Technische Hochschule Rosenheim
Technical University of Applied Sciences

Fakultät für Informatik

Example Exam:	Programming Basics (ProgB),
Examiner:	Prof. Dr. Silke Lechner-Greite
Datum:	
Dauer:	75 Minutes
Material:	A book with ISBN – nr.

Surname: _____

Name: _____

Matriculation number: _____

Achieved Score:

1	2	3	Σ
15	45	15	/ 75

Grade: _____

_____
(First corrector)

_____
(Second corrector)

Hints:

- The staples must not be loosened.
- Please check: The specification includes 11 pages incl. cover sheet. Only the front of one sheet is printed.
- Work on the questions directly in the specification. If necessary, use the reverse side.
- Do not use a pencil or a red or green pen.
- Please write legibly, if possible in block letters.
- If, in your opinion, there are contradictions in the information or information is missing from the assignment, make reasonable assumptions and document them.
- All questions and solutions refer to the programming language Java.
- The number of achievable points is for orientation purposes and may still change.

!!! Good Luck !!!

Name: _____ Points:

A1: Short question - short answer (15 points)

a) How do you define the state and behaviour of objects?

The state is defined through the values of the attributes and the behaviour through the methods.

b) Give a brief example of why it may be important for classes to be immutable.

If an object that is not immutable is passed to a method, one cannot completely rule out the possibility that the object has been modified by the method. If the class is realised as immutable, the state can no longer change.

c) What does the signature of a method look like? Give an example.

Visibility modifier, [static], data type of return value, method name ([0 .. n parameter])
public static void main (String[] args)

d) What does "this()" mean and when is it used?

this is the reference to the own object. Thus, this() calls the parameterless constructor of its own class.

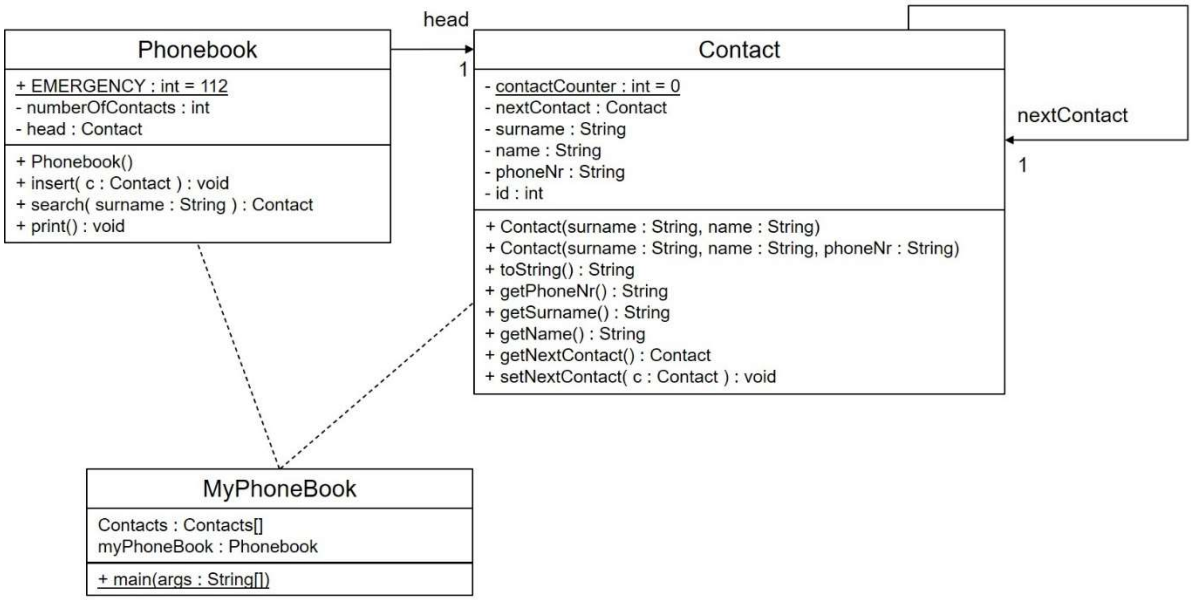
e) What is represented by a class diagram?

A class diagram is used to represent the structure of a system to be developed. Individual classes, their properties and behaviours as well as the connections between classes are depicted/modelled in this diagram.

