Prof. Dr.-Ing. Ralf Schenkel
Martin Blum
Trier University

Wintersemester 2024/2025

Exercises for the Class

Elements of Computer Science: Programming

Assignment 9

Submission of solutions until 16.01.2025 at 10:00
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way in order to get a point!
- Please comment your solutions, so that we can easily understand your ideas!
- If you have questions about programming or the assignments, just ask you teachers!
- **Submission that can't be compiled are graded with 0 points!**

Exercise 1 (Evaluation: predefined `main` method)

Familiarize yourself with the methods of the `String` class[1]:

1. Create a method

   ```
   static int countOccurrence(String haystack, String needle)
   ```

   that counts at how many positions the string `needle` occurs as a substring of `haystack` For example, the call `countOccurrence("abbbba", "bb")` should return the value 3.

2. Create another method

   ```
   static String extractTag(String s)
   ```

   with the following property: For an argument `s` of the form

   prefix '`[`' *infix* '`]`' *suffix*

   the method `extractTag` should return the substring *Infix*. You can assume that '`[`' and '`]`' both occur exactly once in the correct order in `s`.

   For `extractTag("1234[5678]9")` the string `5678` should be found and returned.

Exercise 2 (Evaluation: Predefined `main` method)

This task deals with two-dimensional (**`double`**) arrays and how they can be utilized as matrices. You should implement two methods that make it possible to multiply two matrices [2] together:

[1] https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

[2] https://en.wikipedia.org/wiki/Matrix_multiplication

```
boolean isValid(double[][] matrix1, double[][] matrix2)
double[][] mulMatrices(double[][] matrix1, double[][] matrix2)
```

The method `isValid()` is used to determine if a matrix multiplication is possible. Remember, that you can only multyply two matrices if the number of collumns of `matrix1` is equal to the number of rows of `matrix2`.

The method `mulMatrices()` is used to implement the actual matrix multiplication. It returns a two-dimensional **`double`** array, containing the result of the multiplication. In case that both matrices cannot be multiplied because `isValid()` returns **`false`**, simply return the **`null`** reference.

Exercise 3 (Evaluation: predefined `main` method)

First implement a class `Dice` with the following methods analogous to the coin toss (example `K5B01E_CoinToss` from the lecture):

- A method `throwDice()` randomly assigns a new number (equally distributed) between 1 and 6 to the dice.
- A method `pips()` should return the value thrown by the dice as **`int`**. It must always return a number between 1 and 6!
- Both `throwDice()` and `pips()` should be accessible for users.
- In addition, implement a method in `Dice`

  ```
  public static int pipSum(Dice d1, Dice d2),
  ```

  that returns the total value of both dice.

Write another class `Mia` that internally stores two `Dice` objects. These should be created as follows in the constructor of `Mia`:

```
public class Mia {
    private Dice d1, d2;
    public Mia() {
        d1 = new Dice();
        d2 = new Dice();
    }
}
```

The class should also provide the following methods:

- **`public int`** `valueAndThrowDice()` interprets the current values of the two dice according to Mia rules[3]. In addition, the method should throw both dice again, and then return the interpreted value *of the old throw*.
- **`public static boolean`** `isMia(`**`int`** `value)` checks whether `value` is the maximum Mia result.

[3] See `https://en.wikipedia.org/wiki/Mia_(game)`

- **public static boolean** isDouble(**int** value) tests value to see if it is a doublet.
- **public static boolean** isLess(**int** a, **int** b) compares the two values a and b according to Mia rules and returns **true** exactly when a represents a smaller value than b in this sense.

  Hint: Implement this method by using isMia(**int** value) and isDouble(**int** value)

Watch out: The evaluation only shows you whether you have completed all subtasks; it does not check if your program's logic is correct!

Exercise 4 (Evaluation: predefined main method)

Write a class UserManagement to simulate the management of user names in a computer, in particular password management.

The class shall contain two *private* arrays id and pw of 100 strings each to store the data.

The following methods are to be implemented:

- A constructor UserManagement()
  id and pw are associated. Pay attention to what initial values the individual array components id[i] and pw[i] are set!
- **boolean** newUser(String name, String password)
  This can be used to create a new user with the specified pair of name and initial password.

  If the user already exists, no change should be made in the data and the value false should be returned. The same applies to the case that more than 100 users would exist at the same time together with the new user or that the name consists of less than four characters. Otherwise the user will be saved accordingly and true will be returned. The password may be any non-empty string here.
- **boolean** checkPassword(String name, String password)
  This checks if the specified password matches the user name.
- **boolean** changePassword(String name, String oldPw, String newPw)
  This can be used to change the user name's password if oldPw is correct and the new password newPw has a length of at least eight characters. If the change is successful, the value true is returned, otherwise false.
- There is an administrator password that applies to all objects of this class. This password can be set once by **void** setAdmin(String adminPw), again requiring at least eight characters.

  Any attempt to change a previously set administrator password should be logged with a warning "unauthorized access" on the console.

  On the other hand, attempts to set an unset administrator password to a value that is too short should be ignored completely.

- The administrator password can be used in objects of the class `UserManagement`. Passwords can be set to any non-empty value, using the following method:
  `void` `setPassword(String adminPw, String name, String newPw)`

You have to think for yourself where to put the modifiers `public`, `private` or `static` (matching the given `Test` class). For the sake of simplicity, deletion of users is not intended.

Prof. Dr.-Ing. Ralf Schenkel
Martin Blum
Trier University

Wintersemester 2024/2025

Exercises for the Class

Elements of Computer Science: Programming

Assignment 10

Submission of solutions until 23.01.2025 at 10:00
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way in order to get a point!
- Please comment your solutions, so that we can easily understand your ideas!
- If you have questions about programming or the assignments, just ask you teachers!
- **Submission that can't be compiled are graded with 0 points!**

Exercise 1 (Evaluation: predefined `main` method)

See the class `Queue` from example `K5B05E_Queue_Linked` of the lecture. Extend `Queue` with the following functionalities:

- **`public void`** `append(`**`int`**`[] newData)`
  This function appends all values from the `newData` field to the queue.
- **`public int`** `size()`
  This function outputs how many elements are currently stored in the queue.
- **`public int`** `elementAt(`**`int`** `position)`
  This function outputs the value at the position `position` of the queue (index 0 represents the first element). If `position` is negative or the queue contains too few elements, -1 is returned.
- **`public int`** `contains(`**`int`** `m)`
  This function is to determine whether the value `m` is contained in the queue. If it is included, its position in the queue is returned, if not -1.
- **`public static`** `Queue concat(Queue q, Queue p)`
  This static function should concatenate the two queues, so that first the elements from `q` and then the elements from `p` are contained in the resulting queue. Both `q` and `p` must remain unchanged.
- **`public int`**`[] toArray()`
  This function is intended to create an array containing all elements of the queue. The queue itself should be empty afterwards.

`QueueTest` may be changed for your own tests, but for the grading a new version of `QueueTest` will be used, which will include more extensive tests. Therefore your program must work with the original version of `QueueTest` as well!

Exercise 2 (Evaluation: predefined `main` method)

Take another look at the class `Queue`, but now from the example `K5B05E_Queue_Array` of the lecture. Extend `Queue` with the same functionalities as in the previous task:

```
public void append(int[] newData)
public int size()
public int elementAt(int position)
public int contains(int m)
public static Queue concat(Queue q, Queue p)
public int[] toArray()
```

However, you should also change the implementation of the queue so that queues can never become full when elements are added:

- In the constructor, the value 4 is to be used as the initial size of the array used in the queue.
- A dynamically growing array should be used for storage (similar to `StringBuffer`): If the array of a queue is full (usually by `enQueue()` invocations), a new array of double size is created, into which all data is transferred.
- Similarly, if the queue is shrinking, the array is to be replaced by a new array with half the capacity: If an element is removed and the number of elements in the queue is then less than or equal to a quarter of the current capacity, the size of the array in the queue is to be halved. However, the capacity should not be less than value 4.
- In order to trigger these capacity changes manually, two methods

  ```
  public void setCapacity(int n)
  public int getCapacity()
  ```

  are to be implemented which can be used to manually adjust or query the current capacity. A call `setCapacity(n)` should be ignored if $n < 4$ or $n \leq$`size()` is true.

The note from the previous task for the files `Queue.java` and `QueueTest.java` also applies here.

Prof. Dr.-Ing. Ralf Schenkel
Martin Blum
Trier University

Wintersemester 2024/2025

Exercises for the Class

Elements of Computer Science: Programming

Assignment 11

Submission of solutions until 30.01.2025 at 10:00
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way in order to get a point!
- Please comment your solutions, so that we can easily understand your ideas!
- If you have questions about programming or the assignments, just ask you teachers!
- **Submission that can't be compiled are graded with 0 points!**

Exercise 1 (Evaluation: predefined `main` method)

The goal of this task is to familiarize yourself with nested objects. You are already familiar with examples from the lecture in which classes "store" instance variables such as arrays, integers and others. However, classes can of course also store objects of other classes. To illustrate this in a practical example, your task is to implement a simple family tree. In order to simplify the task, we have not included aspects such as more complex family structures, same-sex parents and gender.

To do this, implement the following components:

- Implement a class `Human`, which has a name (`String name`) and a unique ID (**`int`** `id`). The ID is required in the event that family members have the same name. The ID should start at 1000.
- Implement a constructor for the class `Human` which only receives a name as input. The ID should be set automatically.
- Next, implement the classes `Man` and `Woman`. Both classes are subclasses of the class `Human`. In this case, they do not contain any additional information other than that of the superclass. With the two subclasses, we are able to differentiate between male and female humans in this implementation.
- Next, extend the class `Human` so that it stores the mother (`Woman mother`) and the father (`Man father`) of a human in addition to the name and an ID. The previously implemented constructor should also be extended so that a mother and a father can be passed and set accordingly.
- For this part, it should be possible to create objects of the class `Human` without information about the parents. To do this, you must implement corresponding constructors. Please note that a mother must always be an object of the class `Woman`. The same applies to the father. You can find examples of the constructors to be implemented in the evaluation.

