummer der Aufgabe und Ihren Namen an. Beispiel: 2.4 Michael Maier.
- **Body (Rumpf):** Der Body der Nachricht soll allen Programmcode enthalten. Haben Sie mehrere Klassen geschrieben, so hängen Sie die einzelnen Textstücke bitte hintereinander. Tipp: Übertragen Sie den Programmcode einfach mit Kopieren-Einfügen von der Entwicklungsumgebung in Ihre E-Mail. Verwenden Sie keine Attachements (Anhänge)!

Abgabetermin

Auf jedem Aufgabenblatt finden Sie einen Abgabetermin. Gelingt es Ihnen während des Praktikums nicht, die Aufgaben vollständig zu bearbeiten, so können Sie dies bis zum Abgabetermin nachholen.

Rücksprache

Das Praktikum gilt nur als bestanden, wenn es Ihnen gelingt, mir Ihre Lösungen zu erläutern und meine Fragen hierzu zu beantworten. Ich werde die Arbeiten fortlaufend in Einzelgesprächen mit Ihnen besprechen. Studenten, die während der ersten Wochen des Praktikums nicht dazu gekommen sind, mir Ihre Lösungen zu präsentieren, werde ich am Ende des Semesters gezielt ansprechen, um Ihnen die Gelegenheit hierzu zu geben.

Eigene Laptops

Wer einen eigenen Laptop besitzt, kann diesen gerne zum Praktikum mitbringen und darauf programmieren. Die im Praktikum verwendete Entwicklungsumgebung Eclipse ist im Internet für alle gängigen Betriebssysteme frei verfügbar. Bei der Installation bin ich gegebenenfalls gern behilflich.

Aufgabenblatt 1

Praktikum am 14.3.2012
Abgabe bis 20.3.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 1.1 - Eieruhr

In dieser Aufgabe soll eine Eieruhrfunktion realisiert werden: Einmal gestartet gibt eine Eieruhr nach einer vorgegebenen Zeit einen Alarm aus.

Realisieren Sie die Eieruhrfunktion in Form einer statischen Methode mit dem Namen *eieruhr*. Die Methode soll zwei Parameter haben: eine Zeitangabe in Millisekunden und einen Ausgabetext, der zum Alarmzeitpunkt ausgegeben werden soll. Nach dem Aufruf der Methode *eieruhr* soll während der Wartezeit weiterarbeiten können. Konsequenz: Realisierung durch einen zweiten Thread.

Tipps:

- ✗ Erzeugen und starten Sie innerhalb der eieruhr-Methode einen neuen Thread. Dazu müssen Sie zunächst eine Sohnklasse von Thread implementieren und deren run-Methoden geeignet überschreiben.
- ✗ Verwenden Sie folgende Methode, um einen Thread für eine vorgegebene Zeit (m Sekunden) warten zu lassen:

```
public static void schlafen(int m) {
  try {
     Thread.sleep(1000*m);
  } catch (InterruptedException t) {
  }
}
```

A) Implementieren Sie zunächst die Methode eieruhr. Testen Sie die Methode, indem Sie sie mehrfach hintereinander mit unterschiedlichen Parameterwerten aufrufen.

B) Modifizieren Sie die Methode derart, dass die Eieruhr jede Sekunde die noch verbleibende Zeit und die Nachricht ausgibt.

Aufgabe 1.2 - Dispatcher

Gegeben folgende Schnittstelle:

```
public interface F {
    public int f (int x);
}
```

Es soll eine statische Methode mit dem Namen *execute* programmiert werden. Die Methode *execute* hat zwei Parameter: einen Parameter *f* vom Typ *F* und einen Parameter *n* vom Typ *int*. Die Methode execute soll die Werte *f(0)*, *f(1)*, *f(2)* ... *f(n-1)* bestimmen, diese Werte in einem int-Array nacheinander ablegen und schließlich diesen int-Array zurückgeben. Signatur von *execute*:

```
public static int[] execute(F f, int n)
```

Die Methode *execute* soll die Funktionsaufrufe *f(0)*, *f(1)*, *f(2)* ... *f(n-1)* nicht nacheinander ausführen, sondern vielmehr sollen diese Funktionsaufrufe gleichzeitug je durch einen Thread ausgeführt werden.

Tipps:

- ✗ Starten Sie innerhalb von *execute* *n* Threads. Übergeben Sie jedem der Threads eine der *x*-Werte 0, 1, 2, ... *n*-1, sowie das Objekt *f*. Der Thread soll dann *f(x)* bestimmen.
- ✗ Erzeugen Sie in der Methode *execute* zusätzlich ein Objekt mit von der selbstprogrammierten Klasse *Result*. Übergeben Sie auch dieses Objekt jedem der Threads.
- ✗ In das Objekt vom Typ Result sollen die Threads ihre Einzelergebnisse ablegen können und der Hauptthread (*execute*-Methode) soll daraus das Gesamtergebnis auslesen können. Realisieren Sie diese beiden Funktionen je als eine Objektmethode.
- ✗ Beim Auslesen des Gesamtergebnisses soll der Aufrufer solange blockiert werden, bis alle *n* Ergebnisse vorliegen. Kennzeichnen Sie die beiden Methoden von Result als synchronized und verwenden Sie innerhalb der Methoden *wait* und *notify* zur Synchronisation.

Aufgabenblatt 2

Praktikum am 21.3.2012
Abgabe bis 27.3.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 2.1 - Time-Service

In dieser Aufgabe soll ein Internet-Service programmiert werden, von dem Clients das Datum und die Urzeit des Servers abfragen können.

Das Programm soll den Service auf dem Port 75 zur Verfügung stellen. Das Programm soll in einer Endlosschleife eine Verbindung nach der anderen entgegennehmen: sobald eine Verbindung beendet wurde, nimmt das Programm die nächste entgegen. Die Kommunikation zwischen Client und Server soll zeilenweise erfolgen. Sofort nach dem Verbindungsaufbau soll der Server dem Client die Textzeile "time service" zuschicken. Schickt der Client dem Server anschließend eine Textzeile mit dem Inhalt "date" oder "time" so antwortet der Server mit einer Textzeile, in der das Datum bzw. die Zeit steht. Sobald der Client eine andere Textzeile als "date" oder "time" zum Server schickt, beendet der Server die Verbindung.

