Technische Hochschule Ingolstadt
Studiengang Wirtschaftsinformatik

Prüfung
Programmierung in Java II

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

Note:

Prüfer: 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 (Miscellaneous questions: approx. 18%)

Subtask a)

Illustrate the substitution principle using a small example (Java code!) and explain the principle (in words).

Principle:

(classes) a subclass reference can be used instead of a superclass reference

(interfaces) if an interface type is required instead of a class, any reference to an obj can be used if the object implements the interface

Examples:

- List<Student> liste = new ArrayList<>();

- void method(Person p) { ... } // Student inherits from Person

Call: method(new Student());

- ...

Points: Principle (2P), Example (3P) => 5P.

Subtask b)

Given are the following statements:

```
Student[] studenten = { new Student(3, "Hugo"), new Student(4711,
"Anna"), new Student(29, "Evi") };

/* (*) */
Arrays.stream(studenten)
.mapToInt(s -> s.getMatrikelnummer())
.sorted()
.forEach(nr -> {
    System.out.println(nr);
});
```

Formulate the above statement (*) without Lamba expressions or streams:

```
List<Integer> liste = new ArrayList<>();    2
for (Student s : studenten) {               1
   liste.add(s.getMatrikelnummer());        2
}
Collections.sort(liste);                    2
for (Integer nr : liste)                    1
   System.out.println(nr);                  1
                                            Sum: 9P.
```

Subtask c)

Given the following class for managing a list of student objects:

```
public class StudentDB {
    private List<Student> studentList = new ArrayList<>();

    public StudentDB() {
        /* fill studentList, omitted here ... */
    }
    public Student getStudentForIndex(int index) {

                if (index < 0) {                                                  1
                    throw new IndexOutOfBoundsException("Index negative");        2
                }

                try {                                                             1
                    return studentenListe.get(index);
                } catch (IndexOutOfBoundsException e) {                           2
                    return null;                                                  1
                }
                                                                                  Sum: 6P.
    }
    public static void main(String[] args) {
        StudentenDB studentenDB = new StudentenDB();
        System.out.println(studentenDB.gibStudentZuIndex(2));
        System.out.println(studentenDB.gibStudentZuIndex(12));
        System.out.println(studentenDB.gibStudentZuIndex(-1));
    }
}

/* --------- Konsolen-Ausgabe ------------ */
(3, Peter)
null
IndexOutOfBoundsException: Index negative
```

The student list is filled with 3 student objects.
Supplement the method getStudentForIndex, so that main generates the output shown, i.e. returns "null" if the index is too large and an IndexOutOfBoundsException is thrown if the index is negative.

Task 2 (Collections: ca. 35%)

The following classes are given to implement a chat programme called "WasGehtAb". Similar to WhatsApp, it manages a number of contacts and the messages between them.

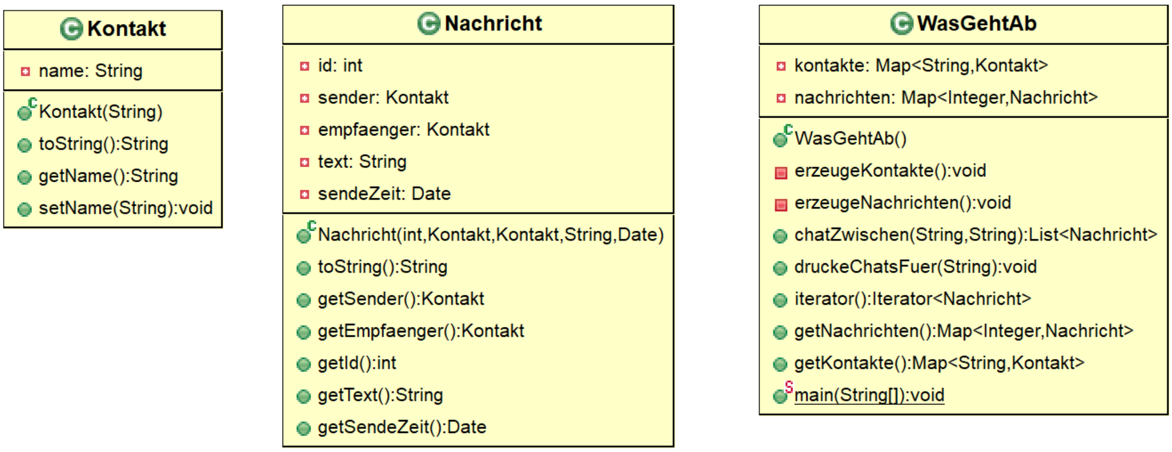


Abbildung 1: Class of the chat app

The WasGehtAb class holds the available contacts in a map data structure; the key here is the unique contact name. All exchanged messages are also managed in a map, whereby the key here is the unique message ID.