Exercise 2 (Evaluation: predefined `main` method)

An abstract class `Recipe` with four methods is given: `recipeName()`, `ingedientsList()`, `equipment()` and `preparationTime()`. It also contains a (static) `main` method for a simple test of the solution.

Derive three classes `Pizza`, `Sandwich` and `Risotto` from `Recipe`. The `recipeName` and special ingredients should be placed in the constructors. General ingredients, the required cooking utensils and the preparation time result from the respective class:

- `Pizza`:
  General Ingredients: Yeast Dough, Tomatoes, Mozzarella, Oregano
  Equipment: Oven, Pizza Plate, Rolling Pin
  Preparation Time: 40
- `Sandwitch`:
  General Ingredients: Butter
  Equipment: Knife, Cutting Board
  Preparation Time: 3
- `Risotto`:
  General Ingredients: Rice, Vegetable Bouillon, Parmesan
  Equipment: Pot, Stirring Spoon
  Preparation Time: 40

The `ingedientsList` should consist of the general ingredients for the basic recipe and the special ingredients from the constructor.

The default `main` method should therefore lead to the following output:

```
Name:         Pizza Salami
Ingredients: Salami, Yeast Dough, Tomatoes, Mozzarella, Oregano
Equipment:   Oven, Pizza Plate, Rolling Pin
Duration:    40.0
Name:         Pizza Diavolo
Ingredients: Salami, Peperoni, Yeast Dough, Tomatoes, Mozzarella,
             Oregano
Equipment:   Oven, Pizza Plate, Rolling Pin
Duration:    40.0
Name:         Ham Sandwich
Ingredients: Brown Bread, Ham, Butter
Equipment:   Knife, Cutting Board
Duration:    3.0
Name:         Toast Hawaii
Ingredients: Pineapple, Cheese, Butter
Equipment:   Knife, Cutting Board
Duration:    3.0
Name:         Pumpkin Risotto
Ingredients: Pumpkin, Rice, Vegetable Bouillon, Parmesan
Equipment:   Pot, Stirring Spoon
Duration:    40.0
```

Exercise 3 (Evaluation: predefined `main` method)

Model different road users in a hierarchical structure. To do this, implement the following model:

a) An abstract class `Traffic`, from which all other classes are derived:

- Road users have common fields: `name` (string), `wheels` (int), `drivenKilometers` (double) and `maxSpeed` (int).
- Write a constructor that receives the expected parameters and writes them into the variables provided.
- A vehicle of class `Traffic` also has an abstract method with the signature

  ```
  abstract boolean licenseNeeded()
  ```

  This method should return whether a driving license is required, depending on type of vehicle (car, motorcycle or bicycle).
- Implement a method with the signature

  ```
  public void addKilometer(int km).
  ```

  This method should be used to increase the number of kilometers traveled.
- `String toString()` should return a string of the following form:

  ```
  name, drivenKilometers, maxSpeed, wheels
  ```

b) Derive from the class `Traffic` the classes `Car`, `Motorbike` and `Bicycle`

- A **bicycle** has only two wheels and a maximum speed of 30 km/h. Furthermore, a bicycle can be used by a courier. Therefore this class should store a value `carrier` (boolean). Design the constructors of the class so that you do not need to specify a number of tires or a maximum speed.
- Like a bicycle, a **motorbike** has only two wheels, but can travel at any speed (depending on the model, etc.). When implementing the constructor, no information about the number of wheels is necessary.
- A **car** has four wheels and can have any maximum speed like a motorcycle.

c) Derive from the class `Car` a class `VintageCar`. This class stores a value `year` (**`int`**) with the year of manufacture of the car. Design an appropriate constructor.

d) Extend all classes derived from `Traffic` (including vintage car) with a `toString()` method. This method should call the `toString()` method of the superclass, but first specify the type of the vehicle. This results in a string of the following form:

```
Car:  name, drivenKilometers, maxSpeed, wheels
```

A vintage car should use the method of the superclass (`Car`) too add the following label:

```
Car:  name, drivenKilometers, maxSpeed, wheels (Vintage)
```

e) Implement two interfaces `Collectible` and `ProtectiveClothing`

- Since any motor vehicle could be a collectible, `Car`, `Motorbike` and `VintageCar` should have the interface `Collectible` The interface has only one method with the signature **`int`** `calcValue()`. The value of a collector's item is, for the sake of simplicity, calculated by multiplying the driven kilometers by the number of wheels.
- The second interface should contain a method with the signature

  ```
  boolean needsProtectiveClothing()
  ```

  All classes derived from `Traffic` should have this interface and implement the method `needsProtectiveClothing()`. Cars and vintage cars do not need special protective clothing.

Attention: The tasks are intentionally underspecified. Find meaningful and suitable solutions and familiarize yourself with the `main` method!

Prof. Dr.-Ing. Ralf Schenkel
Martin Blum
Trier University

Wintersemester 2024/2025

Exercises for the Class

Elements of Computer Science: Programming

Assignment 12

Submission of solutions until 06.02.2025 at 10:00
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way in order to get a point!
- Please comment your solutions, so that we can easily understand your ideas!
- If you have questions about programming or the assignments, just ask you teachers!
- **Submission that can't be compiled are graded with 0 points!**

Exercise 1 (Evaluation: predefined `main` method)

Implement a calculator working on input given in the *Reverse Polish notation*[1].

Example: The string

```
4 5 + 3 23.5 8 - * /
```

is equal to the formula $(4+5)/(3*(23.5-8))$. The individual tokens are separated by a space character and interpreted from left to right. Implement a method

```
double calculate(String s)
```

which is able to process formulas in the *Reverse Polish notation*. Use `java.util.Stack`[2] to store all numbers as `Double` values. Proceed as follows:

1. If a token is a number, put it on top of the stack.
2. If a token is an operator (`+`, `-`, `*` or `/`), take the uppermost two numbers off the stack, calculate the result and put it on top of the stack. If the operator is invalid, throw an `Exception`.

   The uppermost element always corresponds to the 2nd argument, the second element from top corresponds to the 1st argument.
3. Before applying an operator, test if there are enough arguments on the stack to carry out the calculation. If this is not the case, throw an `Exception`.
4. After each step, output the contents of the stack. You may use the `toString()` method of the `Stack` class for this.
5. If all input has been processed and there is only one value left on the stack, output it as the result of the calculation. If the stack is empty or contains ≥ 2 values, throw an `Exception`.

[1] https://en.wikipedia.org/wiki/Reverse_Polish_notation

[2] https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

Example output:

```
Input: 4 5 + 3 23.5 8 - * /
Stack: [4.0]
Stack: [4.0, 5.0]
Calculating: 4.0 + 5.0
Stack: [9.0]
Stack: [9.0, 3.0]
Stack: [9.0, 3.0, 23.5]
Stack: [9.0, 3.0, 23.5, 8.0]
Calculating: 23.5 - 8.0
Stack: [9.0, 3.0, 15.5]
Calculating: 3.0 * 15.5
Stack: [9.0, 46.5]
Calculating: 9.0 / 46.5
Stack: [0.1935483870967742]
0.1935483870967742
```

Exercise 2 (Evaluation: predefined `Test` class)

a) Write an abstract class `Pet` that stores the weight and name of a pet. The class should not have a getter and setter method, but its variables should be declared as **`public`**. Additionally, the following methods should be provided:

- **`void`** `feed(`**`double`** `g)`: The weight of the pet is increased by `g`.
- **`void`** `dropWeight(`**`double`** `g)`: The weight of the pet is reduced by `g` due to the daily basal metabolic rate.
- `String toString()`: Should be implemented as an abstract method.
- `String makeNoise()`: An abstract method that returns a typical sound for the respective animal.
- **`int`** `compareTo(Pet pet)`: Should be implemented as an abstract method to compare two pets by weight. The method shall return -1 if the weight of **`this`** is less than that of `pet`, 0 if both objects have the same weight, and otherwise a 1.

b) Write two more classes `Cat` and `Dog` which extend the class `Pet`:

- The typical noise should be "Meow" or "Woof". Example:

```
Garfield: Meow
Lucky: Woof
```

- The class `Dog` shall include a method for walking a dog. The method has the signature

```
public void walkTheDog()
```

  and satisfies two tasks: (1) the dog should lose 0.2 kg of weight while walking and (2) the method should provide an output of the following form:

```
Lucky goes for a walk and loses weight
```

- Familiarize yourself with exception handling by deriving the two classes

  ```
  NotComparableException
  TooHeavyException
  ```

  from the `Exception` class already contained in Java:

  – The `TooHeavyException` is designed to prevent dogs weighing more than 15 kg from being walked. In this example, they want to eat and sleep all day long! The `TooHeavyException` class has a constructor of the following form:

    ```
    TooHeavyException(double weight)
    ```

    When "throwing" this exception, the overweight of the dog should be calculated and stored in a variable. The class also provides a method, `String getErrMsg()`, which returns a string of the following form:

    ```
    Exception:  Dogs with overweight don't go for walks
    ```

    Extend the method `walkTheDog()` so that too heavy dogs "throw" an exception.

  – The `NotComparableException` is intended to prevent that animals of different species are compared with each other (e. g., dog with cat). To do this, implement this class so that it has a constructor of the following form:

    ```
    NotComparableException(Pet pet)
    ```

    When "throwing" this exception a string is to be stored in a variable. This string depends on whether dogs are compared with cats or other different animal species. The string to be stored has accordingly one of the following forms:

    ```
    Exception: You can not compare cats and dogs
    Exception: No comparison possible
    ```

    The class provides, like the exception before, a method `String getErrMsg()`. It returns the previously saved string.

