Fakultät IV

Klausur MPGI 4
01.04.2010

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	22	
3	14	
4	18	
5	18	

1. Aufgabe (18 Punkte): Graphische Benutzerschnittstellen

1.1. LayoutManager (6 Punkte) Zeichnen Sie das Fenster, das beim Aufruf des unten aufgeführten Programmcodes erzeugt wird.

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

public class NestedLayout extends JFrame {

  public NestedLayout() {
    super("Fenstertitel");

    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel();

    this.setLayout(new BorderLayout());

    panel1.setLayout(new GridLayout(1, 10));
    for(int i = 0; i < 10; i++) {
      panel1.add(new JButton(Integer.toString(i)));
    }
    this.add(panel1, BorderLayout.NORTH);

    panel2.setLayout(new FlowLayout());
    panel2.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
    panel2.add(new JLabel("Berlin"));
    panel2.add(new JLabel("Universitaet"));
    panel2.add(new JLabel("Technische"));

    this.add(panel2, BorderLayout.SOUTH);
    this.add(new JButton("MPGI 4"), BorderLayout.CENTER);

    this.setVisible(true);
    this.pack();
  }

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

1.2. Entwicklung einer graphischen Benutzerschnittstelle (8 Punkte) Entwickeln Sie unter Einsatz passender LayoutManager eine graphische Benutzerschnittstelle, welche ein einfaches Spielfeld für Schach realisiert. Ihr Code soll das in Abbildung 1 gezeigte Fenster reproduzieren. Das Layout der Benutzeroberfläche soll sich flexibel an die vom Benutzer gewählte Fenstergröße anpassen. Ergänzen Sie für diese Aufgabe das vorgegebene Codeskelett auf der nachfolgenden Seite.

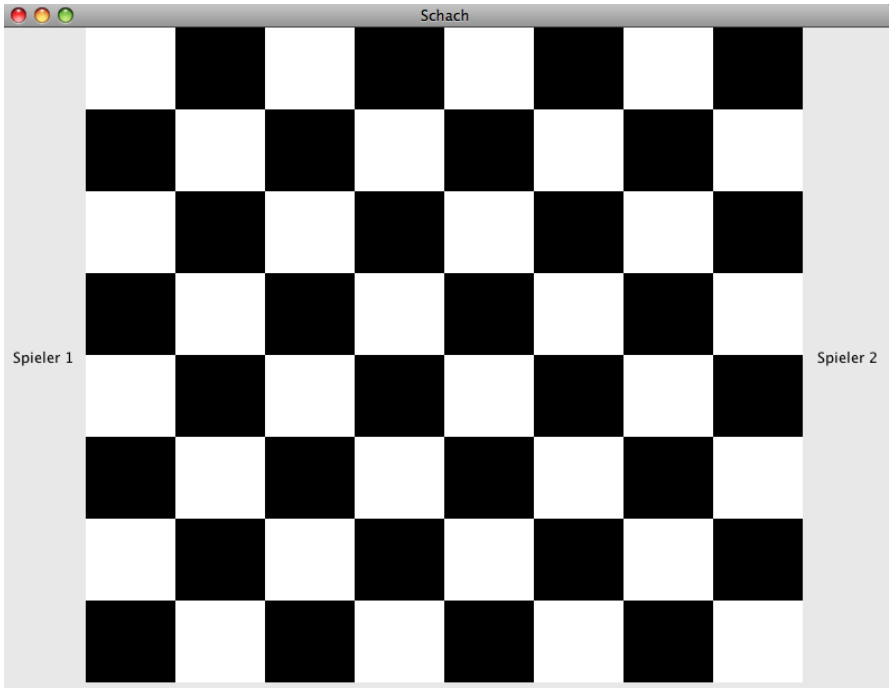


Abbildung 1: Zu implementierende graphische Benutzerschnittstelle

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

public class Chess
{

    public Chess() {









    }


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

1.3. Event Handling (4 Punkte) Vervollständigen Sie das unten angegebene Codefragment dahingehend, dass die aktuelle Position des Mauszeigers innerhalb des erzeugten Fensters mit ihrer x- und y-Koordinate auf der Konsole ausgegeben wird, wenn der Mauszeiger innerhalb des Fensters bewegt wird.

