Studiengang Bachelor Informatik
Studiengang Bachelor Künstliche Intelligenz
Studiengang Bachelor Wirtschaftsinformatik

Technische Hochschule Ingolstadt

Prüfung Grundlagen der Programmierung 2 / Programmierung 2 / Programmieren in Java 2

Aufgabe	1	2	3	4	5	6	7	Summe
Punkte (max.)	(25)	(25)	(25)	(25)				(100)

Note:

Prüfer: S. Apel, H.-M. Windisch
Prüfungsdauer: 90 Min
Hilfsmittel: keine

Studiengang	Dozent	Matrikelnummer	Fachsem.	Raum	Platz

Dieses Geheft enthält sowohl die Aufgabenstellung als auch den Platz für Ihre Antworten. Schreiben Sie möglichst nur in die vorgegebenen Antwortrahmen. Geben Sie am Ende der Prüfung das vollständige Geheft wieder ab.

Füllen Sie die erste Seite noch vor der Prüfung mit Ihren persönlichen Angaben aus.

Die Darbietung Ihrer Aufgabenbearbeitung muss gut lesbar, folgerichtig und verständlich sein.

Viel Erfolg!

Task 1 (Various questions: approx. 25%)

a) Given the source code "Hello.java" of the class Hello, which outputs "Hello World!" to the console. Describe which programmes are used and which files must be used or created in advance in order to run the application. What can be used for large programmes with a large number of classes to quickly install the programme on other computers? (6 Pkt.)

- Javac (Java-Compiler) [1P] translates Hello.java and creates Bytecode [1P] Hello.class [1P]
- Java (JVM) [1P] is called with the Hello parameter to start the programme [1P]
- Bundling the class files of the programme in a JAR (1 P.)

Sum: 6 P.

b) Given the following main method:

```
01  public static void main(String[] args) {
02      Predicate<String> validFilename = s -> {
03          int point = s.lastIndexOf(".");
04          if (point > 0) {
05              String name = s.substring(0, point);
06              String ext = s.substring(point+1);
07              if (name.length() > 0 && ext.length() > 0) {
08                  return true;
09              }
10          }
11          return false;
12      };
13      Consumer<String> output = s -> {
14          String f = s.substring(0,s.lastIndexOf("."));
15          String e = s.substring(s.lastIndexOf(".")+1);
16          System.out.println("file=" + f + ", ext=" + e);
17      };
18      Stream.of("file.mp3", ".mp3", "b.c", "a.", "abc", "42.txt")
19          .filter(validFilename)
20          .forEach(output);
21  }
```

b.1) What output does main() generate when executed? (6 P.)

```
file=file, ext=mp3
file=b, ext=c
file=42, ext=txt
```

Points: 6 P. (2 for every correct line)

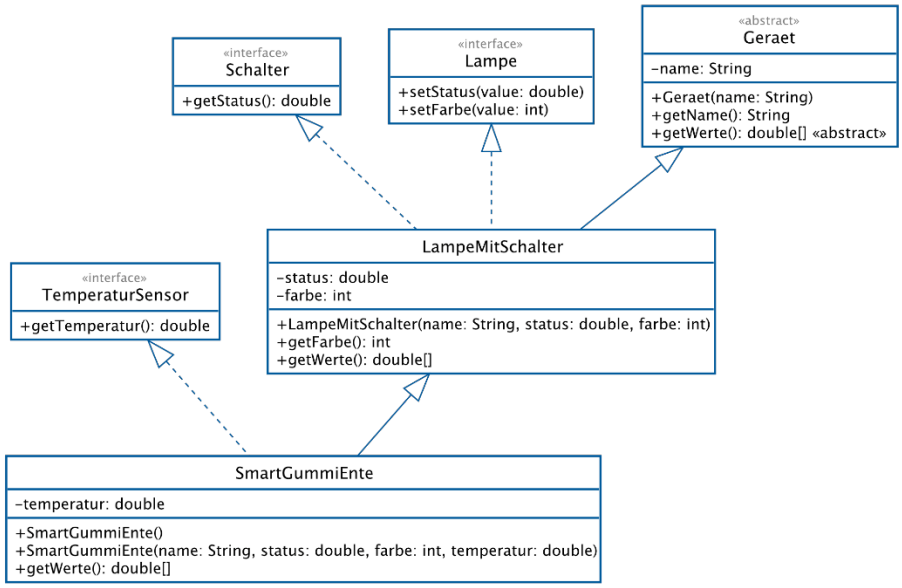
b.2) Complete the following code fragment so that it maps the functionality given above without lambdas and streams and generates the same output. (13 pts.)