    Now extend the methods `compareTo(Pet pet)` of the classes `Cat` and `Dog` so that an exception is "thrown" if animals of different species are compared.

- The method `toString()` shall be overwritten so that a string of the following form is returned:

  ```
  Cat: Garfield weighs 4.2 kg
  Dog: Lucky weighs 9.8 kg
  ```

Exercise 3 (Evaluation: predefined `main` method)

Given is a class `Evaluation` which contains a `main` method. The `main`-method asks the user to specify a number of persons to be created. It then reads information about the `Person` objects to be created and creates a corresponding object.

Your task is first to implement the class Person. The class stores an automatically incrementing ID (**int** id, starting at 1000), the name of a person (String name), the age of a person (**int** age), and the residential address (String address). Furthermore implement

- a suitable constructor of the form

  Person(String name, **int** age, String address)

- and a method String toString() which returns a string of the following form:

  <ID>, <NAME>, <AGE>, <ADDRESS>

  Here <ID>, <NAME>, etc. are just placeholders and should be replaced with the respective values of the objects.

Next, extend the constructor of the class Person so that if the passed name of a person is **null** an NullPointerException is thrown. As message they pass the string "Names are not allowed to be null" to the exception constructor.

Next, implement an exception AgeTooLowException. Note that the exception must also be "identified" as such, otherwise it is just a normal class. The exception contains a constructor of the form AgeTooLowException(String message, **int** age). The constructor should call the super constructor and pass a string of the form

"<MESSAGE>␣<AGE>".

Again, extend the constructor of the Person class so that an AgeTooLowException is thrown in case of a negative age. Pass the string "Age must be greater or equal to zero:" as well as the age of the person to the exception.

Next, implement an exception IllegalAddressException. The exception contains a constructor of the form

IllegalAddressException(String message, String address).

The constructor should call the super constructor and pass a string of the following form as before:

"<MESSAGE>␣<ADDRESS>".

Again, expand the class Person so that in case of an invalid address string an exception of type IllegalAddressException is thrown. An address is invalid if it does not consist of two parts:

Streetname 12, 1234 City

Note: You do not have to use a regular expression to check if it is a valid address. You only have to test if the address string consists of two parts. In case of invalid formatting, pass the exception the string "Address is not correctly formatted:" and the address of the person.

As a last step, you should extend the main-method so that in the **for**-loop all possible exceptions of the Person class are caught. Then output the "message" of the exception to the console.

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 01

Submission of solutions until 3:30 p.m. at 08.11.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 (Evaluation: Numbers)

You receive an executable Java program as a template for the task. In its original state, this predefined program reads two numbers from the console and calculates the sum and difference of the numbers entered from them.

Modify the lines 19 and 22 in this section of the program so that the program will instead of calculating the sum and difference of the numbers,

- at line 20 return three times the first number and
- at line 23 return the sum of the second number and twice the first number.

For some test cases, you can automatically test the correctness of the way your program works.

```
 1 import java.util.Scanner;
 2
 3 public class Exercise {
 4   public static void main(String[] args) {
 5   /* this program reads two numbers from the console and calculates the sum and
 6   difference of the numbers entered from them*/
 7
 8     Scanner sc = new Scanner(System.in);
 9     int    firstNumber;
10     int    secondNumber;
11     int    result;
12
13     System.out.println("First␣Input:␣");
14     firstNumber = sc.nextInt();
15
16     System.out.println("Second␣Input:␣");
```

```
    secondNumber = sc.nextInt();

    result =  firstNumber + secondNumber;
    System.out.println("First␣Result:␣" + result);

    result    =  firstNumber - secondNumber;
    System.out.println("Second␣Result:␣" + result);
  }
}
```

Exercise 2 (Evaluation: Numbers)

The task is based on the first program example `K2B01_first_Example` of the lecture.

Modify the program so that the squares of all natural numbers between 0 and 10 are replaced by a list of even numbers between 0 and 10. You can have the system automatically test the correctness of the operation of your program again.

As in the template the calculation must use a (while) loop.

```
public class K2B01E_first_Example {
  public static void main(String[] args) {
    int index;
    int count;

    index  =  0;
    count = 10;

    /* Just the following lines should be changed */
    while ( index <= count )
    {
      System.out.println( index * index );
      index = index + 1;
    }
  }
}
```

Exercise 3 (Evaluation: Numbers)

This exercise builds on the previous exercises. The predefined program can be compiled, but has no function. Complete the program as follows:
As in the first exercise, two numbers (naming “a” and “b”) are to be read in. Similar to exercise 2, all *even* numbers between 0 and a (inclusive) have to be *multiplied* by b. The result needs to be printed afterwards to the console.

Even if it is not algorithmically necessary, this task must be solved by using the modulo operator (`%`). Modulo can be used to determine if a number is even or odd by calculating the remainder of an integer division. For example:

$$\text{By integer division we have } 27/2 = 13, \text{ remainder } 1$$
$$\text{because } 13 * 2 + 1 = 27$$
$$\text{By using modulo we get } 27 \ \% \ 2 = 1$$
$$\text{For an even number we get } 26 \ \% \ 2 = 0$$

For more information regarding the modulo operator check the following link: `https://en.wikipedia.org/wiki/Modulo_operation`
There are again test cases to check the correctness.

```
import java.util.Scanner;
public class Exercise {
  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int a;
    int b;
    int index;

    index = 0;

/* From here you should complete the program code */


/* The two brackets must remain! */
  }
}
```

Exercise 4 (Evaluation: Numbers)

The exercise is based on the previous tasks and the example program `K2B06_Arrays.java`. The predefined program initially has the same functionality as this example program, but has already been extended in the first lines to include the scanner for entering numbers.

Modify the middle part of the program as follows:

- The program should scan a number.
- This number is used to determine the size of the field/array.
- The individual field elements are not set to the square of the index, but with the double of the index.

```
public class K2B06E_Arrays {
  public static void main(String[] args) {
    int index;
    int count;

    int[] array;
/* Changes are only allowed in the following 10 lines! */
```

```
    count=5;
    array = new int[count];

    index=0;
    while (index<count)
    {
      array[index] = index*index;
      index=index+1;
    }
/* From here on no changes are allowed any more! */
    index=0;
    while (index<count)
    {
      System.out.println(array[index]);
      if (index<count) System.out.println("-");
      index=index+1;
    }
  }
}
```

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 02

Submission of solutions until 3:30 p.m. at 15.11.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 (Evaluation: Numbers)

The task is essentially based on the fourth task of the first assignment:

(a) The given program reads a number (`size`) via the scanner andthen creates an array with the size of the inputted number

(b) At each index in the array, the array should be given the value of the square of the index

(c) The values in the array should then be printed in the console

Now modify (b) and (c) in this program as follows:

(b') Change the program so that as many numbers are stored in the array as the size of the array allows. The numbers should be read by the scanner and stored in the array.

(c') The output should take place in reverse order (i.e. the last input as the first output).

The input "`5 0 1 2 3 4`" therefore becomes "`4 3 2 1 0`". Remember, the first number (namely `5`) is only the size of the array and therefore is not part of the sequence. When evaluating, you will see further example sequences that your program must reproduce.

```java
import java.util.Scanner;
public class Inverse_Output {
  public static void main(String[] args) {
/* The task is essentially based on the fourth task of the first exercise: */
    Scanner sc = new Scanner(System.in);
    int index;
    int size;
    int[] array;
/* Part (a) */
    size = sc.nextInt();
    array = new int[size];

/* Part (b)  */
    index = 0;
    while ( index < size )
    {
      array[index] = index * index;
      index = index + 1;
    }

/* Part (c)  */
    index = 0;
    while ( index < size )
    {
      System.out.print( array[index] + "␣" );
      index = index + 1;
    }

/* The following brackets must be retained. */
  }
}
```

Exercise 2 (Evaluation: Numbers)

The exercise builds on the previous one:

- The size and then the corresponding values are to be read into a field again.
- The new part: After that another number (the variable `barrier`) is entered.
- Now all positions in the field are to be found (and output), at which a larger number than this barrier occurs.
- The input `5 22 10 41 35 27 30` should result in an output of `2 3`, since the values stored at the indexes 2 and 3 are 41 and 35, which are larger than the barrier (the last inputted number) `30`.
- To compare the numbers, you can use the following statement:

  ```
  if ( barrier < array[index] ) ...
  ```

  This line is analogous to line 22 in the example program `K2B06E_Arrays.java`.

```
import java.util.Scanner;

public class Exercise {
  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int index;
    int size;
    int[] array;
    int barrier;

/* The solution has to be typed in the following area: */




/* The following brackets must be retained. */
  }
}
```

Exercise 3 (Evaluation: Numbers)

The exercise is based on the previous one: As before, the program should read as first value the size of an array and afterwards create an array using the entered size. Next, the programm should read as many numbers as the size of the array allows via the scanner and store them in the created array. Your task will be to test **how often** the value `index` is in the field at the position `index`.

You can program the necessary test for equality as follows:

```
if ( array [index] == index ) ...
```

The input `5    2 1 4 3 2` would result in the output `2`, because at the two positions 1 and 3 in the field (starting with position 0) there are also the values 1 and 3.

```
import java.util.Scanner;

public class Exercise {
  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int index;
    int size;
    int[] array;
    int counter;

/* The solution has to be typed in the following area: */




/* The following brackets must be retained. */
  }
}
```

Exercise 4 (Evaluation: Numbers)

As before, the program should read as first value the size of an array and afterwards create an array using the entered size. Next, the programm should read as many numbers as the size of the array allows via the scanner and store them in the created array.

