universität ulm

1. Klausur zur Veranstaltung

OOP & PvS

im Sommersemester 2023

Prof. Dr. Matthias Tichy, Raffaela Groner, Joshua Hirschbrunn, Stefan Höppner

07.08.2023

Musterlösung

Aufgabe 1 - Wissensfragen **12 Punkte**

Kreuzen Sie zu jeder Frage die korrekte Antwortmöglichkeit an. Zu jeder Frage existiert nur eine korrekte Aussage.

a) Gegeben eine Variable a eines beliebigen Datentyps in Java. Welche der folgenden Aussagen über den folgenden Ausdruck ist korrekt?
Ausdruck: `(Object) a`

- ☐ Der Aufruf führt immer zu einer `ClassCastException`.
- ☒ Der Aufruf ist immer korrekt.
- ☐ Der Aufruf funktioniert nur auf Strings.
- ☐ Der Aufruf funktioniert nicht auf primitiven Datentypen.
- ☐ Der Aufruf funktioniert nur auf primitiven Datentypen.

b) Bei dem Methodenaufruf der Methode `exercise1(new int[] {5,6}, 3, "Hello")` in Java

- ☐ werden Referenzen auf alle Parameter übergeben.
- ☐ werden Kopien aller Werte übergeben.
- ☐ werden Kopien des 1. und 3. Parameter und eine Referenz auf den 2. übergeben.
- ☐ werden Kopien der 1. und 2. Parameter und eine Referenz auf den 3. übergeben.
- ☒ werden Referenzen auf die 1. und 3. Parameter und eine Kopie des 2. Parameter übergeben.

c) Gegeben eine Methode `int doStuff(int i)`. Welches der folgenden return-Statements ist zur Compilezeit **nicht** erlaubt?

- ☐ `return i;`
- ☐ `return --i;`
- ☐ `return (Integer) null;`
- ☒ `return "3";`
- ☐ `return 5;`

d) Welche Größe in bits hat der Datentyp `double` in Java?

- ☐ 1 bit
- ☐ 8 bit
- ☐ 16 bit
- ☐ 32 bit
- ☒ 64 bit

e) Gegeben folgenden Java Code. Welche konkrete Methodenimplementierung wird beim Aufruf in Zeile 8 verwendet?

```
1 class A {int dyn() {...}}
2 class B extends A {int dyn() {...}}
3 class C {int dyn() {...}}
4 class D extends B {int dyn() {...}}
5 //...
6 public static void main(String... args) {
7     A x = (B) new D();
8     x.dyn()
9 }
```

- ☐ Die der Klasse `A`.
- ☐ Die der Klasse `B`.
- ☐ Die der Klasse `C`.
- ☒ Die der Klasse `D`.
- ☐ Keine. Der Aufruf führt zu einem `RuntimeError`.

f) Gegeben folgenden Java Code. Welche Kombination an Aufrufen und Zugriffen ausgeführt zwischen Zeilen 10-12 ist korrekt?

```
1  public class E {
2      private int x;
3      public String doThings() {...}
4      public double y;
5  }
6  //...
7  public class F extends E {
8      public static int doStuff() {...}
9      private boolean z;
10     public char exercise(F f) {
11         // 1. Befehl
12         // 2. Befehl
13     }
14 }
```

- ☒ Zeile 11: `var u = f.z;`
  Zeile 12: `doThings();`
- ☐ Zeile 11: `var u = f.y;`
  Zeile 12: `E.doStuff();`
- ☐ Zeile 11: `var u = f.y;`
  Zeile 12: `E.doThings();`
- ☐ Zeile 11: `var u = f.x;`
  Zeile 12: `doStuff();`
- ☐ Zeile 11: `var u = f.x;`
  Zeile 12: `var w = f.y;`

Aufgabe 2 - Imperative & Objektorientierte Programmierung $5 + 3 + 3 + 3 = 14$ Punkte

a) Betrachten Sie folgende Klassen welche einen Binärbaum mit rekursiv implementierter binärer Suche umsetzen. Implementieren Sie die Methode `findL(...)` der Klasse `Tree`, welche ebenfalls binäre Suche im Baum umsetzt. Jedoch **ohne** Rekursion, nur mit Hilfe von Schleifen.

```
1  public class Tree {
2      TreeElement root;
3      public TreeElement find(int value) {
4          if (root == null) {return null;}
5          if(root.value == value) {return root;}
6          else {return root.find(value);}
7      }
8  }
```

```
1  public class TreeElement {
2      int value;
3      TreeElement left,right;
4      public TreeElement find(int value) {
5          if(this.value == value) {
6              return this;
7          } else if (value <= this.value) {
8              return left==null?null:left.find(value);
9          } else {
10             return right==null?null:right.find(value);
11         }
12     }
13 }
```

