Klausur: SS2011, Kurs: MB2, PR2 (Prof. Dr. Schimkat), Datum: Montag, 26.09.11

die beiden folgenden Kopfzeilen sind VOR der weiteren Bearbeitung der Klausur auszufüllen:

Vorname: **Nachname:** **Matrikelnummer:**

letzter Prüfungsversuch: O ja / ☒ nein

Aufgabe	1	2	3	4	5	6	Übung	Gesamt	LV-Note
Punkte max:	6	14	16	26	18	10	10	100	
Punkte	5	8	16	9	14	Ø	7	59	3,7

Hinweise:

1) Sind die die Kopfzeilen nicht vollständig ausgefüllt, führt dies zum Abzug von bis zu 6 Punkten.
2) Beschriften Sie alle verwendeten Blätter nur einseitig -> die Rückseite ist freizulassen.
3) Falls Sie Teile Ihrer Lösung auf Extrablätter schreiben:
   * Geben Sie auf dem Extrablatt Ihren Namen und die Aufgabennummer an.
   * Verwenden Sie für jede Aufgabe ein eigenes Extrablatt.
4) Auf zusätzlichen Blättern lassen Sie links, rechts, oben und unten mindestens je 2 cm Rand und schreiben Sie in die obere rechte Ecke Ihren Namen.
5) Werden die Hinweise 2-4 nicht beachtet, führt dies zum Abzug von bis zu 6 Punkten.
6) Es sind die in der Lehrveranstaltung eingeführten Fachbegriffe zu verwenden.
7) Die Lösungen sind programmiertechnisch sinnvoll, verständlich und den jeweiligen Konventionen entsprechend auszuführen. Unnötig komplizierte oder unsinnige Programmkonstrukte führen zu Punktabzug.
8) Bei Aufgaben mit Ankreuzen einer von zwei Antwortmöglichkeiten führt jede falsche Anworten zur Nicht-Wertung einer richtigen Antwort. Bei Unsicherheit sollten Sie einzelne Fragen ggf. nicht beantworten.
9) Kommunikation jeder Art ist nur mit dem Dozenten gestattet.

Aufgabe 1 (6 Punkte):

5

a) [2] Wieviele Threads und wieviele Sperren sind mindestens notwendig, damit ein Deadlock auftreten kann? 2

Antwort:
Es sind mindestens 2 Threads und mindestens 2 Sperren notwendig, damit ein Deadlock auftreten kann. ✓

b) [4 *vgl. Hinweis 8*] Geben Sie an, ob die folgenden Aussagen für ein Programm, in dem alle Teile von mehreren Threads ausgeführt werden können, richtig oder falsch sind: 3

	richtig	falsch	
Zuweisungen an lokale Variablen müssen synchronisiert werden	O	O	✓
Der Zugriff auf Immutables muss synchonisiert werden	O	☒	✓
Jeder Thread hat einen eigenen Heap	O	☒	✓
Jeder Thread hat einen eigenen Stack	☒	O	✓

8

Aufgabe 2 (14 Punkte):

a) [4 *vgl. Hinweis 8*] Beantworten Sie die folgenden Fragen:

Ist JTextField eine atomare Komponente?	O ja	O nein	✓
Sind LayoutManager atomare Komponenten ?	☑ ja	O nein	f
Kann ein JPanel-Objekt in ein anderes JPanel-Objekt eingefügt werden?	☑ ja	O nein	✓
Kann ein JFrame-Objekt in ein anderes JPanel-Objekt eingefügt werden?	☑ ja	O nein	f

b) [2] Welcher LayoutManager gibt jeder Kindkomponente eine gleichgroße Fläche? 2

GridLayout ✓

c) [2] Welcher Container ist als Zeichenfläche besonders geeignet? 2

JPanel ✓

d) [2] Wieviele Bereiche hat eine JSlpitPane? 2 ✓ 2

e) [4] Geben Sie möglichst genau an, woran man erkennt, daß die Objekte einer Klasse `SoundEvent`s aussenden können: 2

Die Klasse muss Methoden enthalten, mit denen man ihr einen (mehrere) Listener zuordnen (möglichst entfernen) kann.

public void π
20: addSoundListener (SoundListener sl) {} removeSoundListener (SoundListener sl) {
(✓)

Aufgabe 3 (16 Punkte *vgl. Hinweis 8*):

16

Geben Sie an, ob die folgenden Aussagen richtig oder falsch sind:

	richtig	falsch
XML-Tags müssen Attribute besitzen	O	☑
XML-Dokumente können mit CSS dargestellt werden	☑	O
XML-Dokumente müssen wohlgeformt sein	☑	O
Set ist Subtyp von Collection	☑	O
Objekte des Typs Collection sind sortierbar	O	☑
auf die Elemente eines Set kann per Index zugegriffen werden	O	☑
ein Set kann mehrere gleiche Elemente enthalten	O	☑
eine List kann mehrere gleiche Elemente enthalten	☑	O