After that, you should start a “scavenger hunt” in the array, starting with position `index=0`: The value stored in the array at an index defines the next index that should be searched at. If, for example, you search at index 2, and the value stored there is 4, the next index you should search at is 4 (a more complete example is shown below). Use the setting `index = array[index}` to determine the next position to be considered. Output the first 10 indices that you search at in this manor

Example: If `5    2 4 3 1 0` is entered, your program should do as follows:

- array[0] -> 2, therefore the next field in the array to be considered is field 2;
- array[2] -> 3, next field is 3;
- array[3] -> 1, next field is 1;

- array[1] -> 4, next field is 4;
- array[4] -> 0, there we will go back to 0;
- array[0] -> 2, and so on...

from here on, the values found are repeated.

The correct output would then be (10 values!): `0 2 3 1 4 0 2 3 1 4`

```
import java.util.Scanner;

public class Exercise {
  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int index;
    int size;
    int[] array;
    int counter;

/* The solution has to be typed in the following area: */




/* The following brackets must be retained. */
  }
}
```

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 03

Submission of solutions until 3:00 p.m. at 24.11.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 Evaluation: Numbers

In the given program a size and then a corresponding sequence of values are inputted into a field. (The corresponding part of the program is already preset in the program and should not be changed.)

Very often it is necessary to search for data in an array. The structure of the corresponding program parts then usually consists of three components: (a) the preparation of the search, (b) the actual search itself, and (c) an evaluation of the search.

As an example, you should now find and output the largest value among the numbers stored in an array. In this case, (a), (b) and (c) result as follows:

(a) Use a variable in which the maximum value can be stored and assign it a sufficiently small value (here, for example, any of the entered numbers (in the array) as a candidate for the maximum).

(b) Compare all saved values one after the other to see whether they are larger than the candidate found for the maximum (then: update the candidate if necessary).

(c) At the end of the loop: Output the maximum value found in this way.

- The Input **5 0 1 2 3 4** (i.e. 5 values from 0 to 4) returns the output **4**.
- The Input **9 4 6 4 2 0 2 8 6 2** returns the output **8**.
- The Input **1 10** returns the output **10**.

- The Input **1 0** returns the output **0**.
- The Input **2 10 5** returns the output **10**.

Exercise 2 Evaluation: Numbers

Again, first a size (≥ 0) and then a corresponding number of values are read into a field (which is already contained in the program).

For many purposes it is important that the data is sorted in an array (or other datatypes you will learn), i.e. `array[index] <= array[index+1]` always applies. You will learn many sorting algorithms later. But first we just want to know how unsorted the data is.

Now determine the number of ”simple inversions” in the array of the data you read, i.e. the number of positions where `array[index] > array[index+1]` applies. First think about what the three components (a)-(c) of the previous exercise should look like. You can assume that the array will contain at least two numbers, i.e. the size of the array is at least 2.

- For the input **5 0 1 2 3 4** (i.e. 5 values from 0 to 4, all sorted in this case) there are 0 simple inversions; Therefore the output should be **0**.
- For the input **5 4 3 2 1 0** there are 4 positions with simple inversions, because $4 > 3$, $3 > 2$, $2 > 1$ and $1 > 0$; Therefore the output is **4**.
- The input **6 0 1 0 1 0 1** contains three simple inversions; The output is **2**.
- The input **6 0 0 4 4 2 2** contains only one simple inversion at $4 > 2$; The output is **1**.
- The input **2 42 42** contains no simple inversions; The output is **0**.

Exercise 3 Evaluation: Numbers

Again, first a size (≥ 0) and then a corresponding number of values are read into a field (which is already contained in the program).

As in the previous exercise, you should again search for the maximum value in the array, but you should now additionally specify how *often* this maximum value is contained in the array.

Examples:

- The input **5 0 1 2 3 4** (i.e. 5 values from 0 bis 4) returns the output **4 1**, because the greatest value 4 only occurs once.
- The input **9 4 6 4 2 0 2 4 6 2** returns the output **6 2**.
- The input **7 10 10 10 10 1 10 10** returns the output **10 6**.

- The input **1 0** returns the output **0 1**.
- The input **2 10 5** returns the output **10 1**.

Exercise 4 Evaluation: Numbers

Again, first a size (≥ 0) and then a corresponding number of values are read into a field (which is already contained in the program).

As with the above exercise, you should again search for the maximum value in the field, but you should now additionally specify *which indexes* contain this maximum value in the field.

Hint: Use two loops that are executed one after the other.

Examples:

- The input **5 0 1 2 3 4** (i.e. 5 values from 0 to 4) returns the output **4 4**, because the greatest value 4 is found on position 4 in the array.
- The input **9 4 6 4 2 0 2 4 6 2** returns the output **6 1 7**, because 6 is found on position 1 and 7 in the array.
- The input **7 10 10 10 10 1 10 10** returns the output **10 0 1 2 3 5 6**.
- The input **1 0** returns the output **0 0**.
- The input **2 10 5** returns the output **10 0**.

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 04

Submission of solutions until 3:00 p.m. at 01.12.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 Moon Landing (Evaluation: Numbers)

From the early days of programming originates the following small console game to simulate a moon landing:

A manned Lunar Module is to land on the moon. Important are the height $H(i)$ at the time i and the velocity $V(i)$ at the same time. i is the number of seconds since the beginning of the landing attempt.

At the beginning of the landing phase (time: $i = 0$ seconds) the Lunar Module has a height of $H(0) = 200\,\mathrm{m}$ above the surface and a starting speed of $V(0) = 0\,\frac{\mathrm{m}}{\mathrm{s}}$, i.e. it stands still at $200\,\mathrm{m}$ height above the ground. Due to the gravity of the moon it starts to fall (the velocity increases by $5\,\frac{\mathrm{m}}{\mathrm{s}}$ every second). Without countermeasures, it would hit the surface and be destroyed.

For the simplification we will not give the units m, s and $\frac{\mathrm{m}}{\mathrm{s}}$ in the following, moreover only integers are possible values. This allows you to use the already known number type `int`.

The pilot of course wants to gently touch down on the surface, so $H(j) = 0$ at a time j which is still unknown. The passengers should not be injured, therefore the speed $V(j)$ must not be more than 10 at the time of landing.

To achieve this, the pilot can give the Lunar Module limited counter thrust ($0 \leq T \leq 10$). To simplify the procedure, we assume that this thrust can be re-adjusted exactly once per second. In every flight second there is a thrust $T(i)$ entered by the pilot. If the pilot enters a thrust of $T(i) > 10$, the system reduces this to the maximum value of $T(i) = 10$. On the other hand a negative thrust $T(i) < 0$ is not possible, then $T(i) = 0$ is set.

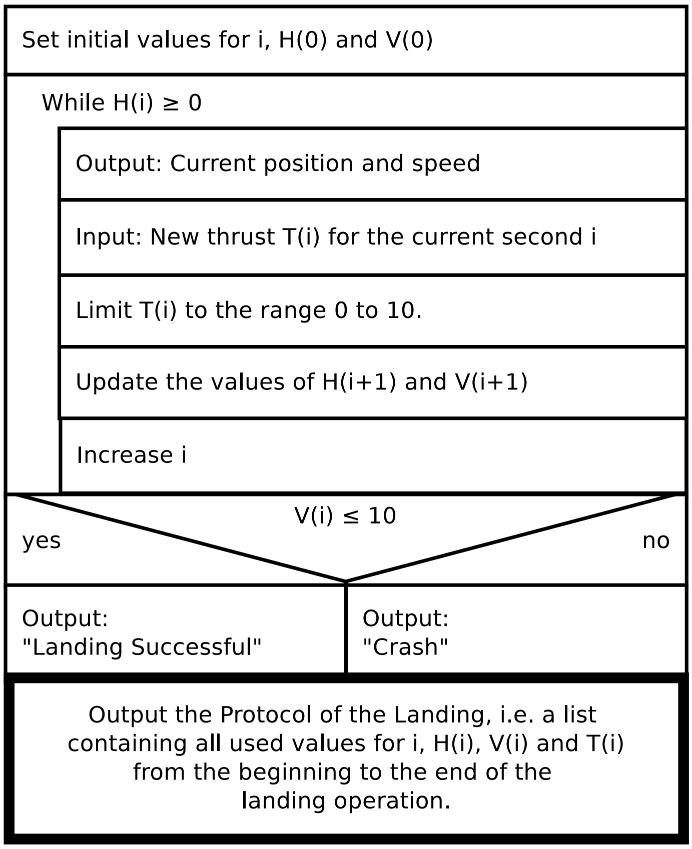
This results in new values for altitude and speed in every second, which are given by the following equations:

$$H(0) = 200, \qquad V(0) = 0$$

and the change of the values from each second i to the next second $i+1$:

$$H(i+1) = H(i) - V(i)$$
$$V(i+1) = V(i) + 5 - T(i)$$

Now implement the algorithm for interactive moon landing given in the following structure chart in a Java program:



Set initial values for i, H(0) and V(0)
While H(i) ≥ 0
Output: Current position and speed
Input: New thrust T(i) for the current second i
Limit T(i) to the range 0 to 10.
Update the values of H(i+1) and V(i+1)
Increase i
V(i) ≤ 10
yes
no
Output: "Landing Successful"
Output: "Crash"
Output the Protocol of the Landing, i.e. a list containing all used values for i, H(i), V(i) and T(i) from the beginning to the end of the landing operation.

Notes:

- In the given program there are already 3 fields H, V, T (each with 1000 components) defined, in which you can store the corresponding values. You can assume that the landing does not take $1000s$ and that the fields are large enough.
- To enter the values for the thrust, please use the Scanner.
- The time elapsed in the game is of course completely independent of the real time, the value of i is determined by the number of entries you make during the game.
- How you implement the last part of the structogram ("output of a protocol") is up to you.