Beispiel:

```
time service
date
19.10.2003
time
15:29:44
time
15:29:46
date
19.10.2003
end
```

A) Schreiben Sie eine Klasse mit dem Namen TimeService, die diesen Dienst realisiert. Starten Sie die Klasse und testen Sie das Programm, indem Sie vom eigenen Rechner aus mit telnet eine Verbindung zu dem Service aufbauen.
Aufruf: telnet 127.0.0.1 75
Betrachten Sie dabei vor und nach dem Verbindungsaufbau die aktuellen Netzwerkverbindungen mit netstat.

B) Stellen Sie von einem anderen Rechner aus eine Verbindung zu dem Rechner her, auf dem der Service läuft. Beobachten Sie dabei wieder die Netzwerkverbindugen mit netstat.
Hinweis: Beim Aufruf von telnet müssen Sie jetzt die Adresse des Zielrechners

angeben. Die IP-Adresse eines Rechners erhalten Sie, indem Sie auf dem jeweiligen Rechner ipconfig aufrufen.

C) Testen Sie nun das Verhalten des Programms, wenn Sie gleichzeitig mehrere Verbindungen zu dem Service herstellen.

Hinweise:

- ✗ Zur Bestimmung der aktuellen Uhrzeit und des Datums können Sie die folgende Klasse verwenden:

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class Clock {

  private static SimpleDateFormat timeFormatter =
     new SimpleDateFormat("kk:mm:ss");
  private static SimpleDateFormat dateFormatter =
     new SimpleDateFormat("dd.MM.yyyy");
  private static Date d = new Date();

  public static String date() {
     d.setTime(System.currentTimeMillis());
     return dateFormatter.format(d);
  }

  public static String time() {
     d.setTime(System.currentTimeMillis());
     return timeFormatter.format(d);
  }

}
```

Aufgabenblatt 3

Praktikum am 28.3.2012
Abgabe bis spätestens 3.4.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 3.1 - Client für Time-Service

Diese Aufgabe baut auf Aufgabe 2.1 auf. Es soll eine Client-Anwendung geschrieben werden, die über das Netzwerk Uhrzeit und Datum von einem Server abfragt. Auf dem Server läuft der Service Time-Service auf Port 75 (siehe Aufgabe 2.1).

Schreiben Sie eine Klasse mit dem Namen TimeServiceClient mit den beiden Klassenmethoden dateFromServer() und timeFromServer(). Beide Methoden haben je einen Parameter vom Typ String und einen Rückgabewert vom Typ String. Über den Parameter wird der Methode die IP-Adresse des Servers übergeben. Die Methoden sollen dann eine Verbindung zum Time-Service des Servers aufbauen, dessen Datum bzw. Uhrzeit abfragen und das Ergebnis als String zurückgeben.

Aufgabe 3.2 - Time-Service mit Multithreading

Diese Aufgabe baut auf Aufgabe 2.1 auf. Durch den Einsatz von Multithreading soll der Programmcode so geändert werden, dass mehrere Clients den Service gleichzeitig nutzen können.

Schreiben Sie eine neue Klasse TimeServiceMultithreaded, die den gleichen Service erbringt wie TimeService, jedoch mit dem Unterschied, dass zu dem Service mehrere Verbindungen gleichzeitig aufgebaut werden können. Dazu soll das Programm für jeden eingehenden Verbindungswunsch einen eigenen Thread starten, der diese Verbindung betreibt. Testen Sie das Programm, indem Sie mit telnet mehrere Verbindungen gleichzeitig aufbauen.

Tipp:

- ✗ Wenn Sie das Programm im Debug-Modus starten, können Sie in Eclipse im Debug-View die einzelnen gestarteten Threads beobachten.

Aufgabenblatt 4

Praktikum am 4.4.2012
Abgabe bis spätestens 10.4.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 4.1 - Http-Client

In dieser Aufgabe soll ein einfacher Http-Client realisiert werden, der mit Hilfe elementarer Netzwerk-Kommunikation einen Get-Zugriff auf einen vorgegebenen URL durchführt.

Schreiben Sie eine statische Methode *get*, der als Parameter ein URL (Typ: String) übergeben wird. Die Methode soll eine Verbindung mit dem im URL angegebenen HTTP-Server aufbauen. Anschließend soll die Methode einen HTTP-Get-Request an den Server schicken und dann die Response auf dem Bildschirm ausgeben.

Der Aufbau eines Get-Requests ist in nachfolgendem Beispiel dargestellt. Der Request setzt sich aus drei Zeilen zusammen, wobei die letzte Zeile eine Leerzeile ist. Die erste Zeile enthält den URL. Hier muss der URL eingefügt werden.

```
GET http://omserver.dm.fh-furtwangen.de/op/testseite.html HTTP/0.9
Host:

```

Die Response setzt sich aus einem Header mit technischen Informationen und dem Inhalt der Web-Seite zusammen. Die erste Leerzeile in der Response kennzeichnet die Trennung zwischen dem Header und dem Inhalt der Web-Seite.

Beispiel:

```
HTTP/1.1 200 OK
Date: Mon, 25 Aug 2003 13:43:29 GMT
Server: Apache/1.3.27 (Linux/SuSE) PHP/4.3.1
Last-Modified: Wed, 20 Aug 2003 09:53:04 GMT
ETag: "29f1c-187-3f434500"
Accept-Ranges: bytes
Content-Length: 391
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
 content="text/html; charset=ISO-8859-1">
  <title>Online-Programmierung Testseite</title>
</head> ...
```

Hinweise:

- Es wird empfohlen, wie oben dargestellt, den Request als HTTP/1.0-

Request zu schicken. In HTTP/1.0 gibt es im Gegensatz zu HTTP/1.1 noch keine "persistent connections", es wird also bei jedem Request immer nur ein Request bearbeitet. Nachdem der Server die Response verschickt hat, schließt er die Verbindung. Hat der Client alle Zeilen der Response ausgelesen, so führt ein weiterer readLine()-Aufrufe des Clients schnell zu einem null-Wert.

✗ Zu einem String s, der einen URL enthält, erhält man mit
(new URL(s)).getHost()
und
(new URL(s)).getPort()
die in dem URL enthaltene Adresse des Hosts und die Port-Nummer. Wurde in der URL keine Port-Nummer angegeben, so liefert getPort() den Wert -1 zurück. In diesem Fall muss beim Verbindungsaufbau als Port-Nummer der Defaultwert 80 angegeben werden.

✗ Die erste Zeile der Response, ein Teil des Headers, enthält eine Zahl, die man als Response Code bezeichnet. Der Response Code steht unmittelbar nach dem ersten Leerzeichen. Dem Response Code kann entnommen werden, ob der URL-Request erfolgreich bearbeitet werden konnte oder nicht und gegebenenfalls auch die Ursache für den Fehler.

Response Code	*Bedeutung*
200	OK Der Zugriff auf den im URL angegebenen Daten war erfolgreich
404	Not found Zu dem im URL angegebenen Pfad stehen auf dem Server keine Daten bereit.

Aufgabe 4.2 - HTTP-Client Tools

Diese Aufgabe baut auf Aufgabe 4.1auf.

A) Schreiben Sie eine Methode, die bestimmt, ob ein URL existiert.
(ein Parameter vom Typ String, Rückgabewert boolean)

B) Schreiben Sie eine Methode, die zu einem URL den Inhalt in Form eines Strings zurückgibt.

C) Schreiben Sie eine Methode, die zu einem URL das Datum der letzten Änderung zurückgibt.

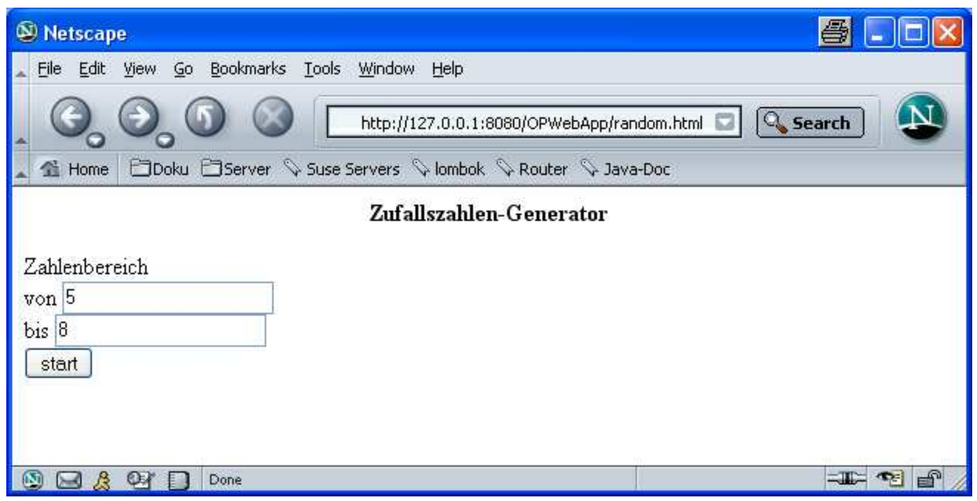
Aufgabenblatt 5

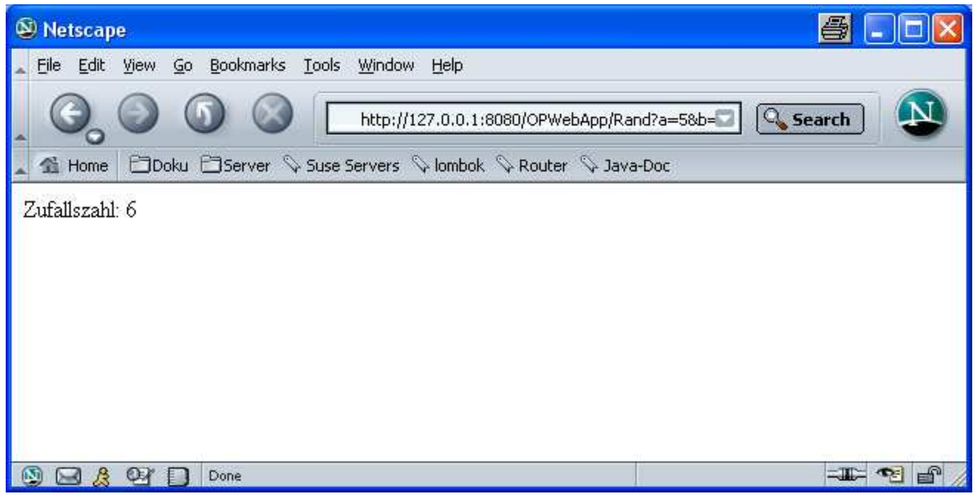
Praktikum am 11.4.2012
Abgabe bis spätestens 17.4.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 5.1 - Einfache Web-Application

In dieser Aufgabe soll eine einfache Web-Applikation erstellt werden, die Zufallszahlen generiert. Der Benutzer soll über ein Formular die untere und die obere Grenze eines Zahlenbereichs eingeben können. Durch das Abschicken des Formulars wird der URL eines Servlets aufgerufen. Das Servlet bestimmt eine ganzzahlige Zufallszahl im angegebenen Zahlenbereich und stellt die Zufallszahl in einer dynamisch erzeugten Web-Seite dar.





A) Legen Sie ein Tomcat-Projekt mit dem Namen OPWebApp an.

B) Implementieren Sie in dem Projekt OPWebApp eine Servlet-Klasse mit dem

Namen Random, der als Request-Parameter zwei ganzzahlige Werte a und b übergeben werden. Das Servlet soll eine Zufallszahl bestimmen, die größer oder gleich a ist und kleiner oder gleich b. Diese Zufallszahl soll in der Response in Form einer HTML-Seite ausgegeben werden.

Konfigurieren Sie das Servlet in WEB-INF/web.xml so, dass es über den Pfad "Rand" angesprochen werden kann. Testen Sie das Servlet mit dem Aufruf: http://127.0.0.1:8080/OPWebApp/Rand?a=7&b=8

Hinweise:

- ⇨ In Tomcat-Projekten befindet sich der Source-Code der Java-Klassen in dem Verzeichnis src. Legen Sie dort die Klasse Random.java an.
- ⇨ Die übersetzten Klassen, die .class-Dateien, befinden sich bei Tomcat-Projekten im Unterverzeichnis WEB-INF\classes.
- ⇨ Die statische Methode Math.random() ermittelt eine Zufallszahl im Bereich zwischen 0 (einschließlich) und 1 (ausschließlich) als double-Wert.

C) Realisieren Sie eine Web-Seite mit dem Namen random.html, in der in einem Formular die Grenzen des Zahlenbereichs a und b eingegeben werden können. Das Abschicken des Formulars soll dazu führen, dass das Servlet mit den eingegebenen Werten aufgerufen wird. (siehe Screenshot)
Verwenden Sie hier zunächst die GET-Parameter-Übermittlung.

D) Modifizieren Sie die Formularseite random.html, indem sie zur Parameter-Übermittlung den Modus "POST" anstelle von "GET" angeben und testen Sie Ihre Anwendung erneut mit dem Browser.

E) Erstellen Sie eine HTML-Seite mit dem Namen invalidvalues.html und dem Inhalt

"Parameterwerte sind ungültig."

Diese Seite soll angezeigt werden, wenn der Benutzer ungültige Werte eingegeben hat. Die Werte seine ungültig, wenn a oder b keine ganze Zahl ist oder wenn a größer als b ist. In diesem Fall soll der Browser keine Zufallszahl anzeigen, sondern das Servlet soll den Browser auf die Seite invalidvalues.html umleiten.

Hinweis:

- ⇨ Verwenden Sie hierzu die Methode response.sendRedirect()

Aufgabenblatt 6

Praktikum am 18.4.2012
Abgabe bis spätestens 24.4.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 6.1 - HTTP-Sessions

Diese Aufgabe baut auf Aufgabe 5.1 auf.

A) Erweitern Sie die Funktion des Servlets so, dass es nicht nur eine Zufallszahl ausgibt, sondern auch auch ausgibt, wie oft die Methode Zufallszahl schon innerhalb dieser Session aufgerufen wurde. Testen Sie das Servlet mit mehreren gleichzeitigen Sessions, indem Sie von mehreren Browsern gleichzeitig auf die Web-Application zugreifen.

Tipps:

- ⇨ Speichern Sie die Anzahl der Aufrufe in einem Session-Attribut mit dem Namen "anzahl".
- ⇨ In Session-Attributen können nur Objekte gespeichert werden, jedoch keine einfachen Datentypen wie beispielsweise int.
- ⇨ Verwenden Sie deshalb ein Instanz der Klasse Integer. Die Klasse Integer ist ein "Wrapper" für einen int-Wert. Mit