Note: Identical lines from the code above may simply be inserted by specifying the line number or a line number range.

public static void main(String[] args) {	
List<String> strings = Arrays.*asList*("file.mp3", ".mp3", "b.c", "a.", "abc", "42.txt");	
for(String file : strings) {	2
int point = file.lastIndexOf(".");	1
if (point > 0) {	1
String name = file.substring(0, point);	1
String ext = file.substring(point+1);	1
if (name.length() > 0 && ext.length() > 0) {	2
String f = file.substring(0, file.lastIndexOf("."));	2
String e = file.substring(file.lastIndexOf(".") + 1);	1
System.***out***.println("file=" + f + ", ext=" + e);	2
}	
}	
}	
}	
	Summe 13P.

Task 2 (Inheritance and Interfaces: approx. 25%)

Given a SmartHome scenario in UML. Devices in this SmartHome can be various actuators and sensors. Actuators can be controlled, sensors provide information. There are devices that both provide information and can be controlled, e.g. lamps with motion sensors. The following diagram shows the interfaces and classes for this scenario.



a.) Given the implementation for the interfaces Schalter, Lampe and TemperaturSensor as well as class Geraet. Implement the classes **LampeMitSchalter** and **SmartGummiEnte.** The result of the method **getWerte** should reflect the state of the called object. (15 Pkt.)

```
01  public interface Schalter {
02    double getStatus();
03  }
04  public interface Lampe {
05    void setStatus(double value);
06    void setFarbe(int value);
07  }
08  public interface TemperaturSensor {
09    double getTemperatur();
10  }
11  public abstract class Geraet {
12      private String name;
13      public Geraet(String name) {
14          this.name = name;
15      }
16      public String getName() {
17          return name;
18      }
19      public abstract double[] getWerte();
20  }
```

Implement class LampeMitSchalter.

public class LampeMitSchalter extends Geraet implements Schalter, Lampe {	2P Impl./Ext.
private double status;	1P Attr.
private int farbe;	
public LampeMitSchalter(String name, double status, int farbe) {	2P Konstr.
super(name);	
this.status = status;	
this.farbe = farbe;	
}	
public double getStatus() {	2P Set/Get
return status;	
}	
public void setStatus(double value) {	
this.status = value;	
}	
public void setFarbe(int value) {	
this.farbe = value;	
}	
public int getFarbe() {	
return farbe;	
}	
public double[] getWerte() {	1P Über.
return new double[] {this.status, this.farbe};	
}	
}	

Implement class SmartGummiEnte.

public class SmartGummiEnte extends LampeMitSchalter	2P Impl./Ext.
implements TemperaturSensor {	
private double temperatur;	0,5P Attr.
public SmartGummiEnte () {	1P 1. Konst.
this("default", 0, 0, 0);	
}	
public SmartGummiEnte(String name, double status, int farbe, double temperatur) {	2P 2. Konst.
super(name, status, farbe);	
this.temperatur = temperatur;	
}	
public double getTemperatur() {	0,5P Getter
return temperatur;	
}	
public double[] getWerte() {	1P Über.
return new double[] {this.getStatus(), this.getFarbe(), this.temperatur};	
}	
}	