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

public class MouseWindow extends JFrame
{
  public MouseWindow() {

    this.setSize(300, 300);
    this.setVisible(true);









  }

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

2. Aufgabe (22 Punkte): Eingabe/Ausgabe und Fehlerbehandlung

2.1. Datei schreiben (8 Punkte) In dieser Aufgabe soll eine mp3 Playlist in eine Datei gespeichert werden. Vervollständigen Sie dafür die vorgegebene Methode **`public void write(List<Song> playlist, String filename)`**, so dass eine neue Datei mit dem Dateinamen `filename` erzeugt wird und die Songinformationen aus `playlist` zeilenweise in diese geschrieben werden. Ihr Dateiformat soll dabei dem unten angegebenen Beispiel entsprechen. Es sollen nur Songs abgespeichert werden, die eine Bitrate von mindestens 128 aufweisen.

Achten Sie sowohl darauf, alle möglicherweise auftretenden `IOExceptions` zu behandeln, als auch die resultierenden Schreibzugriffe auf die Festplatte zu minimieren.

```
class Song {
    String artist;
    String title;
    int bitrate;
}
```

Beispiel für das zu erzeugende Dateiformat:

```
artist_a title_a 160
artist_b title_b 256
artist_c title_c 128
```

```
public void write(List<Song> playlist, String filename) {









}
```

2.2. Datei lesen (4 Punkte) Vervollständigen Sie das folgende Java Programm, so dass die Datei mit dem Namen `filename` eingelesen und anschließend die CRC32-Prüfsumme der Datei auf der Konsole ausgegeben wird. Achten Sie darauf, dass der InputStream am Ende korrekt geschlossen wird. Exceptions können in dieser Aufgabe ignoriert werden.

```
public void readAndCheck(String filename) throws IOException{




}
```

2.3. Fehlerarten (6 Punkte) In Java wird zwischen einer `Exception`, einer `RuntimeException` und einem `Error` unterschieden. Nennen Sie für alle drei Fälle jeweils ein beispielhafte Situation, in der die jeweilige Art von Ausnahme geworfen werden sollte. Erläutern Sie zusätzlich jeweils kurz, ob das Programm im jeweiligen Fehlerfall noch fortgeführt werden sollte.

2.4. Fehlerbehandlung allgemein (4 Punkte) Kreuzen Sie die richtige(n) Aussage(n) an.

☐ Alle Ausnahmen erben von `Throwable`.
☐ Der Compiler erzwingt, dass alle Ausnahmen in einem `try/catch` Block behandelt werden.
☐ Der `finally` Block wird genau dann ausgeführt, wenn im `try` Block eine Ausnahme aufgetreten ist.
☐ Eine Ausnahme wird durch den ersten passenden `catch` Block behandelt.
☐ Im `finally` Block werden typischerweise Ressourcen freigegeben.
☐ Wird eine Ausnahme nicht behandelt, so terminiert der ausführende Thread.
☐ Um die Ausnahmebehandlung zu vereinfachen, bietet es sich an, nur Ausnahmen zu werfen, die von `RuntimeError` erben.
☐ Aus Sicherheitsgründen können selbst implementierte Ausnahmen nicht von `Error` erben.

3. Aufgabe (14 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 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 Parameter "mpgi4" einlesen und innerhalb des Applets auf einem `JLabel` darstellen.