```
Integer k = new Integer(x);
```

können Sie aus einem int-Wert x ein Objekt k vom Typ Integer erzeugen. Das Objekt k können Sie zu Object casten und dann als Session-Attribut speichern (setAttribut). Später, in den nachfolgenden Requests, können Sie das Session-Attribut dann wieder auslesen(getAttribute), es anschließend zurück in ein Integer-Objekt casten, und Sie erhalten dann mit

```
k.intValue()
```

den Inhalt des Session-Attributs als int-Wert zurück.

B) Das in der vorigen Teilaufgabe verwendete Session-Management basiert auf Cookies und setzt voraus, dass Ihr Browser Cookies unterstützt. Testen Sie, wie sich die Web-Application verhält,

- ✗ wenn Sie in Ihrem Browser das vorhandene Cookie während des Betriebs löschen
- ✗ wenn Sie das Erstellen von Cookies blockieren und anschließend wieder deblockieren

Hinweis: Unter Firefox finden Sie die Cookie-Informationen unter dem folgenden Menüpunkt: Extras – Seiteninformationen – Sicherheit – Cookies anzeigen

Aufgabenblatt 7

Praktikum am 25.4.2012
Abgabe bis spätestens 22.5.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 7.1 - Kartenverkauf

In dieser Aufgabe soll eine Web-Anwendung realisiert werden, mit der der Kartenverkauf für eine Veranstaltung abgewickelt werden kann.

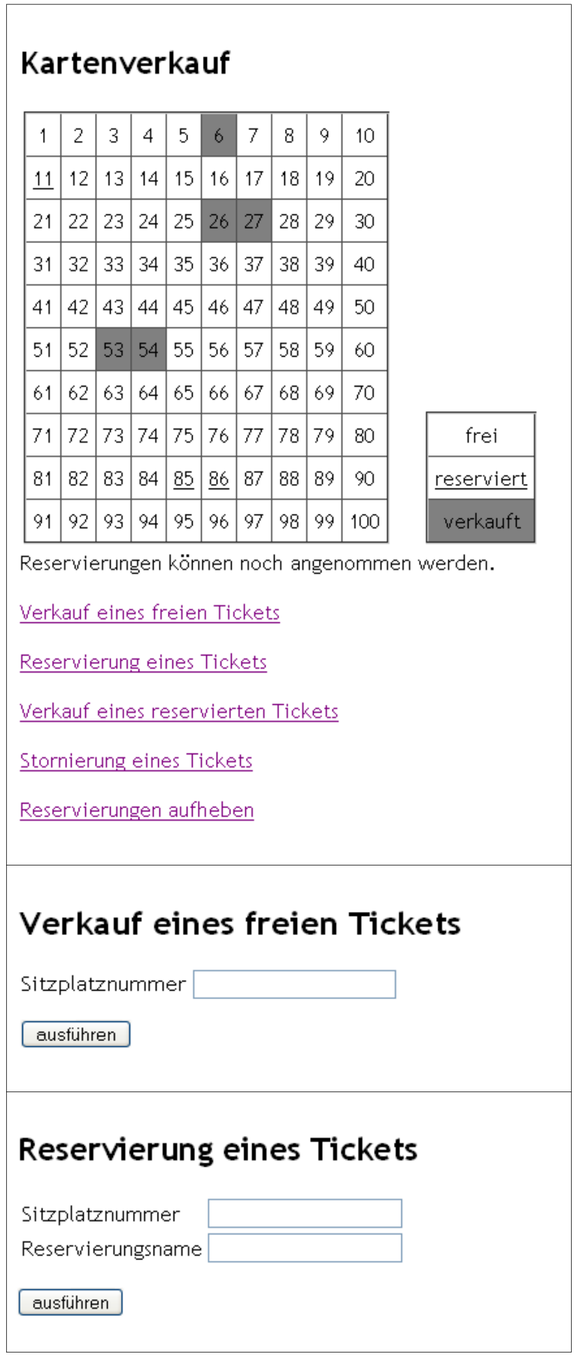
In der Veranstaltung soll es insgesamt 100 Sitzplätze geben, die fortlaufend mit den Nummern 1 bis 100 durchnummeriert sind. Mit dem Portal sollen die folgenden Transaktionen ausgeführt werden können:

- ⇨ **Ticket-Übersicht anzeigen**
  Es wird eine Liste mit allen Sitzplätzen ausgegeben. Zu jedem der Sitzplätze wird angegeben, ob der Sitzplatz frei, reserviert oder verkauft ist. Außerdem soll angegeben, ob Reservierungen noch angenommen werden können oder nicht.
- ⇨ **Verkauf eines freien Tickets**
  Eingabe ist eine Sitzplatznummer. Der Sitzplatz wird durch die Transaktion als verkauft gekennzeichnet. Ist ein Sitzplatz bereits als verkauft gekennzeichnet worden, so kann er nicht noch einmal verkauft werden. Eine weitere Verkaufstransaktion führt zu einer Fehlermeldung.
- ⇨ **Reservierung eines Tickets**
  Eingaben sind eine Sitzplatznummer und der Name der Person, für den der Sitzplatz reserviert werden soll. Der Sitzplatz wird als reserviert gekennzeichnet und es wird ihm der Name zugeordnet.
- ⇨ **Verkauf eines reservierten Tickets**
  Eingaben sind eine Sitzplatznummer und der Name der Person, die den Sitzplatz zuvor reserviert hat. Der Name, der bei der Reservierung angegeben wurde, gilt als „Passwort“. Der Verkauf eines reservierten Tickets ist nur dann erfolgreich, wenn der angegebene Name mit dem Namen identisch ist, der zuvor bei der Reservierung angegeben wurde.
- ⇨ **Stornierung eines Tickets**
  Eingabe ist eine Sitzplatznummer. Ein Ticket, das zuvor reserviert oder verkauft wurde, wird mit dieser Operation wieder zum allgemeinen Verkauf freigegeben.
- ⇨ **Reservierungen aufheben**
  Keine Eingabe. Alle Reservierungen werden gelöscht. Ferner bewirkt diese Transaktion, dass nachfolgend keine Reservierungen mehr angenommen werden. Anmerkung: Diese Operation soll nur einmal ausgeführt werden, und zwar üblicherweise eine definierte Zeit vor Veranstaltungsbeginn.

Die folgenden statischen Web-Seiten können Sie Ausgangsmaterial für das Portal verwenden.

index.html

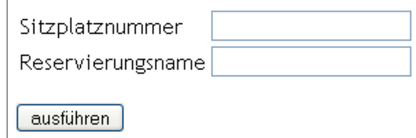
Startseite der Anwendung. Es wird eine aktuelle Ticket-Übersicht angezeigt und es wird angegeben, ob noch Reservierungen angenommen können oder nicht. Außerdem enthält die Seite Verweise auf die verschiedenen Transaktionen.



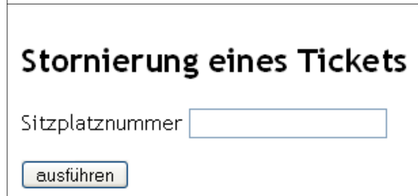
Verkauf_eines_freien_Tickets.html

Reservierung_eines_Tickets.html

Verkauf_eines_reservierten_Tickets.html



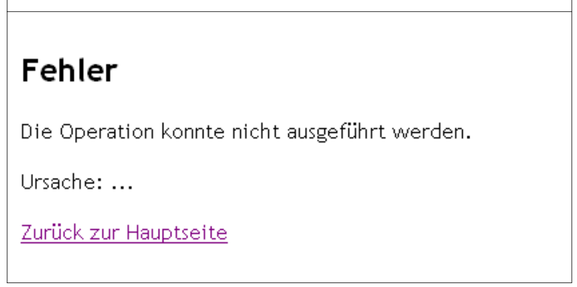
Stornierung_eines_Tickets.html



Reservierungen_aufheben.html

Operation_erfolgreich_ausgefuehrt.html

Fehler.html



Gehen Sie bei der Aufgabe bitte wie folgt vor:

Beginnen Sie zunächst damit, die Datenstrukturen und Operationen zu programmieren, und erstellen Sie erst danach die statischen und dynamischen Seiten der Web-Anwendung.