Lösung

```
1  public TreeElement findL(int value) {
2      if(root == null) return null;
3      TreeElement current = root;
4      while(current != null) {
5          if(current.value == value) {
6              return current;
7          } else if(value <= current.value) {
8              current = current.left;
9          } else {
10             current = current.right;
11         }
12     }
13     return current;
14 }
```

b) Betrachten Sie den folgende Quellcode und beantworten Sie die Fragen.

```
1  public class Ex2b{
2      public static void main(String[] args){
3          int[][] matrix = {{1,2},{3,4,5}};
4          bar(matrix)
5          System.out.println(matrix[1][0]);
6          foo(matrix);
7      }
8      public static void bar(int[][] matrix) {
9          matrix[1][0] = 10;
10         matrix[1] = matrix[0];
11     }
12     public static void foo(int[][] matrix) {
13         System.out.println(matrix[0][3]);
14     }
15 }
```

i. Was was wird in Zeile 5 ausgegeben?

- ☐ Die Zahl 10 wird ausgegeben.
- ☒ Die Zahl 1 wird ausgegeben
- ☐ Die Zahl 3 wird ausgegeben
- ☐ `null` wird ausgegeben

ii. Was geschieht beim Aufruf der Methode `foo`?

- ☒ Ein `RuntimeError`
- ☐ Die Zahl 2 wird ausgegeben
- ☐ Die Zahl 3 wird ausgegeben
- ☐ `null` wird ausgegeben

c) Beschreiben Sie, in ganzen Sätzen, 2 Unterschiede und eine Gemeinsamkeit zwischen einer `abstrakten Klasse` und einem `Interface` in Java.

Lösung

z.B.:

Gemeinsamkeiten: können nicht Instanziiert werden, können Methoden deklarieren
Unterschiede: Mehrfachvererbung nur bei Interfaces möglich, abstrakte Klassen können Objektvariablen deklarieren, Interfaces nicht
.

d) Beschreiben Sie, in ganzen Sätzen, was man unter dem Konzept *information hiding* bzw. *Datenkapselung*

versteht, warum es angewendet wird und mit welchen Syntaxkonstrukten dies in Java umsetzt wird.

Lösung Information hiding bezeichnet das absichtliche verbergen bestimmter Daten vor dem Zugriff von außen. Durch die Kapselung werden nur Angaben über die Funktionsweise einer Klasse nach außen sichtbar, nicht aber die genaue interne Darstellung und Realisierung. In Java wird information hiding durch Sichtbarkeitsmodifikatoren und getter- bzw. setter-Methoden realisiert.

.

Aufgabe 3 - OOP

4 + 8 + 2 = 14 Punkte

Betrachten Sie folgende Klassen:

```java
record Tuple(String BookTitle, Page Page) {};

public class Library {
    //Maps Genretype to Collection of books of that genre
    private Map<String, Collection<Book>> books;
}
```

```java
public class Book {
    public String title;
    protected List<Page> pages;
}
```

```java
public class Page {
    public String content;
}
```

a) Implementieren Sie die Methode `insertBook(...)` der Klasse `Library`, welche ein Buch mit übergebenem Genre in die Menge der Bücher zu diesem Genre hinzufügt. Achten Sie dabei darauf, dass Buchtitel pro Genre eindeutig sein müssen. Sollte ein Buch mit selbem Namen bereits existieren soll dieses mit dem übergebenen Buch ersetzt werden.

