Fakultät IV

Klausur MPGI 4
26.02.2010

Kao
Eitz / Warneke

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 90 Minuten

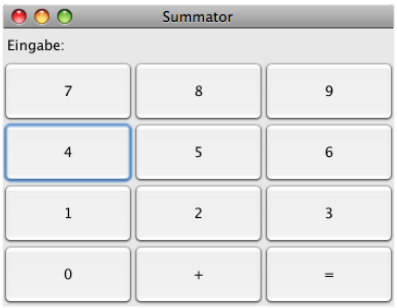
- Es ist ein doppelseitig handbeschriebenes DIN-A4 Blatt als Hilfsmittel zugelassen, wenn es Ihren Namen und Ihre Matrikelnummer enthält. Nicht zugelassen sind elektronische Hilfsmittel, wie z. B. Taschenrechner, Handys oder Laptops.
- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseiten. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn Sie zusätzliche, von uns ausgegebene Blätter verwenden, geben Sie unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet! Streichen Sie gegebenenfalls eine Lösung durch!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können nicht gewertet werden!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Sie können die Aufgaben in einer beliebigen Reihenfolge bearbeiten.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

Aufgabe	Punkte	erreicht
1	18	
2	18	
3	18	
4	18	
5	18	

1. Aufgabe (18 Punkte): Graphische Benutzerschnittstellen

1.1. LayoutManager (6 Punkte) Nennen Sie drei LayoutManager aus dem Sun JDK und erläutern Sie mit jeweils einem Satz die von diesen erzeugte Anordnung der eingefügten Komponenten.

1.2. Entwicklung einer graphischen Benutzerschnittstelle (8 Punkte) Entwickeln Sie unter Einsatz passender LayoutManager eine graphische Benutzerschnittstelle, welche einen einfachen Taschenrechner realisiert (siehe Abbildung). Das Layout der Benutzoberfläche soll sich dabei flexibel an die vom Benutzer gewünschte Fenstergröße anpassen. Ergänzen Sie für diese Aufgabe das vorgegebene Codeskelett.




```
import javax.swing.*;
import java.awt.*;

public class Summator
{
  String[] btnText = {"7", "8", "9", "4", "5", "6", "1", "2", "3", "0", "+", "="};
  JLabel labelEingabe;

  public Summator() {









  }

  public static void main(String[] args) {
    Summator summator = new Summator();
  }
} // Ende Klasse Summator
```

1.3. Event Handling (4 Punkte) Vervollständigen Sie das unten angegebene Codefragment dahingehend, dass der Text "Button geklickt" auf der Konsole ausgegeben wird, sobald der Benutzer auf den Button `clickButton` klickt.

```
import java.awt.event.*;
import javax.swing.*;

public class ClickButton extends JFrame
{
  JButton clickButton = new JButton("Click me!");

  public ClickButton() {

    this.add(clickButton);
    this.setSize(100, 100);
    this.setVisible(true);









  }

  public static void main(String [] args) {
    new ClickButton();
  }
}
```

2. Aufgabe (18 Punkte): Eingabe/Ausgabe und Fehlerbehandlung

2.1. Datei einlesen (9 Punkte) Vervollständigen Sie die vorgegebene Methode **`public boolean read(String filename)`**, so dass sie die Textdatei mit dem Dateinamen `filename` einliest und verarbeitet. Die Textdatei enthält eine beliebige Anzahl von Textzeilen. Jede Zeile besteht aus einer Folge von natürlichen Zahlen zwischen 1 und 1000, getrennt durch ein Komma (,). Die Methode soll `true` zurückliefern, wenn die aufsummierten Zahlen jeder Zeile größer als 2000 sind. Ansonsten soll die Verarbeitung mit dem Rückgabewert `false` abgebrochen werden. Achten Sie darauf, dass die geöffnete Datei vor dem Verlassen der Methode geschlossen wird. Exceptions können hier ignoriert werden.