Definieren Sie zunächst eine Klasse mit dem Namen *Kartenverkauf*. Eine Instanz von dieser Klasse soll den aktuellen Zustand der Kartenreservierung repräsentieren und die Instanz soll alle oben beschriebenen Transaktionen in Form von Objektmethoden enthalten.

In einer Instanz vom Typ Kartenverkauf werden für alle Sitzplätze die Zustände gespeichert sowie bei reservierten Sitzen der Name. Außerdem wird in der Instanz die Information gespeichert, ob noch Reservierungen angenommen werden dürfen oder nicht. Die Klasse *Kartenverkauf* soll Objektmethoden enthalten, die die oben beschriebenen Transaktionen realisieren (Verkauf eines freien Tickets,...).

Kommt es in den Objektmethoden zu Ausnahmesituationen (Sitz bereits verkauft etc.), dann erzeugen Sie in der betreffenden Objektmethode an der jeweiligen Stelle im Programm eine dazu passende Exception und leiten Sie diese mit *throws* aus der Objektmethode heraus.

Verwenden Sie den Context-Listener *contextInitialized*, um eine Instanz von der Klasse *Kartenvekauf* zu erstellen und diese Instanz in einer Application-Scope-Attribut abzuspeichern. Greifen Sie aus den Servlets und JSPs auf dieses Application-Scope-Attribut zu.

Aufgabenblatt 8

Praktikum am 23.5.2012
Abgabe bis spätestens 12.6.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 8.1 - Kartenverkauf / Datenbank

Diese Aufgabe baut auf der Aufgabe „Kartenverkauf“ auf.

Die Geschäftsdaten der Web-Anwendung Kartenverkauf sollen nun persistent in einer Datenbank gespeichert werden. Die Web-Anwendung soll so modifiziert werden, dass die Daten auch nach einem Ausfall und einem erneuten Start der Web-Anwendung weiter zur Verfügung stehen.

Hinweise und Empfehlungen:

Da Sie in der Aufgabe „Kartenverkauf“ die Geschäftslogik in einer Klasse gekapselt haben, müssen Sie in dieser Aufgabe lediglich die Implementierung dieser Klasse ändern. Alles andere an der Web-Anwendung bleibt unberührt.

Zur Speicherung der Ticketdaten empfiehlt es sich eine Tabelle anzulegen, in der jeder Datensatz für einen Sitzplatz steht. Es gibt verschiedene Möglichkeiten, wie Sie die Tabelle anlegen und nutzen können. Sie können beispielsweise darin nur die reservierten und die verkauften Tickets speichern. Gibt es zu einem Sitzplatz noch keinen Datensatz, so bedeutet das, dass der Sitzplatz frei ist. Am Anfang ist dann die Tabelle leer.

Außerdem müssen Sie irgendwo persistent abspeichern, ob Reservierungen noch angenommen werden können oder nicht. Dazu benötigt man nur eine einzige boolesche Variable. Dazu sind wieder mehrere Möglichkeiten denkbar. Es reicht dazu beispielsweise eine Tabelle mit einer booleschen Spalte. In diese Tabelle fügt man nur eine Zeile ein und setzt deren (einzigen) Wert zunächst auf „false“.

Aufgabenblatt 9

Praktikum am 13.6.2012
Abgabe bis spätestens 30.6.2012

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen

Aufgabe 9.1 Kartenverkauf / RMI Service

In dieser Aufgabe soll die Web-Anwendung für den Kartenverkauf so erweitert werden, dass die elementaren Transaktionen auch über ein RMI-Objekt via entfernter Methodenaufrufe angesprochen werden kann.

Wandeln Sie die Klasse Kartenverkauf in ein RMI-Objekt um. Sorgen Sie dafür, dass das RMI-Objekt Kartenverkauf nach dem Start der Web-Anwendung unter einem bestimmten URL zur Verfügung steht. Verwenden Sie einen Context-Listener, um die Registry zu starten und das RMI-Objekt *Kartenverkauf* unter dem Namen “Kartenverkauf“ einzutragen.

Aufgabe 9.2 Kartenverkauf / RMI Clients

Schreiben Sie für jede Art von Kartenverkauf-Transaktion ein kleines Client-Programm.

- ⇨ TicketUebersichtAnzeigen.java
- ⇨ VerkaufEinesFreienTickets.java
- ⇨ ...

Jedes Client-Programm soll ein gewöhnliches Hauptprogramm mit einer *main*-Methode sein. Der URL des RMI-Objekts "Kartenverkauf" sowie die Eingabewerte für die Transaktion sollen dem Programm per Kommandozeilenparameter übergeben werden. Nachfolgend ein paar Beispielabläufe, die zeigen sollen, wie diese Programme genutzt werden können.

Starten des Programms TicketUebersicht