Lösung

```
1 public void insertBook(Book book, String type) {
2     String bookTitle = book.title;
3     Collection<Book> filteredBooks = books.get(type)
4       .stream()
5       .filter(aBook -> !aBook.title.equals(bookTitle))
6       .collect(Collectors.toList());
7     filteredBooks.add(book);
8     books.put(type, filteredBooks);
9 }
```

alternativ:

Lösung

```
1  public void insertBook(Book book, String genre){
2          Collection<Book> c = books.get(genre);
3
3          // Entfernen falls vorhanden
4      Book replaceThis = null;
5      for(Book b : c){
6          if(b.title.equals(book.title)){
7              replaceThis = b;
8          }
9      }
10     if(replaceThis != null){
11         c.remove(replaceThis);
12     }
13     // Hinzufuegen
14     c.add(book);
15 }
```

b) Implementieren Sie die Methode `findPagesContent(...)` welche alle Bücher herausfiltert die eine Page enthalten deren Inhalt den übergebenen String beinhaltet. Das Rückgabeformat soll dabei eine `Collection` von `Tuple` sein, so dass für jedes gefundene Buch ein Tuple mit dem Titel des Buchs und der ersten Seite die den übergebenen String beinhaltet, in der Collection enthalten ist. Verwenden Sie zur Lösung dieser Aufgabe keine Schleifen sondern ausschließlich Methoden aus der Java Collection Streams-API.

Lösung

```
1  public Collection<Tuple> findPagesContent(String needle) {
2      return books.values()
3          .stream()
4          .flatMap($ -> $.stream())
5          .map($ -> new Tuple(
6              $.title,
7              $.pages.stream()
8                  .filter(p -> p.content.contains(needle))
9                  .findFirst().orElse(null)
10             ))
11         .filter($ -> $.Page() != null)
12         .toList();
13 }
```

c) Beschreiben Sie wie in Java Lambda Ausdrücke mit Hilfe von OOP Konzepten technisch umgesetzt sind.

Lösung

Labnda Ausdrücke werden mittels Functional Interfaces umgesetzt. FIFs sind interfaces die nur eine abstrakte Methode enthalten und somit einen einzelnen Funktionsvertrag abbilden. Mit etwas Syntactic shugar wird das implementieren der Methode in eine Form gebracht die sehr der Definition von Lambdaausdrücken ähnelt.

Aufgabe 4 - Decorator-Pattern und JavaIO

7 + 2 = 9 Punkte

Betrachten Sie das Interface `Collection<E>` und die implementierende Klasse `LinkedList<E>`:

```
1  public interface Collection<E> {
2      boolean add(E e);
3      boolean remove(Object o);
4      // weitere Methoden ...
5  }
6  public class LinkedList<E> implements Collection<E> { //... }
```

a) Implementieren Sie eine Klasse `FilteredCollectionDecorator<E>` auf Basis des Interfaces `Collection<E>`. Verwenden Sie hierfür das **Decorator-Pattern**. Ein `FilteredCollectionDecorator<E>` soll beim Hinzufügen eines Elements durch Aufruf der Methode `boolean add(E e)` mittels eines Filters prüfen, ob das Element hinzugefügt werden soll (dann Rückgabewert true) oder nicht (dann Rückgabewert false). Der Filter soll durch einen Lambda-Ausdruck vom Typ `Predicate<T>` angegeben werden können. Die übrigen Methoden (spezifisch hier nur die Methode `boolean remove(Object o)`) sollen kein geändertes Verhalten zeigen. Achten Sie darauf, dass die durch das Interface definierten Signaturen der Methoden nicht verändert werden.

Lösung

```
1  public class FilterCollectionDecorator<E> implements Collection<E> {
2      private Collection<E> collectionToBeDecorated;
3      private Predicate<E> p;
4      public FilterCollectionDecorator(Collection<E> collectionToBeDecorated,
5       Predicate<E> p) {
6          this.collectionToBeDecorated = collectionToBeDecorated;
7          this.p = p;
8      }
9      @Override
10     public boolean add(T e) {
11         if (p.test(e))
12             return collectionToBeDecorated.add(e);
13         else
14             return false;
15     }
16     @Override
17     public boolean remove(Object o) {
18         return collectionToBeDecorated.remove(o);
19     }
20 }
```

b) Implementieren Sie die Methode `printIfExist(String[] paths)`, welches für jeden übergebenen Pfad, den Pfad ausgibt so wie ein Hinweis dazu ob das Ziel des Pfades ein Ordner ist.
Hinweis: Beachten Sie hierfür auch den JavaNIO Teil des **CheatSheets** am Ende der Klausur.