Hinweis: Sie können die Methode **`public String[] split(String regex)`** verwenden, die einen String an bestimmten Stellen zerteilt. An welchen Stellen das sein soll, legt man mittels des Parameters **`regex`** fest. Beispiel: "am,Komma,trennen".split(",") liefert folgendes Array zurück: {`"am","Komma","trennen"`}

```
import java.io.*

public class FileReader {

    public boolean read(String filename) throws IOException {









    }
}
```

2.2. Fehlerbehandlung allgemein (5 Punkte) Java kennt zur Fehlerbehandlung das Konzept der Exceptions. Worin liegt der Unterschied zwischen einer regulären `Exception`, einer `RuntimeException` und einem `Error`? Wie sollte man als Programmierer mit ihnen umgehen?

2.3. Fehlerbehandlung mit Exceptions (4 Punkte) Geben Sie alle möglichen Ausgaben an, die durch die Ausführung des unten angegebenen Programms entstehen können. Gehen Sie davon aus, dass das Array-Feld `args[0]` nicht `null` ist.

```
import java.io.*;

public class ExceptionApp {

  public static void main(String [] args) {

    try {
      System.out.print("A");
      FileInputStream f = new FileInputStream(args[0]);
      System.out.print("B");

    } catch(FileNotFoundException e) {
      System.out.print("C");
      return;
    } finally {
      System.out.print("D");
      System.out.println("");
    }
  }
}
```

Ausgabe:

3. Aufgabe (18 Punkte): Java Applets

3.1. Lebenszyklus (8 Punkte) Zeichnen Sie den Lebenszyklus eines Java Applets auf. Geben Sie für jeden Zustand des Lebenszyklus an, welche typischen Aktivitäten in ihm durchgeführt werden. Erläutern Sie außerdem kurz, wann Übergänge zwischen den Zuständen stattfinden.

3.2. Java Applets und Sicherheit (4 Punkte) Wo liegen potentielle Gefahren bei der Ausführung von Java Applets aus dem Internet? Welchen Schutz bietet die Java Virtual Machine? Wie lässt sich der Schutz steuern?

3.3. Java Applets und HTML (6 Punkte) Vervollständigen Sie das unten angegebene Java-Applet, so dass es sich als Teil der JAR-Datei `myApplet.jar` über das vorgegebene HTML-Fragment laden lässt. Das Applet soll dabei den Paramter "mpgi4" einlesen und innerhalb des Applets auf einem `JLabel` darstellen.

Hinweis: Zum Auslesen von HTML-Paramtern stellt die Klasse `Applet` die Methode `public String getParamter(String name)` zur Verfügung.

Das HTML-Fragment lautet:

```
<html>
  <body>
    <applet archive="myApplet.jar" code"de.tuberlin.cit.mpgi4.MyApplet.class" width=
       "400" height="100">
      <paramter name="mpgi4" value="Wert"/>
    </applet>
  </body>
</html>
```

```
package



import javax.swing.*;

public class
{









}
```

4. Aufgabe (18 Punkte): Threads

4.1. Grundlagen (4 Punkte) Erläutern Sie kurz, wann und wieso die Verwendung von Threads in Zusammenhang mit graphischen Benutzeroberflächen in Java sinnvoll ist. Gehen Sie außerdem kurz auf die Schwierigkeiten von Threads im Zusammenspiel mit Swing ein und deuten Sie ein Lösung an.

4.2. Threads entwickeln und ausführen (6 Punkte) Schreiben Sie eine Klasse `DemoThread`. In der `main`-Methode der Klasse sollen zwei Objekte vom Typ `DemoThread` erzeugt und anschließend gestartet werden. Beide Objekte sollen als separate Threads unabhängig voneinander laufen. Das erste Thread soll in einer Endlosschleife den Buchstaben `A` auf der Konsole ausgeben, das zweite Thread den Buchstaben `B`.

4.3. Synchronisation und Blockieren/Freigeben von Threads (8 Punkte) Gegeben sei die folgende Klasse `StringBuffer`, über die ein Producer- sowie ein Cosumer-Thread Objekte vom Typ `String` austauschen können sollen. Das Producer-Thread soll den Puffer dabei über die Methode `public void putString(String str)` befüllen, während der Consumer zum Abholen und Entfernen der String-Objekte die Methode `public String getString()` nutzen soll. Intern verwendet die Klasse `StringBuffer` dabei eine Datenstruktur mit dem Bezeichner `buf` vom Typ `Deque` zur Speicherung der String-Objekte. Die maximale Speicherkapazität dieser Datenstruktur soll bei zehn String-Objekten liegen.