```
java TicketUebersichtAnzeigen rmi://141.28.122.37:1113/Kartenverkauf
```

Ausgabe dazu

```
1 - frei
2 - reserviert
3 - frei
4 - verkauft
...
```

Starten des Programms VerkaufEinesFreienTickets

```
java VerkaufEinesFreienTickets rmi://141.28.122.37:1113/Kartenverkauf 28
```

Ausgabe dazu

```
Operation erfolgreich. Sie haben das Ticket mit der Nummer 28 verkauft.
```

Aufgabe 9.3 Kartenverkauf / RMI Listener

Diese Aufgabe baut auf den vorangegangenen Aufgaben auf.

Per RMI-Callbacks soll ein "Push-Service" implementiert werden, über den die Clients fortlaufend über den aktuellen Stand des Verkaufs informiert werden.

Schreiben Sie zunächst ein RMI-Klasse *KartenverkaufListener*, mit einer einzigen Methode: *doIt*. Diese Methode soll drei Parameter haben: die Anzahl der freien Plätze, die Anzahl der reservierten Plätze und die Anzahl der verkauften Plätze. Die Methode soll diese Werte auf dem Bildschirm ausgeben (System.out.println).

Selbstverständlich brauchen Sie bei RMI-Klassen auch ein Interface. Nennen Sie das Interface von *KartenverkaufListener* *KartenverkaufListenerInterface*.

Ergänzen Sie die RMI-Klasse Kartenverkauf um eine Methode mit dem Namen *addKartenverkaufListener*. Parameter der Methode ist ein Objekt vom Typ KartenverkaufListenerInterface. Mit dieser Methode kann ein Client einen *KartenverkaufListener* an das RMI-Objekt Kartenverkauf übergeben. Auf der anderen Seite entstehen beim Aufruf der Methode ein Proxy auf das übergebene Objekt. Diese von Clients übergebenen Proxies werden innerhalb des Kartenverkauf-Objekts in einem Sammelbehälter gesammelt (z.B. Vector). Das *Kartenverkauf*-Objekt soll die *doIt* Methoden aller gesammleten Proxies aufrufen, sobald sich an der Anzahl der freien, reservierten oder verkauften Tickets etwas ändert. Dabei kommt es zu Callbacks: Der Service führt entfernte Methodenaufrufe bei den Clients aus.

Aufgabenblatt 10

Praktikum am 15.1.2009
Abgabe bis spätestens 26.11.2006

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 10.1 - Lotto

In dieser Aufgabe soll eine Web-Anwendung realisiert, mit der eine Lottoziehung durchgeführt werden kann.

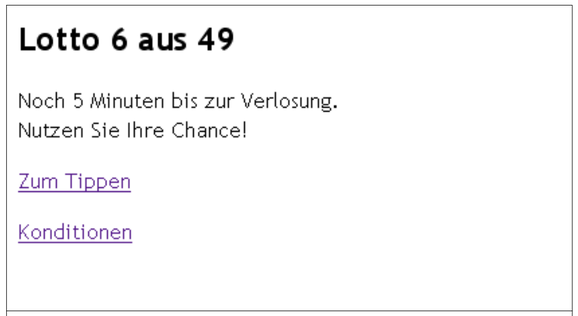
Vor der Inbetriebnahme der Anwendung wird ein Zeitpunkt für die Ziehung festgelegt. Die Web-Anwendung wird entsprechend konfiguriert. Nach dem Start der Web-Anwendung können die Benutzer des Web-Portals über einen Browser Tipps abgeben. Mit jedem Tipp werden neben der gewählten Zahlenkombination (6 aus 49) auch der Name des Benutzers, seine Bankverbindung und seine E-Mail-Adresse gespeichert. Zu dem angegebenen Zeitpunkt führt die Web-Anwendung dann die Ziehung aus: Es werden 6 Zufallszahlen im Bereich von 1 bis 49 bestimmt. Von diesem Zeitpunkt an werden keine Tipps mehr angenommen. Unmittelbar nach der Ziehung gibt die Web-Anwendung eine Teilnehmerliste auf der Konsole aus (System.out.print). Dabei werden für jeden Teilnehmer seine Benutzerdaten (Name, Bankverbindung, E-Mail-Adresse), sein Tipp und die Anzahl der "Richtigen" ausgegeben.

Web-Seiten