b.) The following controller with three devices is given. Depending on the temperature and switch value, other devices are to be adapted. Add the missing data types. Select a suitable data type for the variables (device1, device2, device3, sensor, deviceList1, deviceList2, g1, g2 and g3). Use a data type that provides the minimum scope of functionality, but in such a way that the example can be executed. (10 Pt., 1 P. per type)

```
01  public class Controller {
02    public static void main(String[] args) throws Exception {
03      SmartGummiEnte geraet1 = new SmartGummiEnte("Ente", 0.75, 0xFFFFFF, 28.0);
04    LampeMitSchalter geraet2 = new LampeMitSchalter("Tisch", 1, 0xFFFFFF);
05    LampeMitSchalter geraet3 = new LampeMitSchalter("Steh", 0.5, 0xFFFFFF);
06    TemperaturSensor sensor = geraet1;
07       Lampe[] geraeteListe1 = new    Lampe [] {geraet1, geraet2, geraet3};
08  LampeMitSchalter[] geraeteListe2 = new LampeMitSchalter[] {geraet1, geraet2, geraet3};
09
10      // Ändere Farbe passend zur Temperatur
11      for(   Lampe g1: geraeteListe1) {
12        if(sensor.getTemperatur() > 20.0) {
13          g1.setFarbe(0xFF0000); // rot
14        } else {
15          g1.setFarbe(0x0000FF); // blau
16        }
17      }
18
19      // Synchronisiere Lampenschaltung
20      double value = 0;
21      for(   Schalter g2: geraeteListe2) {
22        value = Math.max(value, g2.getStatus());
23      }
24      for(   Lampe g3: geraeteListe2) {
25        g3.setStatus(value);
26      }
27    }
28  }
```

Task 3 (Collections: approx. 25%)

Coulours may be described using three colour channels. One channel for red, one for green and one for blue. Each channel has a value range from 0 to 255, e.g. orange would be 255 for red, 165 for green and 0 for blue.

The class Colour with the three attributes red, green and blue as well as getters, a constructor, the methods hashCode and equals are provided below. Each colour channel is represented by an attribute of type “int”.

In the following subtasks, add the implementations for the methods “verarbeiten” and “ausgeben” as well as a comparator for sorting colours.

```
01  public class Farbe {
02      private int rot;
03      private int gruen;
04      private int blau;
05      public Farbe(int rot, int gruen, int blau) {
06          this.rot = rot;
07          this.gruen = gruen;
08          this.blau = blau;
09      }
10      public int getRot() { return rot; }
11      public int getBlau() { return blau; }
12      public int getGruen() { return gruen; }
13      public int hashCode() { /*…*/ }
14      public boolean equals(Object o) { /*…*/ }
15      public static void main(String[] args) {
16          String[] tokens = new String[] {
17              "rot", "255", "0", "0",
18              "weiss", "255", "255", "255",
19              "türkis", "0", "255", "255",
20              "orange", "255", "165", "0",
21              "grau", "128", "128", "128" };
22          Map<Farbe, String> map = verarbeiten(tokens);
23          ausgeben(map);
24      }
25  }
```

a.) Realise the static method “verarbeiten”. Process the specified string array so that corresponding instances of the Colour class are created. Insert these instances as a key and the associated name as a value in a new HashMap to be created. Note: You can use Integer.parseInt(String s) to convert a character string into an integer. (9 Pkt.)

```
private static Map<Farbe, String> verarbeiten(String[] tokens) {
    Farbe farbe;
    Map<Farbe, String> map = new HashMap<>();                    1P Map
    for (int i = 0; i < tokens.length; i += 4) {                 2P Iter. / Ind.
        int r = Integer.parseInt(tokens[i + 1]);
        int g = Integer.parseInt(tokens[i + 2]);                 2P Int-Parse
        int b = Integer.parseInt(tokens[i + 3]);
        farbe = new Farbe(r, g, b);                              1P instance
        map.put(farbe, tokens[i]);                               1P put
    }
    return map;
}
```