Vervollständigen Sie den im Folgenden gegebenen Programmcode. Stellen Sie sicher, dass der Zugriff auf `buf` korrekt synchronisiert wird. Sorgen Sie außerdem dafür, dass das Producer- und Consumer-Thread korrekt blockiert bzw. freigegeben wird, wenn der interne Puffer `buf` leer ist bzw. seine Kapazitätsgrenze erreicht hat.

```
import java.util.ArrayDeque;
import java.util.Deque;

public class StringBuffer
{

  private Deque<String> buf = new ArrayDeque<String>();

  public void putString(String str) {









  }

  public String getString() {









  }
}
```

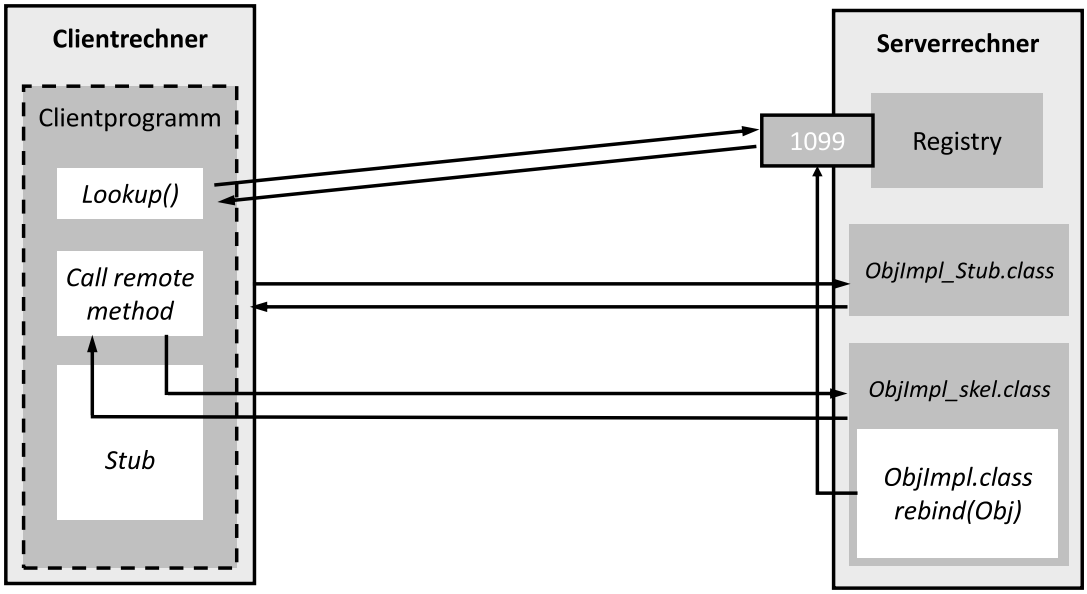
5. Aufgabe (18 Punkte): Sockets und Java RMI

5.1. Sockets (6 Punkte) Vervollständigen Sie die unten vorgegebene Methode `public String translate(String germanWord)` der Klasse `German2English`. Die Methode soll mittels des UDP Protokolls das übergebene deutsche Wort `germanWord` an den Server *www.mpgi4-translate.de*, Port 4444 schicken. Anschließend soll die Methode auf die Anwort des Servers warten, in der sich die englische Übersetzung des zuvor verschickten Wortes befindet. Die empfangene Antwort soll in einen String zurück konvertiert und als Rückgabewert der Methode benutzt werden.

Nehmen Sie zur Vereinfachung an, dass bei der Übertragung keine Packetverluste und Zeitüberschreitungen auftreten. Die Antwort des Servers wird eine Länge von 256 Bytes nicht überschreiten. Exceptions können hier ignoriert werden.

```
public class German2English {

  public String translate(String germanWord) throws IOException {









  }
}
```

5.2. Java RMI (7 Punkte) Im Folgenden finden Sie die Ihnen aus der Vorlesung bekannte Skizze zum Thema Java RMI. Nummerieren Sie die dargestellten Pfeile mit den Zahlen 1 bis 7 gemäß der korrekten Abfolge zur Registrierung und Durchführung eines entfernten Methodenaufrufs. Erläutern Sie zu jedem der sieben Punkte stichpunktartig, was in dem jeweiligen Schritt passiert.