An example of an attempted landing could look like the following:

```
H = 200, V = 0, Thrust: 0
H = 200, V = 5, Thrust: -1
H = 195, V = 10, Thrust: 0
H = 185, V = 15, Thrust: 0
H = 170, V = 20, Thrust: 0
H = 150, V = 25, Thrust: 0
H = 125, V = 30, Thrust: 0
H = 95, V = 35, Thrust: 0
H = 60, V = 40, Thrust: 20
H = 20, V = 35, Thrust: 20
Crash
Protocol of the values:
i = 0, H = 200, V = 0, T = 0
i = 1, H = 200, V = 5, T = 0
i = 2, H = 195, V = 10, T = 0
i = 3, H = 185, V = 15, T = 0
i = 4, H = 170, V = 20, T = 0
i = 5, H = 150, V = 25, T = 0
i = 6, H = 125, V = 30, T = 0
i = 7, H = 95, V = 35, T = 0
i = 8, H = 60, V = 40, T = 10
i = 9, H = 20, V = 35, T = 10
i = 10, H = -15, V = 30, T = 0
```

(For the evaluation using `moodle` only the sequence of the values in the above example is important, the exact formatting is not important!)

Exercise 2 (Evaluation: Numbers)

Write a Java program for processing fields whose values were given randomly (more precisely: pseudo-randomly), i.e. you don't know what values are stored inside the array.

First the (given) program takes three integers `size`, `limit` and `seed` (which are all > 0) as input.

Then an array `int[] array` of size `size` is created and filled with numbers from the range `0` to `limit`, which are determined randomly. A random number generator is used, which is initialized with `seed`. If you use always the same values for `seed`, the same random numbers will be used. But if you change the value of `seed` also the random values will change completly.

Up to this point, the program has been predefined. You should now find the smallest and largest value in the field and then output it, for example:

```
Array Size:  5
Limit: 20
Seed: 6
Minimum: 1
```

```
Maximum: 18
```

A limit of 20 and a random start value `seed`= 6 lead to the field values `11 16 6 18 1`. If, however, you take 9 as `seed`, the field values `9 16 8 15 19` result, and the output will look as follows:

```
Array Size:  5
Limit: 20
Seed: 9
Minimum: 8
Maximum: 19
```

Note: First write a loop in which the field values are displayed to the console, so that you can see which numbers you are working with! This output must of course be removed afterwards.

Exercise 3 (Evaluation: Numbers)

Modify the solution to the previous task so that it shows at which position the minimum was first seen (smallest corresponding index) and when the maximum was last seen (i.e. largest corresponding index). For the formatting of the output, please refer to the following examples:

```
Array Size:  5
Limit: 20
Seed: 6
Minimum 1 for the first time at position 4
Maximum 18 for the last time at position 3
```

respectively

```
Array Size:  5
Limit: 20
Seed: 9
Minimum 8 for the first time at position 2
Maximum 19 for the last time at position 4
```

Exercise 4 (Evaluation: Numbers)

Modify your solution from the last task. Instead of printing only the minimum and maximum values and their corresponding positions, you have to do the following:

- In case the index of the first appearing minimum is less than the index of the last appearing maximum: Print all numbers stored in the array that occur after the position of the minimum until you reach the maximum (excluding) of the array.

- In case the index of the first appearing minimum is greater than the index of the last appearing maximum: Print all numbers stored in the array that occur after the position of the minimum until you reach the end of the array.

You can assume, that minimum and maximum are not equal and the array is filled with multiple numbers.

For the formatting of the output, please refer to the following examples: A limit of 20 and a seed of 9 lead to the field values `9 16 8 15 19` and the output will look as follows:

```
Array Size:  5
Limit: 20
Seed: 9
Output: 15
```

A limit of 10 and a seed of42 lead to the field values `0 3 8 4 0 5 5 8 9 3` and the output will look as follows:

```
Array Size:  10
Limit: 10
Seed: 42
Output: 3 8 4 0 5 5
```

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 05

Submission of solutions until 3:00 p.m. at 08.12.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 (Evaluation: Numbers/Text)

Write (e.g. based on the previous assignment) a Java program that reads a `int` number $m > 1$ and determines how many prime numbers there are between 2 and m (both inclusive). (If $m = 1$ is entered, the program may again produce any output; the input and reading of m has already been performed in the specified program).

```
79
The number of prime numbers up to this input is 22
fast enough
```

Watch out: For this task, the computing time required for the evaluation is also measured. A fast solution requires less than one second per case for all evaluation examples. Slow solutions take much longer. The faster your program is, the more cases can be successfully evaluated within one second. For example, consider whether the prime number test for a given $n \leq m$ really needs to test all numbers between 2 and $n - 1$.

For slow but otherwise correct solutions there is a maximum of 10 points, the remaining 5 points are awarded on the basis of speed. The time measurement incl. the output **fast enough** resp. **too slow** is given in the program and must not be modified.

Advise: A nested loop solves the problem, but will unfortunately be too slow to get 15 points. Look at the *Sieve of Eratosthenes* for a quick solution.

Exercise 2 (Evaluation: Text)

Write a Java program that reads two numbers f and n as input and generates patterns (from $n \times n$ characters) of the following type as output (similar to example `K3B14E_Flag.java`). Here f specifies the format or the pattern of the output (here only 1 or 2, other values may be treated arbitrarily); $n \geq 5$ is always odd.

Format $f = 1$ as 7×7-Pattern:

```
1 7
.X.X.X.
X.X.X.X
.X.X.X.
X.X.X.X
.X.X.X.
X.X.X.X
.X.X.X.
```

Format $f = 2$ as 7×7-Pattern:

```
1 7
XXXXXXX
XX....X
X.X...X
X..X..X
X.X...X
XX....X
XXXXXXX
```

The patterns are to be created with nested `for` loops. `System.out.print` may only be used in the following three forms (but of course multiple times):

- `System.out.print(".")`,
- `System.out.print("X")` und
- `System.out.println()`.

So you have to build the output from single `.`, `X` and line separators. Also make sure that no line ends are missing and that you do not create unnecessary blank lines.

Exercise 3 (Evaluation: Text)

Write a program that reads a number n between 0 and 999 as `int` value from the scanner. Now convert this number into natural language and display it on the console.

Example:

```
 11 -> eleven
 17 -> seventeen
 34 -> thirty-four
100 -> one hundred
258 -> two hundred and fifty-eight
666 -> six hundred and sixty-six
812 -> eighth hundred and twelve
```

Negative inputs or inputs that are too large should return the output `Wrong Input!`:

```
Input: 258
In Words: two hundred and fifty-eight

Input: 1000
Wrong Input!
```

Watch out: The purpose of this task is not to program all numbers manually, but to find a minimal solution. For programs that have all numbers (or all numbers up to 100) programmed in, points will be deducted!

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 06

Submission of solutions until 3:00 p.m. at 15.12.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 (Evaluation: Numbers)

Add the following five methods to the predefined program:

1. `sub`: three `int` parameters, `int` return value;
   the smallest number is subtracted from the largest number.
2. `mul`: three `int` parameters, `int` return value;
   the two largest numbers are multiplied by each other.
3. `mean`: thre `int` parameters, `double` return value;
   the arithmetic mean $\frac{a+b+c}{3}$ of the three parameters is determined as `double` and returned. Example: $\frac{1+2+2}{3} = 1,666...$
4. `allEqual`: three `int` parameter, `boolean` return value;
   the `boolean` value `true` is returned if all three parameters have the same value, otherwise the value `false`.
5. `prime`: one `int` parameter, `boolean` return value;
   return `true`, if the `int` number is positive and a prime numbers. Otherwise return `false`

You may only change the default `main` method as follows: If you have implemented a subtask, remove the comment characters from the line beginning of the corresponding case of the `switch` statement. This will activate the subtask.

The main method first reads in up to four numbers st, a, b, c. The first number st determines the subtask to be considered. An example can be found on the next page.

Watch out: The methods to be written by you must all be provided with the modifier static!

```
Subtask: 3
Test Values: 1 2 2
1.6666666666666667
```

Exercise 2 (Evaluation: Text)

Write two static methods to determine the area $A(s) = s^2$ of a square and the volume $V(s) = A(s) \cdot s$ of a cube with side length s:

```
double squareArea(double side)
double cubeVolume(double side)
```

During implementation, the function for the volume should call the function for the area.

Then write a main method, which in turn calls cubeVolume appropriately. Your program should read two double numbers a, b with sc.nextDouble(), where you can assume that $0 \leq a < b$ applies. Then a small table of 11 values each for area and volume of the cube shall be output, where the page length s results from $s = a + n \cdot \frac{b-a}{10}$ for $n = 0, 1, 2, ..., 9, 10$.

To output the double numbers, use the System.out.format method. For example, System.out.format("%8.2f", x) outputs a double number x with 8 characters width with 2 decimal places. This allows you to reproduce exactly tables of the following form:

```
Range: 0 5

|   Side   |   Area    |  Volume   |
|---------|----------|----------|
|     0.00 |      0.00 |      0.00 |
|     0.50 |      0.25 |      0.13 |
|     1.00 |      1.00 |      1.00 |
|     1.50 |      2.25 |      3.38 |
|     2.00 |      4.00 |      8.00 |
|     2.50 |      6.25 |     15.63 |
|     3.00 |      9.00 |     27.00 |
|     3.50 |     12.25 |     42.88 |
|     4.00 |     16.00 |     64.00 |
|     4.50 |     20.25 |     91.13 |
|     5.00 |     25.00 |    125.00 |
```

Exercise 3 (Evaluation: Numbers)

Normally a year is a leap year when the year number is dividable by 4 without remainder. This does not apply if it is dividable by 100 without a remainder. However, if the number is dividable by 400 even without remainder, it is still a leap year.

