# Probeklausur Hinweise

- Es gibt 90 Punkte insgesamt
- Es liegt eine Bearbeitungszeit von 90 Minuten zugrunde. Die Bearbeitungszeit der Probeklausur wurde nicht überprüft/optimiert
- Diese Probeklausur demonstriert mögliche Aufgabentypen der Prüfungsklausur
- Die Aufgabentypen in der Prüfungsklausur können abweichen
- Die konkreten Aufgaben und Inhalte der Prüfungsklausur werden verändert
- Korrektheit und Vollständigkeit der Probeklausur werden nicht garantiert

## **Aufgabe 1.1** (5 Punkte)

Definieren Sie eine Klasse Addition, die zwei Gleitkommazahlen als private Attribute besitzt und die Methode ***berechneErgebnis()*** anbietet, welche eine Addition auf den beiden Attributen ausführt und das Ergebnis zurückgibt. Darüber hinaus gibt es einen Konstruktor der den Attributen Werte zuweist.

## **Aufgabe 1.2** (7 Punkte)

Implementieren Sie ein ausführbares Java Programm, das vom Nutzer zwei Zahlen über die Standardeingabe einliest, ein Objekt der Klasse Addition aus der Aufgabe 1.1 mit den beiden Zahlen erzeugt und das Ergebnis der Methode ***berechneErgebnis()*** auf der Konsole mit 2 Nachkommastellen ausgibt.

## **Aufgabe 2** (6 Punkte)

Definieren Sie für die folgende Zusammenhänge Klassen mit den entsprechenden Attributen. Definieren Sie keine Methoden.

A) Jedes Ei hat eine Nummer und eine Größe:

B) Ein Osterei ist ein bemaltes Ei, das zusätzlich den Titel eines Motivs besitzt:

C) Ein Ostereimaler hat einen Vornamen und einen Nachnamen:

D) Eine Maltechnik hat einen Beschreibungstext und einen Array von Farbnamen als Strings:

E) Ein sorbisches Osterei ist ein Osterei, das zusätzlich einen Ostereimaler und eine Maltechnik hat:

F) Ein Huhn hat eine Nummer und einen Liste der Eier, die es gelegt hat:

## **Aufgabe 3** (7 Punkte)

Analysieren sie die folgende Methode und Erläutern Sie pro Zeile welche Verarbeitung in dieser Methode durchgeführt wird. Vergessen Sie nicht die Methodensignatur zu erläutern. Was berechnet die Methode?

```
01  protected int compute( int[] numbers ) {
02      if (numbers == null) return Integer.MIN_VALUE;
03      if (numbers.length == 0) return Integer.MIN_VALUE;
04
05      int result = 1;
06      for (int i=0; i<numbers.length; i++) {
07          result *= numbers[i];
08      }
09      return result;
10  }
```

Zeile 1:

Zeile 2:

Zeile 3:

Zeile 5:

Zeile 6:

Zeile 7:

Zeile 9:

## **Aufgabe 4** (11 Punkte)

Es soll ein objekt-orientiertes Programm erstellt werden, dass für die Auswertung von Materialtests eingesetzt werden soll. Mehrere Materialien werden auf Ihre Zugfestigkeit untersucht. Jedes Material hat eine eindeutige Nummer und einen Namen. Jeder Test hat eine eindeutige Nummer und verweist auf alle im Test verwendeten Materialien. Während eines Tests wird die Zugfestigkeit pro Material in $N/mm^2$ bestimmt und gespeichert. Darüber hinaus soll es möglich sein für einen Test das Material mit der höchsten Zugfestigkeit zu bestimmen und auf dem Bildschirm auszugeben. Definieren Sie entsprechende Java-Klassen mit den dazugehörigen Attributen und Methoden. Denken Sie bei der Modellierung der Materialien an gute Datenkapselung.


## **Aufgabe 5.1** (11 Punkte)

Gestalten und implementieren Sie die Klasse GeoPoint, die folgende Eigenschaften und Verhalten realisiert:

- Attribute ***longitude***, die geographische Länge als Gleitkommazahl doppelter Genauigkeit
- Attribute ***latitude***, die geographische Breite als Gleitkommazahl doppelter Genauigkeit
- Attribute ***elevation***, die Höhe über dem Meeresspiegel als Gleitkommazahl doppelter Genauigkeit
- Konstruktor der Werte aller drei Attribute entgegen nimmt
- Konstruktor der nur Werte für Länge und Breite entgegennimmt
- Methode ***getVerticalDistance()*** berechnet die Höhendifferenz zweier Objekte der Klasse GeoPunkte und gibt das Ergebnis als Gleitkommazahl zurück.

Die Klasse selber soll nicht öffentlich, sondern nur innerhalb des eigenen Pakets sichtbar sein. Die Attribute sollen außerhalb der Klasse nicht sichtbar sein. Bieten sie für den Lesezugriff eine Alternative an. Stellen Sie sicher, dass der Rückgabewert der Methode ***getVerticalDistance()*** nicht negative ist. Verzichten Sie auf eine Fehlerbehandlung.

## **Aufgabe 5.2** (1 Punkt)

Entscheiden Sie sich bzgl. der Methode ***getVerticalDistance()*** für oder gegen die Implementierung einer Klassenmethode und begründen Sie Ihre Entscheidung.

## **Aufgabe 6.1** (5 Punkte)

Implementieren Sie eine Java-Methode, die als Parameter eine Matrix (in Form einer 2-dimensionalen Speicherstruktur vom Typ double) übergeben bekommen.