5.3. Objektserialisierung (5 Punkte) Vervollständigen Sie das Codeskelett der Klasse `ObjectSerializer`. Die Klasse soll beim Aufruf der `main`-Methode ein Objekt von sich selbst erzeugen und dieses im Anschluss in die Datei "object.dat" serialisieren. Anschließend soll die Datei geschlossen werden. Exceptions können bei dieser Aufgabe ignoriert werden.

```
import java.io.*

public class ObjectSerializer
{

  public static void main(String [] args) throws IOException {









  }
}
```

Auszug aus der Java-API (1)

AbstractButton

```
class AbstractButton {
    void addActionListener(ActionListener l) // Adds an ActionListener to the button.
    ...
}
```

ActionListener

```
interface ActionListener {
    void actionPerformed(ActionEvent e)     // Invoked when an action occurs.
}
```

BufferedReader

```
class BufferedReader {
    BufferedReader(Reader in)               // Creates a buffering character-input
                                            // stream that uses a default-sized
                                            // input buffer.
    void close()                            // Closes the stream and releases any
                                            // system resources associated with it.
    void mark(int readAheadLimit)           // Marks the present position in the stream.
    int read()                              // Reads a single character.
    int read(char[] cbuf, int off, int len) // Reads characters into a portion of an array.
    String readLine()                       // Reads a line of text, null on EOF
    boolean ready()                         // Tells whether this stream is ready to be read.
    ...
}
```

Component

```
class Component {
    void setSize(int width, int height)     // Resizes this component so that it has
                                            // width width and height height.
    void setVisible(boolean b)              // Shows or hides this component
                                            // depending on the value of parameter b.
    ...
}
```

Container

```
class Container {
    Component add(Component comp)           // Appends the specified component to the
                                            // end of this container.
    Component add(Component comp, int index) // Adds the specified component to
                                            // this container at the given position.
    LayoutManager getLayout()               // Gets the layout manager for this container.
    Dimension getMaximumSize()              // Returns the maximum size of this container.
    Dimension getMinimumSize()              // Returns the minimum size of this container.
    Dimension getPreferredSize()            // Returns the preferred size of this container.
    void paint(Graphics g)                  // Paints the container.
    void setLayout(LayoutManager mgr)       // Sets the layout manager for this container.
    ...
}
```

DatagramSocket

```
class DatagramSocket {
    DatagramSocket()                        // Constructs a datagram socket and binds it to any
        available
                                            // port on the local host machine.
    DatagramSocket(int port)                // Constructs a datagram socket and binds it to the
        specified
                                            // port on the local host machine.
    void bind(SocketAddress addr)           // Binds this DatagramSocket to a specific address & port.
    void close()                            // Closes this datagram socket.
    void connect(InetAddress address, int port) // Connects the socket to a remote address for this
                                            // socket.
    void disconnect()                       // Disconnects the socket.
```

```
    InetAddress getInetAddress()           // Returns the address to which this socket is connected.
    InetAddress getLocalAddress()          // Gets the local address to which the socket is bound.
    int getPort()                          // Returns the port for this socket.
    boolean isClosed()                     // Returns whether the socket is closed or not.
    boolean isConnected()                  // Returns the connection state of the socket.
    void receive(DatagramPacket p)         // Receives a datagram packet from this socket.
    void send(DatagramPacket p)            // Sends a datagram packet from this socket.
    ...
}
```