Implement to methods:

- `isLeapYear` checks if the input parameter is a leap year, corresponding to the previous mentioned rules.
- `previousLeapYear` returns the leap year that occurred before the entered year.

`previousLeapYear` should call the method `isLeapYear` (possibly several times). Decide for each method which data to pass and what the return type is.

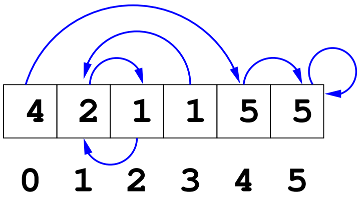
Write a `main` function that allows corresponding tests of `previousLeapYear`, suitably for the following example:

```
Year:  2018
Previous Leap Year: 2016
```

Exercise 4 (Evaluation: Numbers)

Consider a field `array` analogous to the scavenger hunt examples (i.e., it's about repeatedly setting `i=field[i]`). However, all entries in the field are certainly so small that they can be used as an index for the field again. Every 'scavenger hunt' must therefore run in a loop at some point:

With the values `4 2 1 1 5 5`, for example, the following transitions occur in the array and thus obviously also two loops (loop 1 is 5-5-5-..., loop 2 is 1-2-1-2-...):



Now determine how many loops there are in an input field (including the indexes to the indexes leading to the loops).

In the example, the indices 0,4,5 lead into loop 1, the indices 1,2,3 lead into loop 2:

```
Size:  6
Content: 4 2 1 1 5 5
Number of Loops:  2
Assignment of Indexes: 1 2 2 2 1 1
```

Advice: Use a second field `marked` of type `int`, in which you enter for each index the loop in which it is located, if this is already known. So meaningful initial values for the `marked` field are `0`.

You can now proceed as follows:

1. At the beginning of the algorithm no loop is known, so initialize a counter k with the value 0.

2. Then look (repeatedly, see below) at the smallest index `i` that has not yet been assigned to a loop, where `marked[i]` still has the value `0`.

   (At the beginning there is always `i=0`*. This procedure also results in a unique numbering of the loops as "loop 1", "loop 2", etc.).*

3. Then check (e.g. as in task 4 of Assignment 4) whether you reach an already known loop from `i` (i.e. an index `j` is reached from `i`, where `marked[j]` has a value >`0`).

4. If a known loop is reached, save the value found in the loop `marked[j]` in a variable `s`.

5. But if no known loop is reached, you have found a new loop; so increment `k` and also set `s` to the new value of `k`.

6. Now set the `marked` field to the value `s` for `i` and for all indexes that can be reached from `i`.

7. As long as the `marked` field still contains `0` values, repeat this procedure from step 2.

8. At the end, output the value of k and the values of the `marked` field.

It is best to first execute the algorithm "by hand" on some of the evaluation examples before implementing it!

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 07

Submission of solutions until 3:00 p.m. at 22.12.2021
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 (Evaluation: Predefined **`main`** Method)

This task deals with two-dimensional (**`double`**) arrays and how they can be utilised as matrices. Their task is to implement methods that make it possible to multiply two matrices together. In case you have lost track of how matrix multiplication is performed, you can find guides under following links[1][2].

For this task you have to implement the following methods:

```
boolean isValid(double[][] matrix1, double[][] matrix2)
double[][] mulMatrices(double[][] matrix1, double[][] matrix2)
```

The method `isValid()` is used to determine if a matrix multiplication is possible. Remember, that you can only multyply two matrices if the number of collumns of `matrix1` is equal to the number of rows of `matrix2`.

The method `mulMatrices()` is used to implement the actual matrix multiplication. It returns a two-dimensional **`double`** array, containing the result of the multiplication. In case that both matrices cannot be multiplied because `isValid()` returns **`false`**, simply return the **`null`** reference

A suitable `main` method for testing your two implementations is already predefined there.

[1] `https://www.mathsisfun.com/algebra/matrix-multiplying.html`
[2] `https://en.wikipedia.org/wiki/Matrix_multiplication`

Exercise 2 (Evaluation: predefined `main` method)

Consider the recursively defined sequence $(f_n)_{n=0,1,\dots}$ of integers f_n:

$$\begin{aligned} & f_0 := 0 \\ & f_1 := 2 \\ & f_2 := 4 \\ \text{and in case } n \geq 3\text{:} \quad & f_n := (f_{n-1} + f_{n-2}) \cdot f_{n-3} \end{aligned}$$

1. Implement a "**static double** frec(**int** n)" method to calculate the n" th value f_n of this sequence in the form of a *recursive algorithm*.
2. Implement a "**static double** fdyn(**int** n)" method to calculate f_n using *dynamic programming*[3].

You should write both partial solutions into the given file. A suitable `main` method for testing your two implementations is already predefined there.

Exercise 3 (Evaluation: predefined `main` method)

Familiarize yourself with the methods of the `String` class[4]:

1. Create a method

   static int countOccurrence(String haystack, String needle)

   that counts at how many positions the string `needle` occurs as a substring in `haystack` For example, the call `countOccurrence("abbbba", "bb")` should return the value 3.

2. Create another method

   static String extractTag(String s)

   with the following property: For an argument `s` of the form

   prefix '`[`' *infix* '`]`' *suffix*

   the function `extractTag` should return the substring *Infix*. You can assume that '`[`' and '`]`' both occur exactly once in the correct order in `s`.

   At `extractTag("1234[5678]9")` the string `5678` should be found and returned.

[3] See the slides of chapter 3 part 4

[4] http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Exercise 4 (PCP - Post correspondence problem)

Create a method

```
static int pcpTest(String[] x, String[] y, int[] t)
```

with two `String` arrays (`x` and `y`) of the same size and one **`int`** array (usually of a different size) as parameter. It is to be tested whether `t` is a solution to the post correspondence problem (PCP) [5] of `x` and `y` is.

This test is to proceed as follows:

- First, the method should test the validity of the parameters: It should return the value `-1` if one of the following cases occurs:
  - if `x`, `y` or `t` is a **`null`** reference,
  - if the arrays `x` and `y` have different numbers of elements,
  - if one of the arrays `x` or `y` contains a **`null`** reference instead of a string,
  - if one of the strings in one of the arrays `x` or `y` is the empty string `""`,
  - if the array `t` has the length 0,
  - if one of the `t.length`-many values in the field `t` is negative or $\geq$ `x.length` is.

  These cases are invalid parameters for the post correspondence problem.

- If the parameters are valid, the following must be tested: If the string `testX` resulting from the concatenation of the strings

  `x[t[0]]` + `x[t[1]]` + ... + `x[t[t.length-1]]`

  matches the string `testY` from the concatenation of

  `y[t[0]]` + `y[t[1]]` + ... + `y[t[t.length-1]]`

  If yes, `1` should be returned; if no, `0` should be returned.

Example: With `x={"aba", "ba", "b"}` and `y={"a", "ba", "bab"}` there is a match for `t={0,1,1,2}`:

`t`:	0		1		1		2		
`testX` =	aba	+	ba	+	ba	+	b	=	abababab
`testY` =	a	+	ba	+	ba	+	bab	=	abababab

[5] see `https://en.wikipedia.org/wiki/Post_correspondence_problem`

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 08

Submission of solutions until 3:00 p.m. at 05.01.2022
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!

Exercise 1 (Book Class, Evaluation: predefined `main`)

Create a class `Book` with the following structure:

- Every book has as instance variables `author`, `title` (both as `String`), a `year of publication`, a `page number` (both as **`int`**) and of course a `content` (again as `String`).
- Write a constructor that sets the first four values at once (in the above order). In addition, the `content` is preset with the empty string.
- The content can then be written with a method `append(String str)` and extended later.
- The class also contains a class variable `String publishing` that can be set with a static `setPublishing` method.
- All variables are supposed to be **`private`**
- Write a `toString` method that wraps all of these values except the content in a string following format:

```
The book "xx" with "xx" pages by the author "xx"has been
published by the publishing company "xx"in the year xx.
```

- An object method `string quote( int start,int length)` should return exactly `length` characters, starting at position `start`. If the book has fewer characters than necessary, the return value should be correspondingly shorter.

A suitable test class with a `main` method is specified for the evaluation.

Exercise 2 (Dice and Mia, Evaluation: predefined `main`)

First implement a class `Dice` with the following methods analogous to the coin toss (example `K5B01E_CoinToss` from the lecture):

- A method `throwDice()` randomly assigns a new number (equally distributed) between 1 and 6 to the dice.
- A method `pips()` should return the value thrown by the dice as `int`.

  The function `pips()` of a finished `Dice` object must return **always** a number between 1 and 6!
- Both `throw()` and `pips()` should be accessible for users.
- In addition implement a function in `Dice`

  `public static int pipSum(Dice d1, Dice d2),`

  that returns the total value of both dice.

Write another class `Mia` that internally stores two `Dice` objects. These should be created as follows in the constructor of `Mia`

```
        public class Mia {
                private Dice d1, d2;
                public Mia() {
                        d1 = new Dice();
                        d2 = new Dice();
                }
        }
```

The class should also provide the following functions:

- `public int valueAndThrowDice()` interprets the current numbers of the two dice as numbers according to Mia rules[1]. In addition, the function should throw both dice again, and then return the interpreted value *of the old throw*.

[1] See `https://en.wikipedia.org/wiki/Mia_(game)`

- **public static boolean** isMia(**int** value) checks whether value is the maximum "‘Mia"’ result.
- **public static boolean** isDouble(**int** value) tests value to see if it’s a doublet.
- **public static boolean** isLess(**int** a, **int** b) compares the two values a and b according to Mia rules and returns **true** exactly when a represents a smaller value than b in this sense.

  Hint: Implement this method by using isMia(**int** value) and isDouble(**int** value)

Watch out: The evaluation only shows you whether you have completed all subtasks; it does not help you to be correct!

