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.