DatagramPacket

```
class DatagramPacket {
    DatagramPacket(byte[] buf, int length)  // Constructs a DatagramPacket for receiving packets
                                            // of length length.
    InetAddress getAddress()                // Returns the IP address of the machine to which this
                                            // datagram is being sent or from which the datagram
                                            // was received.
    byte[] getData()                        // Returns the data buffer.
    int getLength()                         // Returns the length of the data to be sent
                                            // or the length of the data received.
    int getPort()                           // Returns the port number on the remote host
                                            // to which this datagram is being sent or from
                                            // which the datagram was received.
    void setAddress(InetAddress iaddr)      // Sets the IP address of the machine to which this
                                            // datagram is being sent.
    void setData(byte[] buf)                // Set the data buffer for this packet.
    void setData(byte[] buf, int offset, int length) //Set the data buffer for this packet.
    void setLength(int length)              // Set the length for this packet.
    void setPort(int iport)                 // Sets the port number on the remote host to which
                                            // this datagram is being sent.
    ...
}
```

Deque

```
interface Deque<E> {
    boolean add(E e)                        // Inserts the specified element into the
                                            // queue represented by this deque.
    E poll()                                // Retrieves and removes the head of the queue
                                            // represented by this deque.
    int size()                              // Returns the number of elements in this deque.
    ...
}
```

FileOutputStream

```
class FileOutputStream {
    FileOutputStream(File file)            // Creates a file output stream to write to
                                           // the file represented by the specified File
                                           // object.
    FileOutputStream(String name)          // Creates an output file stream to write to the
                                           // file with the specified name.
    void close()                           // Closes this file output stream and releases any system
                                           // resources associated with this stream.
    void write(byte[] b)                   // Writes b.length bytes from the specified byte
                                           // array to this file output stream.
    void write(int b)                      // Writes the specified byte to this file output stream.
    ...
}
```

FileReader

```
class FileReader {
    FileReader(File file)                  // Creates a new FileReader, given the File
                                           // to read from.
    FileReader(FileDescriptor fd)          // Creates a new FileReader, given the
                                           // FileDescriptor to read from.
    FileReader(String fileName)            // Creates a new FileReader, given the name
                                           // of the file to read from.
}
```

InetAddress

```
class InetAddress {
    byte[] getAddress()                                  // Returns the raw IP address of this
                                                         // InetAddress object.
    static InetAddress[] getAllByName(String host)       // Given the name of a host, returns
                                                         // an array of its IP addresses, based
                                                         // on the configured name service on the system.
    static InetAddress getByAddress(byte[] addr)         // Returns an InetAddress object given the raw
                                                         // IP address.
    static InetAddress getByAddress(String host, byte[] addr) // Create an InetAddress based on the
        provided
                                                         // host name and IP address No name service is
                                                         // checked for the validity of the address.
    static InetAddress getByName(String host)            // Determines the IP address of a host, given
                                                         // the host's name.
    String getHostName()                                 // Gets the host name for this IP address.
    static InetAddress getLocalHost()                    // Returns the local host.
    ...
}
```

Integer

```
class Integer {
    static int  parseInt(String s)      // Parses the string argument as a signed decimal integer.
    ...
}
```

ObjectOutputStream

```
class ObjectOutputStream {
    ObjectOutputStream(OutputStream out)        // Creates an ObjectOutputStream that writes to the
                                                // specified OutputStream.
    void close()                                // Closes the stream.
    void defaultWriteObject()                   // Write the non-static and non-transient fields of
                                                // the current class to this stream.
    void flush()                                // Flushes the stream.
    void write(byte[] buf)                      // Writes an array of bytes.
    void write(byte[] buf, int off, int len)    // Writes a sub array of bytes.
    void write(int val)                         // Writes a byte.
    void writeBoolean(boolean val)              // Writes a boolean.
    void writeByte(int val)                     // Writes an 8 bit byte.
    void writeChar(int val)                     // Writes a 16 bit char.
    void writeFields()                          // Write the buffered fields to the stream.
    void writeObject(Object obj)                // Write the specified object to the ObjectOutputStream.
    ...
}
```

String

```
class String {
    String(byte[] bytes)                                    // Constructs a new String by decoding the
         specified
                                                            // array of bytes using the platform's
                                                                default charset.
    byte[] getBytes()                                       // Encodes this String into a sequence of
        bytes using
                                                            // the platform's default charset, storing
                                                                 the result
                                                            // into a new byte array.
    ...
}
```