b.) Implement a comparator for the concrete data type Colour, which sorts instances of class Colour in ascending order first by the red attribute, then by the green attribute and finally by the blue attribute. Pay attention to parameters that do not reference an instance, i.e. are null. (8 Pkt.)

```
public class FarbComparator implements Comparator<Farbe> {      1P class / gen. type
    public int compare(Farbe f1, Farbe f2) {
        if (f1 == f2 || (f1 == null && f2 == null)) {            2P for null: NPE oder -1, 0 1
            return 0;
        }
        if (f1 != null && f2 == null) {
            return -1;
        }
        if (f1 == null && f2 != null) {
            return 1;
        }
        if (f1.getRot() != f2.getRot()) {                        1P red
            return f1.getRot() - f2.getRot();
        }
        if (f1.getGruen() != f2.getGruen()) {                    1P green
            return f1.getGruen() - f2.getGruen();
```

```
        }
        return f1.getBlau() - f2.getBlau();
    }
}
```

1P blue
2P corr. Sort.

c.) Finally, realise the static method “ausgeben”. The output should be sorted according to the key, i.e. the colour! Output each entry in "map" as "<colour> = <name>". The first entry in "tokens" would be output as follows: “255,0,0 = red”. (8 Pt.)

```
private static void ausgeben(Map<Farbe, String> map) {
        Set<Farbe> keys = map.keySet(); // not sortable
        // Transfer to list, e.g. ArrayList, iteration also conceivable
        List<Farbe> sorted = new ArrayList<>(map.keySet());
        Collections.sort(sorted, new FarbComparator());

        for (Farbe f : keys) {
            System.out.println(
                    f.rot + "," +
                    f.gruen + "," +
                    f.blau + " = " + map.get(f));
        }
}
```

2P keySet
2P to List
2P Sort
1P Iter.
1P output

Task 4 (GUI: approx. 25 %)

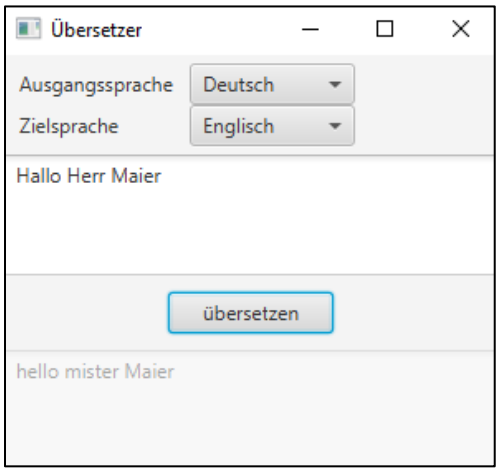
In this task, a user interface for translating text from one language into another is to be developed (see illustration below). The application works as follows:

A1. The selection field for the source language is preset with a list of languages (see source text below). The first language should be preselected (here: “Deutsch”).

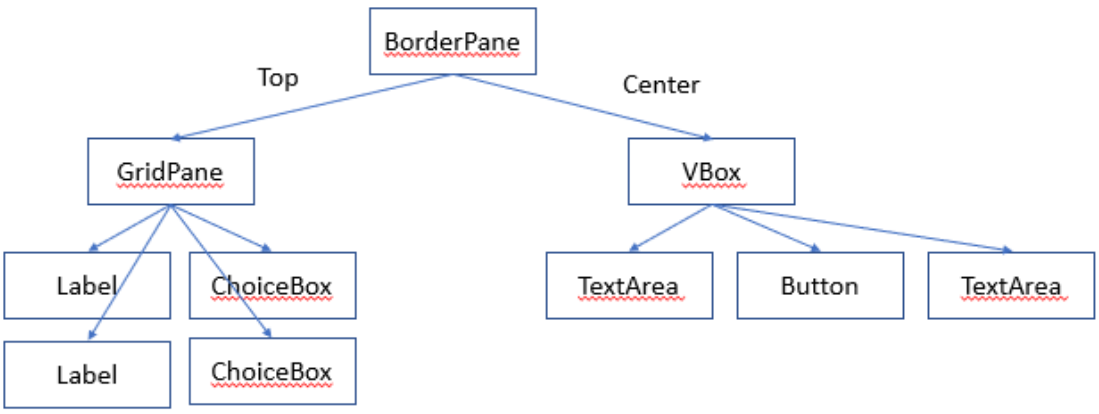
A2. The selection field for the target language contains all languages except for the source language. The data required for initialisation is obtained by calling the predefined method **getTargetLanguages** (see source code below).