Hinweis: Zum Auslesen von HTML-Parametern stellt die Klasse `Applet` die Methode `public String getParameter(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">
      <parameter name="mpgi4" value="Wert"/>
    </applet>
  </body>
</html>
```

```
package



import javax.swing.*;

public class
{









}
```

4. Aufgabe (18 Punkte): Threads

4.1. Grundlagen (4 Punkte) Wieso wird in der aktuellen Java API davon abgeraten, Threads von aussen mit der Methode `Thread.stop()` zu beenden, sondern stattdessen die Methode `Thread.interrupt()` zu verwenden? Welche Anforderungen ergeben sich dadurch für den Programmcode innerhalb des zu beendenden Threads? Gehen Sie insbesondere auf das spezielle Problem in Zusammenhang mit der Methode `Object.wait()` ein und über welches Sprachkonstrukt das Problem gelöst wird.

4.2. Threads entwickeln und ausführen (6 Punkte) Schreiben Sie eine Klasse `DemoThread`. In der `main`-Methode der Klasse sollen zehn Objekte vom Typ `DemoThread` erzeugt und anschließend gestartet werden. Alle zehn Objekte sollen als separate Threads unabhängig voneinander laufen. Jedes Thread soll in einer Endlosschleife die Zahl ausgeben, die der Startreihenfolge der Threads entspricht (Also das erste Thread eine 1, das zweite Thread eine 2, usw...).

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

  public void putString(String str) {









  }

  public String getString() {









  }
}
```

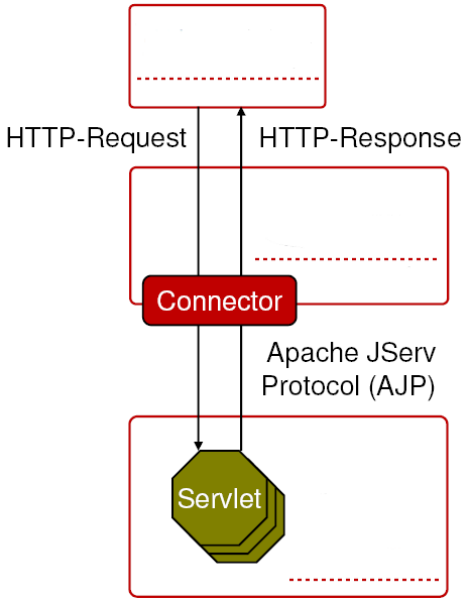
5. Aufgabe (18 Punkte): Sockets und Servlets/JSP

5.1. Sockets (6 Punkte) Gegeben ist der Server `www.exchange.biz`, der einen Währungsumrechnungsdienst auf dem Port 33000 über das TCP-Protokoll bereitstellt. Dieser rechnet USD in EUR um und umgekehrt. Ein verbundener Client kann den Dienst nutzen, indem er dem Server zwei Stringzeilen sendet. In der ersten Zeile steht der Wert, der umgerechnet werden soll, in der zweiten Zeile die Einheit. Nach Empfang der Daten sendet der Server im Gegenzug den umgerechneten Wert und die neue Einheit als einen String an den Client zurück.

Vervollständigen Sie die o.a. Implementierung der Klasse `ExchangeClient`, so dass der Währungsumrechungsdienst genutzt wird. Der umzurechnende Wert bzw. die Einheit werden der Methode `caculate` als Parameter `String value` und `String unit` übergeben. Das Ergebnis der Umrechnung soll auf der Konsole ausgegeben werden. Achten Sie darauf, dass Sie den Socket nach Empfang des Ergebnisses korrekt schließen. Exceptions können hier ignoriert werden.

```
public class ExchangeClient {

  public void convert(String value, String unit) throws IOException {









  }
}
```

5.2. Java Servlets (3 Punkte) Bennen Sie die drei Komponenten einer Standard-Servlet Architektur. Tragen Sie die Namen jeweils auf die gestrichelten Linien ein.