Name: _____ Points:

A2: Programming classes (45 points)

You want to manage your telephone contacts through a simple telephone book. The following UML class diagram is given to help you:



Note: Static attributes are underlined. Please refer to the class diagram for visibilities and data types.

a) (16 P) Implement the class `Contact`:

- The class Contact has a surname and name, a phone number and an internal ID. The ID represents a counter that increases with newly initiated objects of that class. The ID is also used to set the contactCounter.
- There are two value constructors. The constructor with 2 input parameters sets the phone number to "0" and initialises `nextContact` with zero. The constructor with 3 input parameters checks whether the input phone number is preceded by a "0". If not, an `InvalidTelephoneNumberException` is thrown (assume this one as given.
- The `toString()-` method returns a String that contains the objects surname, name and phone number separated by a comma.
- There should be a getter method for all attributes.
- A setter method is to be defined only for the attribute `nextContact`, whereby the object attribute `nextContact` is reassigned to the passed contact.

Solution on next page.

Name: _____ Points:

```
public class Contact {

    private static int contactCounter = 0;  // Instance counter - 1P
    private Contact nextContact;  // object attributes - in total 2P
    private String surname;
    private String name;
    private String phoneNr;
    private int id;


    // Value constructor 1: Telephone number unspecified. This is set to 0.
2P
    public Contact(String surname, String name) {
        contactCounter++;
        id = contactCounter;
        this.surname = surname;
        this.name = surname;
        this.nextContact = null;
        this.phoneNr = "0";
    }

    // Value constructor 2: important here: Concatenation & Exception. 3P
    public Contact(String surname, String name, String phoneNr) throws
InvalidTelephoneNumberException{
        this(surname, name);
        char t = phoneNr.charAt(0);
        System.out.println(phoneNr.charAt(0));// call other constructor
        if (phoneNr.charAt(0)!='0')
            throw new InvalidTelephoneNumberException("Incorrect phone
number format");
        this.phoneNr = phoneNr;
    }


   // Getter and setter methods together 2P
    public String getPhoneNr() { return phoneNr; }
    public String getSurname() {
        return surname;
    }
    public String getName() {
        return name;
    }
    public Contact getNextContact() {
        return nextContact;
    }
    public void setNextContact(Contact nextContact) {
        this.nextContact = nextContact;
    }

    // String-Representation of contact 2P
    @Override
    public String toString() {
        return getSurname() + "," + getName() + "," + getPhoneNr();
    }

}
```

b) (18P) Implement the class `Phonebook`:

- The class `Phonebook` contains an immutable class attribute to store the emergency number 112.
- The attribute `numberOfContacts` stores how many elements are currently in the phone book.
- A `Phonebook` knows the beginning of the contact list. A reference variable `head` refers to the first contact in the telephone contact list.
- The class `Phonebook` contains a default constructor that initialises the object attributes appropriately: `head` refers to a null object, and `numberOfContacts` is set to 0.
- The `insert`() method inserts a contact `c` at the beginning of the list of the phone book entries. Here, `head` should now refer to the new contact.
- The `search`() method searches for a contact with a specific surname and returns null if there is no such contact in the list of phonebook entries.
  Hints:
  - Verwenden Sie die Methode `equals()` der Klasse String um zwei Strings auf inhaltliche Gleichheit zu überprüfen.
  - Um das Telefonbuch *nach dem Nachnamen* zu durchsuchen, muss man bei `kopf` beginnen und sich durch die Liste der Kontakte „hangeln“ bis man auf einen `null`-Verweis trifft (while()).
  - Use the `equals`() method of the String class to check the equality of two Strings.
  - To search the phone book for the *surname*, you have to start at `head` and go through the list contact for contact until a null reference is hit (while()).
- Die Methode `drucken()` gibt auf der Konsole alle Kontakte des Telefonbuchs aus. Beispielausgabe:
- The `print`() method prints all contacts of the phone book on the console. Example output:

  `2 Contacts [surname, name, phoneNr] [surname, name, phoneNr]`
- Call the `toString`() method of the `Contact` class here.

```
public class Phonebook {

    // Class attribute and object attributes together 2P
    public final static int EMERGENCY = 112;
    private Contact head;        // Start of list
    private int numberOfContacts; // number of contacts in address book

    // Default constructor 1P
    public Phonebook() {
        head = null;
        numberOfContacts = 0;
    }

    // Insert contact at the top of the list - 3P
    public void insert(Contact k) {
        k.setNextContact(head);
        head = k;
        numberOfContacts++;
    }

    // Search for a contact based on "name" (assumption: name is unique) -
4P
```