Für die Lotto-Anwendung wurden bereits mehrere Seiten entworfen. Diese Seiten sollen die Ausgangsbasis für Web-Appliction sein, die auf Servlet- und JSP-Basis realisiert werden soll.

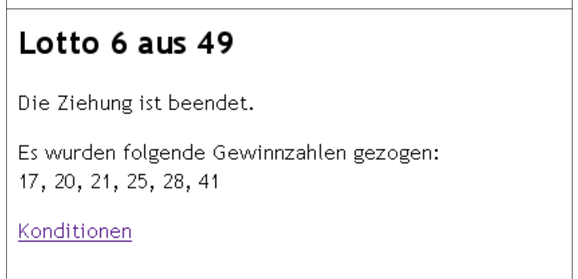
index_in_progress.html

Diese Seite erscheint als Startseite solange, bis es zur Ziehung kommt.



index_game_over.html

Diese Seite erscheint als Startseite nach der Ziehung.



input.html

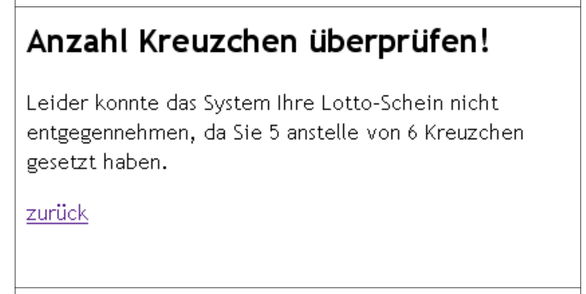
Über diese Seite gibt der Benutzer seinen Tipp ab.

confirm.html

Nach erfolgreicher Eingabe eines Tipps erscheint diese Bestätigung.

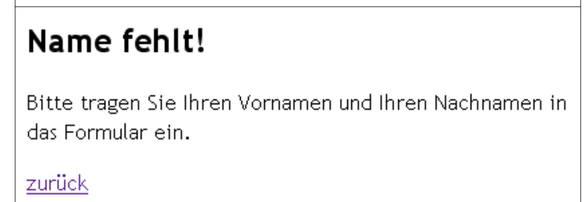
error_wrong_ticks.html

Seite, die nach fehlerhafter Eingabe der Daten erscheint.



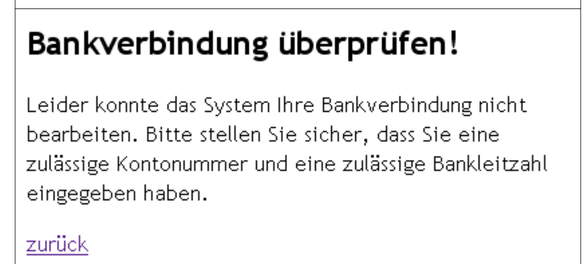
error_wrong_name.html

Seite, die nach fehlerhafter Eingabe der Daten erscheint.



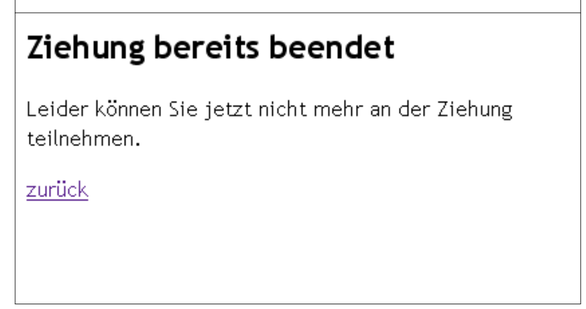
error_wrong_bank.html

Seite, die nach fehlerhafter Eingabe der Daten erscheint.



error_game_over.html

Diese Seite erscheint wenn der Benutzer nach der Ziehung versucht, auf die Eingabeseite zu kommen die Eingabe abzuschicken.



conditions.html

Von der Startseite aus immer zu erreichen.

Ausgabe auf der Konsole nach der Ziehung

```
Ergebnisse der Ziehung vom 20.11.2003 21:28:23


Gewinnzahlen: 18, 29, 34, 37, 43, 47
Teilnehmer: 3

Name: Kerstin Michelsen
Kontonummer: 345234
BLZ: 2452
E-Mail: kerstin@web.de
Tippreihe: 18, 23, 30, 31, 32, 33
Richtige: 1

Name: Mark Maier
Kontonummer: 23414
BLZ: 542555
E-Mail: mark@gmx.de
Tippreihe: 21, 29, 30, 31, 37, 47
Richtige: 3

Name: Michaela Lang
Kontonummer: 14234
BLZ: 234125
E-Mail: m.lang@fh-furtwangen.de
Tippreihe: 23, 24, 25, 26, 27, 28
Richtige: 0
```

Tipps

Starten Sie bei der Inbetriebnahme der Web-Anwendung einen "Lotto-Thread", der die Ziehung zur vorgegebenen Zeit ausführt. Der Lotto-Thread

- ⇨ bestimmt die verbleibende Dauer bis zur Ziehung,
- ⇨ blockiert sich dann mit sleep bis zur Ziehung,
- ⇨ bestimmt dann die Zufallszahlen
- ⇨ gibt schließlich mit System.out.print die Ergebnisse der Ziehung und die Teilnehmerliste auf der Konsole aus.

Starten Sie den Lotto-Thread aus einem ServletContextListener (Methode contextInitialized). Verwenden Sie einen Context Parameter für den Zeitpunkt der Ziehung, damit dieser Wert frei konfiguriert werden kann.

Verwenden Sie eine Klasse mit dem Namen Lotto, in deren Objektattributen die nachfolgenden Werte abgespeichert werden. Die Objektattribute sollen private sein. Zum Lese-/Schreibzugriff werden eigene Methoden deklariert, die synchronized sein sollen. Objektattribute von Lotto:

- ⇨ der Zeitpunkt der Ziehung
- ⇨ die gezogenen Zufallszahlen - so sie bereits vorliegen
- ⇨ eine Liste mit den bisher abgegebenen Tipps

Bilden Sie beim Start der Web-Application eine Instanz dieser Klasse und speichern Sie diese als Application-Attribut.

Zum Arbeiten mit Zeitpunkten sind folgende Klassen und Methoden hilfreich.

- ⇨ Enthält der String s ein Datum im Format "20.11.2003 21:28:23", dann erhalten Sie mit nachfolgender Anweisung ein Date-Objekt.