- Methode ***isDiagonal()*** überprüft, ob eine übergebene Matrix eine Diagonalmatrix ist, d.h. alle Nichtdiagonalenelemente sind 0: $a_{i,j} = 0$ für jedes i,j mit $i \neq j$. Die Methode gibt ***true*** zurück falls eine Diagonalmatrix vorliegt, ***false*** sonst.

Nehmen Sie an, dass die übergebene Matrix rechteckig ist, d.h. alle Zeilen der Matrix haben die gleiche Länge. Für die Methode ***isDiagonal()*** soll zusätzlich gelten, dass übergebene Matrizen quadratisch sind, d.h. die Anzahl Zeilen entspricht der Anzahl der Spalten. Beide Annahmen sollen nicht überprüft werden.

## **Aufgabe 6.2** (3 Punkte)

Modifizieren Sie die Methode isDiagonal aus Aufgabe 6.1 in der Art, dass überprüft wird, ob die übergebene Matrix quadratisch ist (Anzahl Zeilen = Anzahl Spalten). Wenn keine quadratische Matrix übergeben wurde, soll eine Exception geworfen werden (Verwenden Sie hierzu die Klasse Exception).

## **Aufgabe 6.3** (3 Punkte)

Zeigen Sie mit einem Block von Anweisungen, wie die in Aufgabe 6.2 erweiterte Methode mit der üblichen Fehlerbehandlung aufgerufen wird. Wenn die Exception auftritt, soll eine Fehlermeldung Ihrer Wahl auf dem Bildschirm ausgegeben werden.

## **Aufgabe 6.4** (3 Punkte)

Nennen Sie weitere Tests die im Rahmen der Validierung der Eingabeparameter der Methode isDiagonal durchgeführt werden sollten.

## **Aufgabe 7.1** (10 Punkte)

Gegeben sei eine einfache graphische Nutzerschnittstelle in Form der Klasse Taschenrechner:

```
public class Taschenrechner extends Frame {
	TextField textfield;
	Button button;
	Label label;

	public Taschenrechner() {
		setLayout(new GridLayout(2,3));

		add(new Label("Zahl:"));

		textfield = new TextField();
		add(textfield);

		button = new Button("Komm!");
		add(button);

		add(new Label("Quadrat:"));

		label = new Label();
		add(label);

	}

}
```

Modifizieren und ergänzen Sie die Klasse Taschenrechner so, dass folgende Reaktionen auf Ereignisse ausgeführt werden:

- wird die Maus in den Bereich des Button hineinbewegt, ändert sich die Beschriftung des Button auf „Mach!“
- wird die Maus aus dem Bereich des Button hinausbewegt, ändert sich die Beschriftung des Button auf „Komm!“

Eine Fehlerbehandlung soll nicht durchgeführt werden. Nutzen Sie folgende vereinfachte API-Dokumentation:

```
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e); }

public interface MouseListener extends EventListener {
    public void mouseClicked(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mousReleased(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);

public interface TextListener extends EventListener {
    public void textValueChanged(TextEvent e); }

public interface WindowListener extends EventListener {
    public void windowOpened(WindowEvent e);
    public void windowClosing(WindowEvent e);
    public void windowClosed(WindowEvent e);
    public void windowIconified(WindowEvent e);
    public void windowDeiconified(WindowEvent e);
    public void windowActivated(WindowEvent e);
    public void windowDeactivated(WindowEvent e); }

public class Button
    public String getLabel() {…}// gibt die Beschriftung des Button zurück
    public void setLabel(String label) {…} // ändert die Beschriftung des Button

public class Label
    public String getText() {…} // gibt den Text auf dem Label zurück
    public void setText(String label) {…} // ändert den Text auf dem Label

public class TextField
    public String getText() {…} // gibt den Text aus dem TextField zurück
    public void setText(String label) {…} // ändert den Text des TextField
```

## **Aufgabe 7.2** (2 Punkte)

Skizzieren Sie das Aussehen des Frames aus Aufgabe 7.1 und seiner Komponenten.

## **Aufgabe 8.1** (4 Punkte)

Skizzieren und beschreiben Sie den Aufbau und die Bestandteile einer einfach verketteten Liste. Skizzieren Sie graphisch anhand eines Beispiels mit mindestens zwei Listenelementen.

## **Aufgabe 8.2** (4 Punkte)

Benennen Sie alle notwendigen Arbeitsschritte (unter Verwendung von Pseudocode), um in die in Aufgabe 8.1 beschriebene Liste an der ersten Stelle ein neues Element einzufügen.

## **Aufgabe 8.3** (2 Punkte)

Nennen Sie mindestens 2 Vorteile einfach verketteter Listen gegenüber Arrays.

## **Aufgabe 9** (6 Punkte)

Führen Sie einen Code Review durch und finden Sie syntaktische und semantische Fehler sowie fehlende Validierungen von Eingabeparametern.

```
interface Record<T> {
    int compare( T arg0, D arg1 );

    double computeAvg( int[] transactions ) {
        double result = 0.0;
       for (int i=0; i<transactions.length; i++) {
           result += transactions[i] / transactions.length;
       }
       return result;
    }
}

abstract class Account {
    String accountHolder;
    boolean isAccountHolder( String arg0 ) {
             return this.accountHolder == arg0 ? true : false;
      }
}

public class SavingsAccount extends Account, List implements Record {
    double value;
    int number;

    SavingsAccount(String accountHolder, int number ) {
       super(String accountHolder);
        this.value = number;
    }

    static String getAccountHolder() {
        return accountHolder;
    }
}
```