Name: _____ Points:

```
    public Contact search(String name) {
        Contact actual = head;
        while (actual != null && !actual.getSurname().equals(name)) {
            actual = actual.getNextContact();
        }
        return actual;  // Attention: "actual" is "null" in case the
phonebook is empty
    }

    // Printout of all contacts in the phone book - 4P
    public void print() {
        Contact actual = head;
        System.out.print(numberOfContacts + " Contacts ");
        while (actual != null) {
            System.out.print("[" + actual + "]");
            actual = actual.getNextContact();
        }
        System.out.println();
    }

    // Deleting a contact k from the list
    public void delete (Contact k) {
        Contact actual = head;
        Contact previous = null;        // Saving the previous contact
during the search
        while (actual != null && actual != k) {
            previous = actual;
            actual = actual.getNextContact();
        }
        if (actual != null) {        // Contact to be deleted could not be
found
            if (previous == null) {     // Contact to be deleted is the
first contact in the list
                head = actual.getNextContact();
            }
            else {                      // Contact to be deleted is not the
first contact in the list
                previous.setNextContact(actual.getNextContact());
            }
            numberOfContacts--;
        }
    }

}
```

(If you need more space then please continue writing on the reverse side).

c) (11P) The class `MyPhonebook` simulates your phone book and calls different functions of the classes `Contact` and `Phonebook`. **Complete the programme at the points marked with // ToDo:**

```
public class MyPhonebook{

    // ToDo: Declare
          - a class attribute contacts of type Contact array
          - a class attribute myPhonebook of the type Phonebook
    private static Contact[] contacts;
    private static Phonebook myPhonebook;

    public static void main(String[] args) {
      // ToDo: Fill the contact array with the following three
      contacts:
         // Berger, Frank, 031232
         // Huber, Pamela, 083882
         // Maier, Sonja, 029399
         // Hint: Be sure to catch any exceptions that may be thrown.

    contacts = new Contact[3];
    try {
        contacts[0] = new Contact("Berger", "Frank", "031232");
        contacts[1] = new Contact("Huber", "Pamela", "083882");
        contacts[2] = new Contact("Maier", "Sonja", "029399");
    } catch (InvalidTelephoneNumberException e) {
        System.out.println(e.getMessage());
    }


       // ToDo: Initialise your phone book and add one contact
                after the other to the phone book using insert().

    myPhonebook = new Phonebook();
    if(contacts.length != 0 ) {
        System.out.println(contacts.length);
        for (Contact c : contacts) {
            myPhonebook.insert(c);
        }
    }


       // ToDo: Output the current contents of the phone book to
              the console.
    myPhonebook.print();


       // ToDo: Search for the telephone number of "Maier".
              Issue a message on the console if the contact was
              not found.
    Contact found = myPhonebook.search("Maier");
    if (found != null) {
        System.out.print("Found: " + found.getPhoneNr());
    } else {
        System.out.println("Contact could not be found.");
    }
    }
}
```

Name: _____ Points:

A3: Processing Strings (15 points)

The following String array is given:

```
String[] contacts = {
        "Berger,Frank,031232",
        "Huber,Pamela,083882",
        "Maier,Sonja,029399"
};
```

Each individual String contains surname, name and telephone number separated by a comma.

The aim of this task is to format the individual entries in such a way that all telephone numbers that start with a "0" are given the "+49". The preceding "0" is then to be removed.

Complete the class Util in the places marked with *// ToDo.*

```
public class Util {

    // ToDo: Implement a static method format() that receives a String as
input and also returns a String.
    //- Use .split(",") to separate the input string after a comma and
store it in a string array.
    //- Use .substring(1) to join "+49" with the rest of the string.
    //- Work with a StringBuilder to reconnect the split string parts of
the string array.
    // 10P

    public static String format(String input) {  // 1P

        String[] inSplit = input.split(","); // 2P

        if(inSplit[inSplit.length-1].charAt(0)=='0') // 1P
            inSplit[inSplit.length-1] = "+49" + inSplit[inSplit.length-
1].substring(1); // 2P

        StringBuilder sb = new StringBuilder();     // 3P
        for (int i = 0; i< inSplit.length; i++) {
            if (i==0)
                sb.append(inSplit[i]);
            else
                sb.append("," + inSplit[i]);
        }
        return sb.toString(); // 1P
    }

    // ToDo: Complete the main() method so that the format() method is
called for all of
    //  the three contacts. Both before and after, the string is to be
output to the console.
    // 5P
    public static void main(String[] args) {
        String[] contactCSV = {
                "Berger,Frank,031232",
                "Huber,Pamela,083882",
                "Maier,Sonja,029399"
        };
```

```
        for (int i = 0; i< contactCSV.length; i++) {
            System.out.println("Before: " + contactCSV[i]);
            contactCSV[i] = format(contactCSV[i]);
            System.out.println("After: " + contactCSV[i]);
        }

    }
}
```