Subtask a)

The chatZwischen method of the WasGehtAb class returns a list of the messages that have been exchanged between two contacts. The contacts are given by their names.
Please note: it does not matter whether a message was sent or received by "name1", the same applies to "name2"!
In the example: name1 = Paul, name2 = Anna.

Example (from the main method): The following code

```
WasGehtAb wasGehtAb = new WasGehtAb();
System.out.println("a) Ausgabe von chatZwischen(Paul, Anna): ");
List<Nachricht> nachrichtenPaulUndAnna = wasGehtAb.chatZwischen("Paul", "Anna");
for (Nachricht n : nachrichtenPaulUndAnna)
        System.out.println(n);
```

generates this - still chronologically unordered - output:

```
a) Ausgabe von chatZwischen(Paul, Anna):
Paul -> Anna (05.08.2018 10:30:10): Solala, habe morgen einen Zahnarzttermin :-(
Paul -> Anna (01.08.2018 10:21:10): Hi, wie geht's?
Anna -> Paul (01.08.2018 10:23:12): Danke, gut und dir?
```

Give an implementation for chatZwischen():

```
public List<Nachricht> chatZwischen(String name1, String name2) {

    Kontakt kontakt1 = kontakte.get(name1);                                  2
    Kontakt kontakt2 = kontakte.get(name2);                                  2
    List<Nachricht> ergebnis = new ArrayList<>();                            2

    for (Nachricht n : nachrichten.values()) {                               2
        if (n.getSender() == kontakt1 && n.getEmpfaenger() == kontakt2) {    3
            ergebnis.add(n);                                                 1
        } else if (n.getSender() == kontakt2                                 1
                && n.getEmpfaenger() == kontakt1) {                          1
            ergebnis.add(n);                                                 1
        }
    }

    return ergebnis;                                                         1
                                                                             Sum: 16P.
```

Subtask b)

The method printChatsFor of the WasGehtAb class prints the chats between a specific contact and other contacts to the console.

Example:

```
wasGehtAb.druckeChatsFuer("Paul");
```

generates this - still chronologically unordered - output:

```
>Chat zwischen Paul und Anna
Paul: Solala, habe morgen einen Zahnarzttermin :-(
Paul: Hi, wie geht's?
Anna: Danke, gut und dir?
>Chat zwischen Paul und Susi
Susi: Danke, gut! Freu' mich auf die Ferien :-)
Paul: Hi, was macht die Uni?
```

Give an implementation for druckeChatsFuer:

```
public void druckeChatsFuer(String name) {

    Kontakt ich = kontakte.get(name);                                            2

    System.out.println("b) Ausgabe von druckeChatsFuer(" + name + ")");          2
    for (Kontakt k : kontakte.values()) {                                        3
        if (ich == k) {                                                          1
            continue;                                                            1
        }

    List<Nachricht> chat = chatZwischen(name, k.getName());                      3
    System.out.printf(">Chat zwischen %s und %s\n", name, k.getName());          2
    for (Nachricht n : chat) {                                                   2
        System.out.println(n.getSender().getName() + ": " + n.getText());        3
    }
}                                                                                Sum:
                                                                                 19P.







}
```

Task 3 (Interfaces: ca. 19%)

Consider the classes from task 2 as given.

Subtask a)
Define a comparator called "NachrichtenComparator" to organise messages in ascending order by transmission time.
Note: class Date implements the Comparable interface!

Complete the missing Comparator definition!

```
class NachrichtenComparator implements Comparator<Nachricht> {                3
        public int compare(Nachricht n1, Nachricht n2) {                      2
                return n1.getSendeZeit().compareTo(n2.getSendeZeit());        3
        }
}                                                                             Summe: 8P.
```

Subtask b) Iterator for class WasGehtAb

The messages should now be output in the correct chronological order of the messages with the help of an iterator (see for loop).