A3. Each time the source language is changed, the selection field for the target language must be reassigned.

A4. After entering a text and clicking on the "übersetzen" button, the translated text is determined and displayed using the **translateSentence** method of the **Translator** class (specified below).



a) Sketch the dialog layout using a layout hierarchy (6 Pt.).



Points: 10 Elements + Top/Center = 12 / 2 = 6 P.

Now complete the following fragment for the application.
b) In the **start** method, add the code for the structure of the dialog. Pay special attention to requirements A1 and A2. (6 points)
c) Also add the code for event handling for requirements A3 and A4 in the start method. (13 pts.)

```
class Translator {
   public Translator() { … }
   /**
    * Translates a sentence from a source language into a target language.
    * @param sourceLanguage Source language
    * @param targetLanguage Target language
    * @param text Text to be translated
    * @return text result in target language
    */
   public String translateSentence(String sourceLanguage, String targetLanguage, String text) { … }
}

public class UebersetzerApp extends Application {
   static public void main(String[] args) {
      launch(args);
   }
   /**
    * Returns the language list without the source language as ObservableList
    * @param allLanguages List with all languages
    * @param sourceLanguage selected source language
    * @return language list
    */
   private ObservableList<String> getTargetLanguages(List<String> allLanguages,
            String sourceLanguage) { … }

   @Override
   public void start(Stage stage) {
      List<String> allLanguages = Arrays.asList(
            "Deutsch", "Englisch", "Französisch", "Russisch");
      ObservableList<String> sprachen = FXCollections.observableList(allLanguages);

/* from here: subtask b) Dialog setup */
      stage.setTitle("Übersetzer");                                          12 / 2
      BorderPane mainPane = new BorderPane();                                = 6 P.
      GridPane grid = new GridPane();
      VBox vbox = new VBox();
      ChoiceBox<String> languageCombo1 = new ChoiceBox<>(sprachen);
      languageCombo1.getSelectionModel().select(0);
      ChoiceBox<String> languageCombo2 = new ChoiceBox<>();
      languageCombo2.setItems(getTargetLanguages(allLanguages, "Deutsch"));
      languageCombo2.getSelectionModel().select(0);
      grid.add(new Label("Ausgangssprache"), 0, 0);
      grid.add(languageCombo1, 1, 0);
      grid.add(new Label("Zielsprache"), 0, 1);
      grid.add(languageCombo2, 1, 1);

      TextArea orgText = new TextArea();
      Button translateButton = new Button("übersetzen");
      TextArea translatedText = new TextArea();
      translatedText.setDisable(true);
      vbox.getChildren().addAll(orgText, translateButton, translatedText);
```

```
      mainPane.setTop(grid);
      mainPane.setCenter(vbox);


/* from here: subtask c) event handling */

translateButton.setOnAction(e -> {
   Translator translator = new Translator();
   String sourceLanguage = languageCombo1.getSelectionModel().getSelectedItem();
   String targetLanguage = languageCombo2.getSelectionModel().getSelectedItem();
   String text = orgText.getText();
   String textOut = translator.translateSentence(sourceLanguage, targetLanguage, text);
   translatedText.setText(textOut);
});

languageCombo1.setOnAction(e -> {
   String sourceLanguage = languageCombo1.getSelectionModel().getSelectedItem();
   languageCombo2.setItems(getTargetLanguages(allLanguages, sourceLanguage));
   languageCombo2.getSelectionModel().select(0);
});
```

```
1
2
1
1
3
1

1
2
1
---
13 P.
```

```
      Scene scene = new Scene(mainPane, 300, 250);
      stage.setScene(scene);
      stage.show();
   }
}
```