✓

Vorgaben für die Aufgaben 4 und 5:
Für die GUI-basierte Steuerung eines Baukrans seien gegeben:

```
public class Befehl {
    public static final String MOVE_UP = "MOVE_UP";
    public static final String MOVE_DOWN = "MOVE_DOWN";
}

public interface Model ...  {
    public static final String MODEL_UPDATE = "MODEL_UPDATE";
    public abstract void move(boolean up);
    public abstract double getHeight();
    ...
}

public class ModelImpl implements Model {
    ...
}

und

public class View ...  {
    /** Abgleich mit den Modeldaten und anschliessendes Neuzeichen */
    public void update() {
      ...
    }
    public void paintGitter(Graphics g){ ... }
    public void paintLast(Graphics g){ ... }
    public void paintKran(Graphics g){ ... }
    ...
}
```

Die Klassen sind entsprechend dem im Unterricht besprochenen MVC-Muster miteinander gekoppelt. Das Model feuert bei Änderungen seines Zustands java.beans.PropertyChangeEvents, die vom View empfangen werden.

Aufgabe 4 (26 Punkte):

a) [8] Schreiben Sie eine Klasse `Controller` für einen zentralen Controller, der als Empfänger für Steuerungsereignisse dient, die durch Mausklicks auf entsprechende `JButton`s entstehen. Der `Controller` steuert aufgrund der Steuerungsereignisse das `Model` an und sorgt damit auch dafür, dass der `View` entsprechend aktualisiert wird.

b) [14] Schreiben Sie die Klasse `Comp`, die eine GUI-Komponente ist, die für die oben beschriebenen Steuerungsbefehle am oberen und unteren Rand jeweils einen `JButton` und in der Mitte eine Ansicht des Models enthält.
`Model`, `View` und `Controller` sollen im Konstruktor von `Comp` erzeugt werden. Achten Sie auf die richtigen Verbindungen zwischen `Model`, `View` und `Controller`. Dabei dürfen Sie annehmen, dass die notwendigen Konstruktoren in den drei Klassen existieren.

c) [4] Schreiben Sie (ohne den Rest der zugehörigen Klasse `View`) eine Instanzmethode `public void paintGitter(Graphics g)`, die im Bereich [0, 1000] ein Gitter aus horizontalen und vertikalen grünen Linien zeichnet. Die Gitterlinien sollen den Abstand haben, der in der Instanzvariablen `distance` vom Typ `int` gespeichert ist. Die Achsen selbst sollen blau gezeichnet werden (als einfache Linien ohne Einteilung oder Beschriftung).

Aufgabe 5 (18 Punkte):

Hinweis: Um Schreibaufwand zu sparen, dürfen Sie in dieser Aufgabe überall die Silbenfolge PropertyChange durch PC abkürzen (damit wird z.B. aus PropertyChangeEvent einfach PCEvent).

Geben Sie die Instanzvariablen und die Implementierung der Instanzmethoden (aber nicht den Rest der Klasse) an, die für Kopplung von Model und View durch java.beans.PropertyChangeEvents notwendig sind

a) [10] für die Klasse `ModelImpl`

Hinweis: Es sind hier nur die für die Kopplung von Model und mit notwendigen Instanzvariablen und -methoden anzugeben. Der Ablauf des Feuerns des PropertyChangeEvent muss nicht angegeben werden.

b) [6] für die Klasse `View`.

c) [2] Ergänzen Sie die Programmzeile

```
public class View ...  {
```

Aufgabe 6 (10 Punkte):

Geben Sie eine rekursiv arbeitende Implementierung für die binäre Suche in einem aufsteigend sortierten Array an:

```
/**
 * Falls der Wert value nicht im array vorhanden ist,
 * wird -1 zurückgegeben.
 * Falls der Wert value im array vorhanden ist,
 * wird sein Array-Index zurückgegeben.
 */
public static void indexOf(int value, int[] array)
```

int

Aufgabe 5a)

14

```
protected PropertyChangeSupport pcs;

    public void addPCListener (PCListener l) {
        pcs.addPCListener(l);
    }

    public void removePCListener (PCListener l) {
        pcs.removePCListener(l);
    }
```

10

5b

```
    public void propertyChangeEvent (PCEvent e) {
        if (e.getPropertyName().equals("MODEL_UPDATE")) {
            e.update();
        }
    }
```

2

5 c

```
    public class View extends JPanel implements PCListener { }
```

2