Main method:

```
Kontakt k1 = null, k2 = null;
for (Nachricht n : wasGehtAb) {
      if ((n.getSender() != k1 || n.getEmpfaenger() != k2) &&
          (n.getSender() != k2 || n.getEmpfaenger() != k1)) {
            k1 = n.getSender();
            k2 = n.getEmpfaenger();
            System.out.printf(">Chat zwischen %s und %s\n",
                k1.getName(), k2.getName());
      }
      System.out.println(n.getSender().getName() + ": " + n.getText());
}
```

Output:

```
>Chat zwischen Paul und Anna
Paul: Hi, wie geht's?
Anna: Danke, gut und dir?
Paul: Solala, habe morgen einen Zahnarzttermin :-(
>Chat zwischen Paul und Susi
Paul: Hi, was macht die Uni?
Susi: Danke, gut! Freu' mich auf die Ferien :-)
```

What needs to be added to the WasGehtAb class to make it iterable?

```
1.  public class WasGehtAb implements Iterable<Nachricht> {        2

2.  Add method:
            public Iterator<Nachricht> iterator() {                2
                    return new WasGehtAbIterator(this);            2
            }
                                                                   6 P.
```

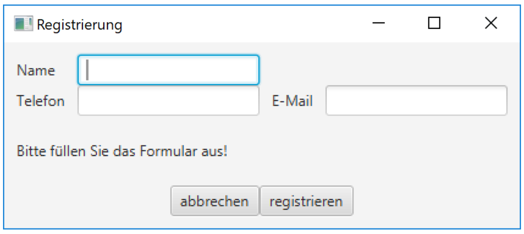
Complete the WasGehtAb iterator below:

Hint: The constructor builds a list of sorted chats to be iterated over.

```
class WasGehtAbIterator implements Iterator<Nachricht> {
    /* your attributes here... */


    public WasGehtAbIterator(WasGehtAb app) {
        Map<String, Kontakt> kontakte = app.getKontakte();
          // Prepare list of messages from all chats for iteration
        int i1 = 0, i2; // Auxiliary indexes to reduce the number of contact pairs
        for (Kontakt von : kontakte.values()) {
              i2 = 0;
                for (Kontakt an : kontakte.values()) {
                    if (i1 <= i2) {
                        // Prevents multiple contact combinations
                        continue;
                        }
                List<Nachricht> chat = app.chatZwischen(
                   von.getName(), an.getName());
                Collections.sort(chat, new NachrichtenComparator());
                chats.addAll(chat);
                i2++;
            }
            i1++;
        }
    }

    // Is there a next element?
    public boolean hasNext() {
        return index < chats.size();
    }                                                                        2

    // deliver next element
    public Nachricht next() {
        Nachricht n = null;                                                  1

        if (hasNext()) {                                                     1
            n = chats.get(index++);                                          2
        }
        return n;                                                            1
    }                                                                        Summe:
}                                                                            7P.
```

Task 4 (GUI: ca. 30 %)

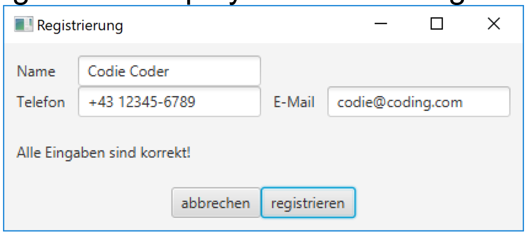
A registration form is to be developed with JavaFX.



Bitte füllen Sie das Formular aus!

Abbildung 2: Empty form

Abbildung 2 shows the form after it has been started. If a name and a correct e-mail address have been entered (the telephone number is optional, so it can be empty), the message shown in Figure 3 is displayed after clicking on the Register button.

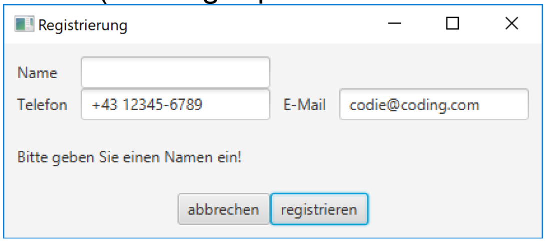


Name: Codie Coder
Telefon: +43 12345-6789
E-Mail: codie@coding.com

Alle Eingaben sind korrekt!