5.3. Java Server Pages (4 Punkte) Vervollständigen Sie die unten stehende Java Server Page dahingehend, dass die daraus erzeugte HTML-Seite für alle Zahlen $0 \leq i < 1000$ die Ausgabe "Das Quadrat von i ist i^2" enthält.

```
<html>
    <head>
        <title>Java Server Page</title>
    </head>
    <body>









    </body>
</html>
```

5.4. 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)

BufferedWriter

```
class BufferedWriter {
    BufferedWriter(Writer out)      // Creates a buffered character-output stream that
                                    // uses a default-sized output buffer.
    void close()                    // Closes the stream, flushing it first.
    void flush()                    // Flushes the stream.
    void newLine()                  // Writes a line separator.
    void write(String str)          // Writes a string.
    ...
}
```

CheckedInputStream

```
class CheckedInputStream {

    CheckedInputStream(InputStream in, Checksum cksum)  // Creates an input stream using
                                                        // the specified Checksum.
    void close()                                        // Closes this input stream and releases any
                                                        // system resources associated with the stream.
    Checksum getChecksum()                              // Returns the Checksum for this input stream.
    int read()                                          // Reads a byte, return -1 on end of stream
    int read(byte[] buf, int off, int len)              // Reads into an array of bytes.
    ...
}
```

Color

```
class Color {

    Color(int r, int g, int b) // Creates an opaque sRGB color with the specified
                               // red, green, and blue values in the range (0 - 255).
    static Color BLACK         // A color representing black
    static Color RED           // A color representing red
    static Color WHITE         // A color representing white
    ...
}
```

Component

```
class Component {
     void addMouseMotionListener(MouseMotionListener l) // Adds the specified mouse
                                                        // motion listener to receive
                                                        // mouse motion events from this 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.

    void setBackground(Color c)             // Sets the background color of this component.
    ...
}
```

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.
    ...
}
```

CRC32

```
class CRC32 {

    CRC32()             // Creates a new CRC32 object.
    long getValue()     // Returns CRC-32 value.
    ...
}
```

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.
    ...
}
```

FileInputStream

```
class FileInputStream {
    FileInputStream(String name)    // Creates a FileInputStream by opening a
                                    // connection to an actual file, the file
                                    // named by the path name name in the file system.
    ...
}
```

FileWriter

```
class FileWriter {
    FileWriter(String fileName) // Constructs a FileWriter object given a file name.
    ...
}
```

GridLayout

```
class GridLayout {
    GridLayout(int rows, int cols)  // Creates a grid layout with the specified number of rows and
        columns.
    ...
}
```

JFrame

```
class JFrame {
    JFrame()                    // Constructs a new frame that is initially invisible.
    JFrame(String title)        // Creates a new, initially invisible Frame with the specified title.
    Container getContentPane()  // Returns the contentPane object for this frame.
    void setContentPane(Container contentPane) // Sets the contentPane property.
    ...
}
```

JPanel

```
class JPanel {
    JPanel()                            // Creates a new JPanel with a double buffer
                                        // and a flow layout.
    JPanel(LayoutManager layout)        // Create a new buffered JPanel with the
                                        // specified layout manager.
    ...
}
```

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.
    ...
}
```

ObjectStreamWriter

```
class ObjectStreamWriter {

    OutputStreamWriter(OutputStream out)    // Creates an OutputStreamWriter that
                                            // uses the default character encoding.
    ...
}
```

Point

```
class Point {
    double  getX()  // Returns the X coordinate of this Point2D in double precision.
    double  getY()  // Returns the Y coordinate of this Point2D in double precision.
    ...
}
```

Socket

```
class Socket {
    Socket(String host, int port)   // Creates a stream socket and connects it to
                                    // the specified port number on the named host.
    InputStream getInputStream()    // Returns an input stream for this socket.
    OutputStream getOutputStream()  // Returns an output stream for this socket.
    void close()                    // Closes this socket.
    ...
}
```

MouseEvent

```
class MouseEvent {

    int getButton()           // Returns which, if any, of the mouse buttons has changed state.
    Point getPoint()          // Returns the x,y position of the event relative to the source component.
    ...
}
```

MouseMotionListener

```
interface MouseMotionListener {
    void mouseDragged(MouseEvent e)     // Invoked when a mouse button is pressed on a component
                                        // and then dragged.
    void mouseMoved(MouseEvent e)       // Invoked when the mouse cursor has been moved onto a component
                                        // but no buttons have been pushed.
}
```

Window

```
class Window {
    void pack()     // Causes this Window to be sized to fit the preferred size and
                    // layouts of its subcomponents.
    ...
}
```