```
Date d = DateFormat.getDateTimeInstance().parse(s);
```

- ⇨ Ein Date-Objekt können Sie mit nachfolgender Anweisung in einen String im Format "20.11.2003 21:28:23" umwandeln.

```
String s = DateFormat.getDateTimeInstance().format(d);
```

- ⇨ Zu einem Date-Objekt kann mit der Objektmethode getTime() die Anzahl der Millisekunden seit dem 1.1.1970 00:00 bestimmt werden (eine long-Zahl).

Aufgabenblatt 11

Praktikum am 19.6.2006
Abgabe bis spätestens 2.7.2006

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 11.1 - Lotto - Datenbankanbindung

Diese Aufgabe baut auf Aufgabe 10.1 auf.

In der Lotto-Web-Application werden während des Betriebs zwei Arten von Daten erzeugt, die für das Geschäft des Betreibers von großer Bedeutung sind:

- ⇨ die abgegebenen Tipps
- ⇨ die bei der Ziehung bestimmten Lottozahlen

Ihre Lotto-Web-Application soll nun so modifiziert werden, dass diese Daten auch nach einem Ausfall der Web-Application noch zur Verfügung stehen und die Lotto-Web-Application nach einem Neustart ihre Arbeit fortsetzen kann. Ergänzen Sie dazu Ihren Programmcode so, dass die oben beschriebenen Daten in einer Datenbank persistent abgespeichert werden.

Tipp

Wenn Sie den Tipp aus Aufgabe 10.1 mit der Lotto-Klasse befolgt haben, so bedeutet die Anbindung an die Datenbank lediglich, dass Sie die in der Lotto-Klasse gespeicherten Werte nicht in Objektattributen sondern in der Datenbank ablegen. Sie müssen dazu die entsprechenden Zugriffsmethoden ändern.

Aufgabenblatt 12

Praktikum am 3.7.2006
Abgabe bis spätestens 9.7.2006

Verteilte Anwendungen 1
Sommersemester 2012
Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler

Aufgabe 12.1 - Sitzplatzreservierung

In dieser Aufgabe soll eine einfaches System zur Sitzplatzreservierung für Veranstaltungen wie Theater, Kino etc. realisiert werden. Das System soll als Client/Server-Lösung auf der Basis von RMI implementiert werden.

Die Anzahl der Sitzplätze *n* wird dem Service beim Start als Kommandozeilenparameter übergeben. Der Service verwaltet eine Liste der Sitzplätzen, bei der für jede Sitzplatznummer gespeichert wird, ob und gegebenenfalls von wem der Sitzplatz bereits reserviert.

⇨ Empfehlung: Verwenden Sie einen String-Array der Länge n, um darin für jeden Sitzplatz den Namen des Reservierenden zu speichern. Alle Array-Elemente werden zu Anfang auf null (nicht reserviert) gesetzt.

Der Service soll drei Funktionen anbieten:

Reservierung durchführen	Übergeben wird eine Sitzplatznummer im Bereich von 0 bis n-1 und ein Name (String). Der Rückgabewert ist boolesch und gibt an, ob die Reservierung erfolgreich war, d. h. ob der Platz zum Zeitpunkt der Reservierung noch frei war.
Belegungsliste abfragen	Keine Parameter. Der Client erhält als Rückgabewert die Liste mit dem aktuellen Stand der Reservierung (String-Array).
Reservation-Listener anmelden	Der Client übergibt dem Service einen Listener. Der Listener meldet an den Client fortlaufend zurück, wie viele Sitzplätze noch frei sind (int).

A) Realisieren Sie eine RMI-Objekt-Klasse mit dem Namen *ReservationService*, das zunächst nur die ersten beiden Funktionen (Reservierung durchführen, Belegungsliste abfragen) realisiert.

B) Schreiben Sie ein Programm, das den Service startet: Starten der RMI-Registry und Binden einer Instanz von *ReservationService*.

C) Implementieren Sie ein Client-Programm, mit dem man eine Reservierung durchführen kann. Kommandozeilenparameter: URL des *ReservationService*-Objekts, Sitzplatznummer, Name. Das Programm gibt aus, ob die Reservierung erfolgreich war oder nicht.

D) Implementieren Sie ein Client-Programm, das die aktuelle Belegungsliste

abfragt und auf dem Bildschirm ausgibt. Kommandozeilenparameter: URL des *ReservationService*-Objekts.

E) Realisieren Sie eine RMI-Objekt-Klasse mit dem Namen *ReservationListener*. Die Klasse enthält nur eine Objektmethode mit dem Namen *message*, die immer dann aufgerufen werden soll, wenn der Service die Anzahl der freien Sitzplätze an den Client meldet. Die Methode *message* ist denkbar einfach: Sie hat einen Parameter vom Typ int (Anzahl der noch freien Sitzplätze), und sie gibt diesen Wert auf dem Bildschirm aus.

F) Legen Sie in *ReservationService* ein zusätzliches Attribut vom Typ *Vector* an, um dort *RerservationListener*-Objekte zu speichern. Fügen Sie in der Klasse *ReservationService* die Methode *addRerservationListener* hinzu. Parameter: ein *ReservationListener_Interface*. Rufen Sie innerhalb der Methode *addRerservationListener* zunächst den *ReservationListener* auf und legen Sie den *ReservationListener* dann in dem Vector ab. Erweitern Sie die Reservierungsmethode von ReservationService so, das am am Ende jeder Reservierung alle im *Vector*-Objekt gespeicherten *ReservationListener* aufgerufen werden.

G) Implementieren Sie ein Client-Programm, das fortlaufend die Anzahl der noch freien Sitzplätze auf dem Blidschirm ausgibt. Kommandozeilenparameter: URL des *ReservationService*-Objekts.