Abbildung 3: Form after successful registration

When you click on the Register button, the mandatory fields Name and E-mail are checked. Errors lead to a corresponding message. Figure 4 shows how the system reacts to an empty name field (message “please enter a name!”).



Telefon: +43 12345-6789
E-Mail: codie@coding.com

Bitte geben Sie einen Namen ein!

Abbildung 4: Form with validation error

Subtask a)

Add a start method to the RegistrationForm class given below, which sets up the form as shown above and displays it on the screen.

Attention:
the assignment of the variable ButtonHandler should be made in subtask b)!
However, it should be used in start() to set up event handling.

```
public class RegistrierungsFormular extends Application {
        private Label nameLabel, telefonLabel, emailLabel, nachrichtLabel;
        private TextField nameTextField, telefonTextField, emailTextField;
        private Button abbrechenButton, registrierenButton;

      /* *** Teilaufgabe b) *** */
        private EventHandler<ActionEvent> abbrechenButtonHandler = ...;
        private EventHandler<ActionEvent> registrienButtonHandler = ...;

        public static void main(String[] args) {
                launch();
        }

        private void initialisiereControls() {
                nameLabel = new Label("Name");
                telefonLabel = new Label("Telefon");
                emailLabel = new Label("E-Mail");
                nameTextField = new TextField();
                telefonTextField = new TextField();
                emailTextField = new TextField();
                abbrechenButton = new Button("abbrechen");
                registrierenButton = new Button("registrieren");
        }

        private boolean korrekteEmail(String email) {
                Pattern pattern = Pattern.compile("^.+@.+\\..+$");
                Matcher matcher = pattern.matcher(email);
                return matcher.matches();
        }

        @Override
        public void start(Stage stage) throws Exception {
                initialisiereControls();

                stage.setTitle("Registrierung");                              1
                BorderPane borderPane = new BorderPane();                     1
                VBox vBox = new VBox();                                       1
                GridPane gridPane = new GridPane();                           1
                FlowPane buttonPane = new FlowPane();                         1

                gridPane.add(nameLabel, 0, 0);                                2
                gridPane.add(nameTextField, 1, 0);                            1
                gridPane.add(telefonLabel, 0, 1);                             1
                gridPane.add(telefonTextField, 1, 1);                         1
                gridPane.add(emailLabel, 2, 1);                               1
                gridPane.add(emailTextField, 3, 1);                           1

                nachrichtLabel = new Label("Bitte füllen Sie das Formular     1
                aus!");
                vBox.getChildren().addAll(gridPane, nachrichtLabel);          2
                borderPane.setCenter(vBox);                                   1

                buttonPane.getChildren().addAll(abbrechenButton,              2
                registrierenButton);
                buttonPane.setAlignment(Pos.CENTER);                          1
                borderPane.setBottom(buttonPane);                             1
                Scene scene = new Scene(borderPane);                          1

                abbrechenButton.setOnAction(abbrechenButtonHandler);          1
                registrierenButton.setOnAction(registrienButtonHandler);      1

                stage.setScene(scene);  // scene bitte vorher erzeugen!       Summe: 23P.
                stage.show();
        }
}
```

Subtask b)

Now add the event handling to implement the functionality of the form described above. To do this, assign a suitable lambda expression to the variables cancelButtonHandler and registerButtonHandler.

Note: use the given method correctEmail!

```
private EventHandler<ActionEvent> abbrechenButtonHandler = event ->           1
    System.exit(0); // oder Dialog schliessen                                 1

private EventHandler<ActionEvent> registrienButtonHandler = event -> {
    if (nameTextField.getText().trim().length() == 0) {
        nachrichtLabel.setText("Bitte geben Sie einen Namen ein!");
    } else if (!korrekteEmail(emailTextField.getText())) {
        nachrichtLabel.setText(
            "Bitte geben Sie eine korrekte E-Mail-Adresse ein!");
    } else {                                                                  1
        nachrichtLabel.setText("Alle Eingaben sind korrekt!");                2
    }                                                                         1
};                                                                            2
                                                                              1


                                                                              1

                                                                              Summe: 10P.
```