Exercise 3 (Evaluation: predefined main)

Write a class UserManagement to simulate the management of user names in a computer, in particular password management.

The class shall contain two *private* arrays id and pw of 100 strings each to store the data.

The following methods are to be implemented:

- A constructor UserManagement()
  Here the fields id and pw are associated. Pay attention to what initials values the individual components of id[i] are set!
- **boolean** newUser(String name, String password)
  This can be used to create a new user with the specified pair of name and initial password.

  If the user already exists, no change should be made in the data, but the value false should be returned. The same applies to the case that more than 100 users should exist at the same time together with the new user or that the username consists of less than four characters. Otherwise the user will be saved accordingly and true will be returned. The password may be any non-empty string here.
- **boolean** checkPassword(String name, String password)
  This checks if the specified password matches the user name.
- **boolean** changePassword(String name,
  String oldPw, String newPw)
  This can be used to change the user’s name password if oldPw was correct and the new password newPw has a length of at least eight characters.

  If the change is successful, the value true is returned, otherwise false.

- There is an administrator password that applies to all objects of this class. This password can be set once by **`void`** `setAdmin( String adminPw)`, again requiring at least eight characters.

  Any attempt to change a previously set administrator password should be logged with a warning `"unauthorized access "` on the console.

  Attempts to set an unset administrator password to a password that is too short, on the other hand, are completely ignored.

- The administrator password can be used in objects of the class `UserManagement`. Passwords can be set to any non-empty values, using the following method:

  ```
  void setPassword( String adminPw,
                                  String name, String newPw)
  ```

You have to think for yourself where to put the modifiers `public`, `private` or `static` (matching the given `test` class). For the sake of simplicity, deletion of users is not foreseen.

Attention: An associated class `test` with some calls to test the user management is given. However, this is supposed to be replaced by another class when correcting your solution, which among other things will try to crash your solution (which you should avoid...)

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 09

Submission of solutions until 3:00 p.m. at 12.01.2022
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

Exercise 1 (Evaluation: predefined `main` method)

See the class `Queue` from example `K5B05E_Queue_Linked` of the lecture. Extend `Queue` with the following functionalities:

- `public void append(int[] newData)`
  This function appends all values from the `newData` field to the queue.
- `public int size()`
  This function is to output how many elements are currently in the queue.
- `public int elementAt(int position)`
  This function is to output the value at the position `position` of the queue (it is to count as in a field starting from index 0). If `position` is negative or the queue contains too few elements, -1 is returned.
- `public int contains(int m)`
  This function is to determine whether the value `m` is contained in the queue. If it is included, its position in the queue is returned, if not -1.
- `public static Queue concat(Queue q, Queue p)`
  This static function should concatenate the two queues, so that first the elements from `q` and then the elements from `p` are contained in the resulting queue. Both `q` and `p` should remain unchanged.

- **public int**[] toArray()
  This function is intended to create an array containing all entries of the queue. The queue should then be empty.

QueueTest may be changed for your own tests, but for the correction a new version of QueueTest will be used, which will make more extensive tests. Therefore your program must work with the original version of QueueTest as well!

Exercise 2 (Evaluation: predefined main method)

Take another look at the class Queue, but now from the example K5B05E_Queue_Array of the lecture. Extend Queue with the same functionalities as in the previous task:

```
public void append(int[] newData)
public int size()
public int elementAt(int position)
public int contains(int m)
public static Queue concat(Queue q, Queue p)
public int[] toArray()
```

However, you should also change the implementation of the queues so that queues can never become full when elements are added:

- In the constructor, the value 4 is to be used as the initial size of the field used in the queue.
- A dynamically growing field should be used for storage (similar to StringBuffer): If the field of a queue is full (usually by enQueue), a new field of double size is created, into which all data is transferred.
- Similarly, if the queue is shrinking, the field is to be replaced by a new field with half the capacity: If an element is removed and the number of elements in the queue is then less than or equal to a quarter of the current capacity, the size of the field in the queue is to be halved. However, the size should not be less than value 4.
- In order to trigger these capacity changes manually, two methods

  ```
  public void setCapacity(int n)
  public int getCapacity()
  ```

  are to be implemented which can be used to manually adjust or query the current capacity. A call setCapacity(n) should be ignored if n< 4 or n≤size() is valid.

The note from the previous task for the files Queue.java and QueueTest.java also applies here.

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 10

Submission of solutions until 3:00 p.m. at 19.01.2022
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

Exercise 1 Technical Dictionary (without evaluation; `main` for tests)

Write a class `TechnicalDictionary` to assist in learning technical terms with a computer.

A class `Entry`, which should store the following data, is helpful for this;

- `String technicalTerm` denotes the term that should be learned,
- `String explanation` for a brief explanation,
- **`double`** `learned` for an evaluation of how well the word was learned.

External access to these components should only be allowed with getter/setter methods, furthermore you should write a `toString` method.

The constructor for `Entry` should have the two strings as parameters and set them accordingly, where the variable `learned` should be initialized with 0.

The class `TechnicalDictionary` should then have the following characteristics:

- A `TechnicalDictionary` must contain an array with 1000 references of the type `Entry`.
- This array is to be created in the constructor (containing 1000 null references). The scanner used for the input should be passed as the only parameter in the constructor and stored in an (already given) instance variable so that the method `exercise()` given below has access to it.

- With a method `enter(String technicalTerm, String explanation)` a new entry should be added to the lexicon (and replace a null reference in the field).
- Another method `exercise()` should now search for the entry with the least knowledge in the lexicon and output the technical word. Then the method should first ask the user if he knows the explanation of the entry.

  The user should then be able to indicate with an empty entry (i.e. Return-Key only) that he wants to see the explanation of the technical term.

  The user can then specify whether he actually knew the explanation. An empty input signals 'yes', in this case the value of the variable `learned` should be increased by the value 1. A non-empty input means 'no', then the value of the variable `learned` is halved.
- The `toString` method should output the entire dictionary content including the knowledge values (but of course without the null references), one entry per line.

The predefined `main` method creates a technical dictionary with 3 entries, lets the user practice ten times and then outputs the technical dictionary so that the learning success can be controlled.

There is again a hidden sample solution that shows you how the program should run. To create your own solution, you must exchange two selected lines in the source code of the class `TechnicalDictionary`.

Exercise 2 Cookbook (Evaluation: predefined `main` method)

The default is an abstract class named `Recipe` containing four methods: `recipeName()`, `ingedientsList()`, `equipment()` and `preparationTime()` (in minutes). It also contains a (static) `main` method for a simple test of the solution.

Get out of `Recipe` three classes `Pizza`, `Sandwich` and `Risotto` The `recipeName` and special ingredients should be placed in the constructors. General ingredients, the required cooking utensils and the preparation time result from the concrete class:

- `Pizza`:
  General Ingredients: Yeast Dough, Tomatoes, Mozzarella, Oregano
  Equipment: Oven, Pizza Plate, Rolling Pin
  Preparation Time: 40
- `Butterbrot`:
  General Ingredients: Butter
  Equipment: Knife, Cutting Board
  Preparation Time: 3
- `Risotto`:
  General Ingredients: Rice, Vegetable Bouillon, Parmesan

Equipment: Pot, Stirring Spoon
Preparation Time: 40

The `ingedientsList` should consist of the general ingredients for the basic recipe and the special ingredients from the constructor.

The default `main` method should therefore lead to the following output:

```
Name:         Pizza Salami
Ingredients: Salami, Yeast Dough, Tomatoes, Mozzarella, Oregano
Equipment:   Oven, Pizza Plate, Rolling Pin
Duration:    40.0
Name:         Pizza Diavolo
Ingredients: Salami, Peperoni, Yeast Dough, Tomatoes, Mozzarella,
             Oregano
Equipment:   Oven, Pizza Plate, Rolling Pin
Duration:    40.0
Name:         Ham Sandwich
Ingredients: Brown Bread, Ham, Butter
Equipment:   Knife, Cutting Board
Duration:    3.0
Name:         Toast Hawaii
Ingredients: Pineapple, Cheese, Butter
Equipment:   Knife, Cutting Board
Duration:    3.0
Name:         Pumpkin Risotto
Ingredients: Pumpkin, Rice, Vegetable Bouillon, Parmesan
Equipment:   Pot, Stirring Spoon
Duration:    40.0
```

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 11

Submission of solutions until 3:00 p.m. at 26.01.2022
at moodle.uni-trier.de

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

Exercise 1 (No Evaluation: Test class is provided)

A class `UniMember` is given, which can register university members in general. All other classes should be derived from this class. Another given class `Test` shall be used by you and extended for testing purposes to test all(!) functions and classes implemented by you.

Implement the following model:

1. (4 points) A class `Student` derived from `UniMember`:
   - In addition to the data from `UniMember`, instances of type Student also have a component `matriculationNumber`, which is filled with a continuous and unique **`int`** value for each student.
   - A student is generated by a constructor that expects the same parameters as the constructor for instances of type `UniMember`. The matriculation number is assigned automatically (via an instance counter analogous to`count` in the example `K5B02E_Rectangle`).
2. (4 points) A class `Staff` also derived from `UniMember`:
   - Instances of type `Staff` have additional components `room` and `phoneNumber`.
   - An employee is generated by a constructor that expects the same parameters as the constructor for instances of type `UniMember` and additionally the information `phoneNumber` and `room`. The values are inserted into the corresponding components.