Lösung

```java
private record Pair (Path path, boolean exists) {};

public static void printIfExist(String[] paths) throws IOException {
    Arrays.stream(paths)
          .map(path -> Path.of(path))
          .map(path -> new Pair(path, Files.isDirectory(path)))
          .forEach(System.out::println);
}
```

Aufgabe 5 - XML, JSON, Build Tools

8 + 3 + 3 = 14 Punkte

Betrachten Sie die folgende *music.xml* (auch zu finden am Ende der Klausur).

```
<?xml version="1.0" encoding="UTF-8"?>
<album name="Great Debut">
    <songs>7</songs>
    <band>
        <artist>Mr. X</artist>
        <artist>Ms. X</artist>
    </band>
</album>
```

a) Implementieren Sie die Methode *change(...)* (aus dem gegebenen Code-Ausschnitt), welche (1) den Namen des Albums in *music.xml* in „Good Debut“ ändert, (2) die Anzahl der Lieder von 7 zu 8 ändert und (3) ein neues Bandmitglied (*artist*) mit dem Namen „The X" hinzufügt.
Hinweis: Sie können davon ausgehen, dass die notwendigen Klassen importiert sind (*Document* ist *org.w3c.dom.Document*).
Hinweis: Beachten Sie hierfür auch die Teile *Element*, *NodeList*, *Node* und *Document* des **CheatSheets** am Ende der Klausur.
Hinweis: *Element* und *Document* sind Subinterfaces von *Node*.

```
DocumentBuilder parser = DocumentBuilderFactory.newInstance()
        .newDocumentBuilder();
Document doc = parser.parse("music.xml");
change(doc);
Transformer trans = TransformerFactory.newInstance().newTransformer();
trans.transform(new DOMSource(doc), new StreamResult(
        "music_changed.xml"));
```

Lösung

```
public static void change(Document doc) {
    Element album = doc.getDocumentElement();
    album.setAttribute("name", "Good Debut");
    Element songs = (Element) album.getElementsByTagName("songs").item(0);
    songs.setTextContent("8");
    Node theX = doc.createElement("artist");
    theX.setTextContent("The X");
    Node band = album.getElementsByTagName("band").item(0);
    band.appendChild(theX);
}
```

b) Definieren Sie ein JSON Dokument, welches äquivalent zum gegebenen XML Dokument ist (die Manipulation in Teilaufgabe a) wird dabei nicht beachtet).

Lösung

```
{
    "album": {
        "name": "Great Debut",
        "songs": "7",
        "band": [
            "Mr. X",
            "Ms. X"
        ]
    }
}
```

c) Nennen und beschreiben Sie **3** Vorteile, die sich durch die Verwendung eines Build Automation Tools wie z.B Gradle ergeben.

Lösung

Zeiteffizienter als manuelle Builds
Notwendige Voraussetzung für Continuous integration / Continuous Deployment
Übernimmt das Dependency-Management
Beschleunigte Builds bei geringen Änderungen
Bauen und Ausführen von Tests
...

Aufgabe 6 - GUI

2 + 4 = 6 Punkte

Betrachten Sie folgenden Scene Graph:

```
<BorderPane prefHeight="500.0" prefWidth="700.0"
  xmlns="http://javafx.com/javafx/8.0.111"
  xmlns:fx="http://javafx.com/fxml/1">
    <center>
        <BorderPane>
            <top>
                <ToolBar>
                    <items>
                        <Button background-color="blue">
                            <graphic>
                                <ImageView pickOnBounds="true"
                                  preserveRatio="true">
                                    <image>
                                        <Image url="@/open.png" />
                                    </image>
                                </ImageView>
                            </graphic>
                        </Button>
                    </items>
                </ToolBar>
            </top>
            <center>
                <SplitPane orientation="VERTICAL">
                    <items>
                        <AnchorPane minHeight="0.0" minWidth="0.0">
                            <children>
                                <Label text="no file loaded..."/>
                                <ScrollPane fx:id="imageScrollPane">
                                    <content>
                                        <ImageView fx:id="imageView"/>
                                    </content>
                                </ScrollPane>
                            </children>
                        </AnchorPane>
                    </items>
                </SplitPane>
            </center>
        </BorderPane>
    </center>
</BorderPane>
```

a) Nennen Sie **2** Stellen im SceneGraph, die eine Anwendung des Composite-Patterns zeigen, und erklären Sie, in ganzen Sätzen, auf Basis dieser Stellen, was dieses Pattern ist. Nennen und erläutern Sie 1 Vorteil des Composite-Patterns.