3. (4 points) A class `Professor` derived from `Staff`:
   - Instances of the type `Professor` additionally have an array `assistants`, in which assistants are stored. A professor can have a maximum of 10 employees.
   - The constructor for professors looks like the constructor `Staff` and fills the corresponding components.
4. (8 points) Generate a class `Assistant` that derives from `Staff`:
   - Instances of type `Assistant` also have a component `supervisor` of type Professor.
   - The constructor of the class `Assistant` contains data for all necessary components and additionally sets the superior of the assistant, so it looks like this:

     ```
     public Assistant(String name,..., Professor supervisor)
     ```

     All necessary components of the instance are filled; in addition, the just generated assistant is entered into the array of the assistant of the professor, who is his boss. If the professor already has 10 assistants, an error message is displayed and the program is terminated (with `System.exit(0)`).
   - Implement a method with the signature

     ```
     public void resign()
     ```

     where a assistant quits its job, i.e. is removed from the assistants array of its boss. In addition, he should not have his own boss anymore.
5. (4 points) Implement a consecutive, unique personnel number `staffNo` starting at 1000 for all employees.
6. (6 points) Implement an additional method with the signature

   ```
   public boolean employed()
   ```

   that returns whether an instance of any class from the model is an employee of the university or not. All classes of the `Staff` class count as busy, with the exception of assistants after termination.
7. (4 points) Implement a method with the signature

   ```
   public String toString()
   ```

   in each class that returns all components of that instance.

   For professors, the method should output a list of his assistants and their data. In addition to the first name, surname, address, telephone number, room and employee number, the first name and surname of the boss, if he has one, must also be output when outputting the assistants.
8. (6 points) Test the methods and classes you have implemented in the given class `Test` with static inputs. Use (implicitly) the function `toString()` for the output. To test `employed()` and `toString()`, you should cache instances of the classes you have defined in variables of the class `UniMember`.

Watch out: Pay attention to reasonable comments in your `.java` files!

Exercise 2 (Evaluation: predefined `main` method)

The given queue `q` in the main class can store object of three different classes (`String`, `Integer` and `Boolean`). These objects are stored in an unknown order.

Implement a method `evaluateThreeClasses` that processes each element of the queue individually and outputs the following information on the terminal:

- First count how long all `String` objects in the queue are and output only the size.
- Compute the sum of all integer values in the queue and output the sum only
- The last step is count how many **`boolean`** value were set to **`true`**.

Note: After a `x = q.deQueue()` you must first determine to which of the three classes `x` actually belongs.

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 12

Submission of solutions until 3:00 p.m. at 02.02.2022
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

Exercise 1 (Evaluation: predefined `main` method)

a) Write an abstract class `Pet` that stores the weight and name of a pet. The class should not have a getter and setter method, but its variables should be declared as **`public`**. Additionally, the following methods should be provided:

- **`void`** `feed(`**`double`** `g)`: The weight of the pet is increased by `g`
- **`void`** `dropWeight(`**`double`** `g)`: The weight of the pet is reduced by `g` due to the daily basal metabolic rate.
- `String toString()`: To be implemented as an abstract method.
- `String makeNoise`: An abstract method that returns a typical sound for the respective animal. See down below for more details about the noises.
- **`int`** `compareTo(Pet pet)`: Is to be implemented as an abstract method to compare two pets by weight. The method shall return a -1 if the weight of **`this`** is less than that of `pet`, 0 if both objects have the same weight and otherwise a 1.

b) Write two more classes `Cat` and `Dog` which extend the class `Pet`

- The name of the animal and a "Meow" or "Woof". Example:

```
Garfield: Meow
Lucky: Woof
```

- The class `Dog` shall include a method for walking a dog. The method has the following signature

  ```
  public void walkTheDog()
  ```

  and has two tasks: (1) the dog should lose 0.2 kg of weight while walking and (2) the method should provide an output of the following form:

  ! Lucky goes for a walk and loses weight!

- Familiarize yourself with exception handling by deriving the two classes

  ```
  NotComparableException
  TooHeavyException
  ```

  from the `Exception` class already contained in Java.

  - The `TooHeavyException` is designed to prevent dogs weighing more than 15 kilos from being walked. In this example, they want to eat and sleep all day long! The `TooHeavyException` class has a constructor of the following form:

    ```
    TooHeavyException(double weight)
    ```

    When "throwing" this exception, the overweight of the dog should be calculated and stored in a variable. The class also provides a method, denoted as `String getErrMsg()`, which returns a string of the following form:

    ```
    Exception:  Dogs with overweight don’t go
    for walks
    ```

    Extend the method `walkTheDog` so that too heavy dogs "throw" an exception.

  - The `NotComparableException` is intended to prevent that animals of different species are compared with each other (e.g. dog with cat). To do this, implement this class so that it has a constructor of the following form:

    ```
    NotComparableException(Pet pet)
    ```

    When "throwing" this exception a string is to be stored in a variable. This string depends on whether dogs are compared with cats or other different animal species. The string to be stored has accordingly one of the following forms:

    ```
    Exception: You cannot compare cats and dogs
    Exception: No comparison possible
    ```

    The class provides, like the exception before, a method with the signature `String getErrMsg()` This method returns the previously saved string.

    Now extend the methods `compareTo(Pet pet)` of the classes `Cat` and `Dog` so that an exception is "thrown" if cats and dogs are compared.

- The method `toString()` shall be overwritten so that a string of the following form is returned

```
Cat: Garfield weighs 4.2 kg
Dog: Lucky weighs 9.8 kg
```

Exercise 2 (Evaluation: predefined `main` method)

Model different road users in a hierarchical structure. To do this, implement the following model:

a) An abstract class `Traffic`, from which all other classes are derived:

- Road users have several values like `name` (string), `wheels` (int), `drivenKilometers` (double) and `maxSpeed` (int).
- Write a constructor that receives the expected parameters and writes them into the variables provided.
- A vehicle of class `Traffic` also has an abstract method with the signature

  ```
  abstract boolean licenseNeeded()
  ```

  This method should return whether a driving licence is required depending on the road user (car, motorcycle or bicycle).
- Implement a method with the signature

  ```
  public void addKilometer(int km).
  ```

  This method should be used to increase the number of kilometres travelled.
- Implement another method with the signature `String toString()`.
  This should return a string of the following form:

  ```
  name, drivenKilometers, maxSpeed, wheels
  ```

b) Derive from the class `Traffic` the classes `Car`, `Motorbike` and `Bicycle`

- A **bicycle** has only two wheels and a maximum speed of 30 km/h. Furthermore, a bicycle can be used by a courier. Therefore this class should store a value `carrier` (boolean). Design the constructors of the class so that you do not need to specify a number of tires or a maximum speed.
- Like a bicycle, a **motorbike** has only two wheels, but can travel at any speed (depending on the model, etc.). When implementing the constructor, note that here too, no information about the number of wheels is necessary.
- A **car** has four wheels and can have any maximum speed like a motorcycle.

c) Derive from the class `Car` a class `Oldtimer`. This class stores a value `year` (**`int`**) with the year of manufacture of the car. Design an appropriate constructor.

d) Extend all classes derived from `Traffic` (also oldtimer) with a method denoted as `String toString()` This method should call the `toString()` method of the superclass, but first specify the type of the car. This results in a string of the following form:

```
Car:  name, km, maxSpeed, wheels
```

An oldtimer should call the constructor of the superclass (`Car`) and add the following addition:

```
Car:  name, km, maxSpeed, wheels (Oldtimer)
```

e) Implement two interfaces `Collectible` and `ProtectiveClothing`

- Since vehicles can also always be collectibles, the classes `Car`, `Motorbike` and `Oldtimer` should have the interface `Collectible` The interface has only one method with the signature **`int`** `calcValue()` The value of a collector's item is, for the sake of simplicity, calculated by multiplying the driven kilometers by the number of tires.
- The second interface should contain a method with the signature

  ```
  boolean needsProtectiveClothing()
  ```

  All classes derived from `Traffic` should have this interface and implement the method `needsProtectiveClothing().` Cars and oldtimers do not need special protective clothing.

Attention: The tasks are intentionally underspecified. Find meaningful and suitable solutions yourself and familiarize yourself with the `main` method!

Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 21/22

Exercises for the Class

Elements of Computer Science: Programming

Assignment 13

Submission of solutions until 3:00 p.m. at 09.02.2022
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

Exercise 1 Reading and processing files

Implement a class `Product`:

1. Read the given file `Productlist.txt` in a static method of your solution (e.g., but not necessarily, in the `main` method). The imported data contains information about products using key-value pairs. Each line identifies its own product. The attributes of a product are `artno`, `name` and `price` The order of the attributes in the file is not fixed, also a `artno` of a product can occur several times.

   The instances of the class `Product` should store the data of a single product in three corresponding instance variables (i.e. in two `String` (`artno`, `name`) and one **`int`** (`price`) variables). Write a constructor for `Product` that appropriately splits a read line (i.e. a string) into these three parts. `artno` and `name` should be stored without the enclosing quotation marks. `StringTokenizers` are not suitable for the decomposition, because the name of a product may contain many different characters (including ',' and ':') (but for simplification it does not contain any further quotation marks).

   Save the read data in a `ArrayList<Product>` file.

2. Write the data into a file `ModProductlist.txt` in a form comparable to the original file, but without additional spaces (except in names and article numbers) and in the fixed order `artno`, `name` and `price` for each product.

3. Write a corresponding file `CheapProducts.txt` with all products that cost less than 20 (Euros).

4. Write a corresponding file `CheapProducts.txt` with all products that cost less than 20 (euros). Calculate the average price of all products and enter it (rounded down to an integer) in the form ‘`Average Price: ... Euro`’ on the console. You can also find the lowest and the highest price and additionally display

   ‘`Price Range: between ... Euro and ... Euro`’

Watch out: Unfortunately, you cannot view the output files (.txt) created with your program. This means that you can only test whether they are correct for reading/writing using the print function. Therefore it is better to compile the tasks on your local system and test it there extensively. The only important thing is that your Java files and the text files are in the same folder.