Lösung

z.B Z1 und Z5, wir haben eine BorderPane in einer BorderPane.
Dies Funktioniert da BorderPane ein Composite des Patterns ist uns somit andere Components/Nodes beinhalten kann (somit auch sich selbst). Das Composite Pattern beschreibt eine Gruppe von Objekten, die wie eine einzelne Instanz desselben Objekttyps behandelt werden. Das Pattern ermöglicht die Gleichbehandlung von primitiven Objekten und Behältern und erlaubt somit eine einfache Repräsentation von verschachtelten Strukturen. Das Pattern sorgt für hohe Flexibilität und Erweiterbarkeit erschwert allerdings die Unterscheidung/ unterschiedliche Behandlung einzelner Komponenten.

b) Ergänzen Sie folgenden Code so, dass ein Fenster mit Button angezeigt wird und beim Mausklick des Buttons der Text *Ouch!* auf der Console ausgegeben wird.

Lösung

```
1  int counter = 0;
2  public void start(Stage primaryStage) {
3      var sp = new ScrollPane();
4      var button = new Button();
5      sp.getChildren().add(button);
6      button.setOnMouseClicked(new EventHandler<MouseEvent>() {
7          @Override
8          public void handle(MouseEvent event) {
9              System.out.println("Ouch!");
10         }
11     });
12     primaryStage.setScene(new Scene(sp));
13     primaryStage.show();
14 }
```

Aufgabe 7 - Threads **7 Punkte**

a) In dieser Aufgabe sollen Sie eine Näherung von Pi mittels der Bailey-Borwein-Plouffe Formel berechnen. Diese ist als Blackbox in der Funktion *BBPFormula(int n)* gegeben. Pi ist gleich der unendlichen Summe (von n = 0 bis unendlich) über diese Formel. Ergänzen Sie den folgenden Code um 50 (gespeichert in *amount*) Summanden der Summe parallel zu berechnen. Dabei soll das Ergebnis parallel in der *DoubleAdder* Variable *acc* gespeichert werden. Der *AtomicInteger amount* gibt an wieviele Summanden noch berechnet werden müssen, dabei soll jeder Thread iterativ den aktuell höchsten noch nicht berechneten Summanden berechnen (*amount* dekrementieren) und so lange laufen bis insgesamt *amount* viele Summanden berechnet wurden. Verwenden Sie zur Synchronisation der Threads nicht das *synchronized* Keyword, sondern die Methoden der Klassen *DoubleAdder* und *AtomicInteger*.
Hinweis: Beachten Sie hierfür auch die Teile *Executor*, *DoubleAdder* und *AtomicInteger* des **CheatSheets** am Ende der Klausur.
Hinweis: *ExecutorService* ist ein Subinterface von *Executor*.

```
1  private static DoubleAdder acc = new DoubleAdder();
2  private static AtomicInteger amount = new AtomicInteger(50);

3  public static double BBPFormula(int n) {
4      // Blackbox
5  }

6  public static double calculatePi() {

7      int threads = Runtime.getRuntime().availableProcessors();
8      ExecutorService executorService = Executors
9                  .newFixedThreadPool(threads);

10     for (int i = 0; i < threads; i++)
11     {
12         executorService.execute(() -> {
13             // Zu ergänzen auf der nächsten Seite
14         });
15     }
16     try {
17         executorService.shutdown();
18         executorService.awaitTermination(Integer.MAX_VALUE,
19                     TimeUnit.SECONDS);
20     }
21     catch (InterruptedException ex) {
22         ex.printStackTrace();
23     }
24     return acc.sum();
25 }
```

Lösung

```
executorService.execute(() -> {
    int n = amount.decrementAndGet() + 1;
    double res;
    while (n > 0) {
        res = BBPFormula(n);
        acc.add(res);
        n = amount.decrementAndGet() + 1;
    }
});
```

music.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<album name="Great Debut">
    <songs>7</songs>
    <band>
        <artist>Mr. X</artist>
        <artist>Ms. X</artist>
    </band>
</album>
```