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

Wintersemester 2024/2025

Exercises for the Class

# Elements of Computer Science: Programming

## Assignment 13 (Bonus)

Submission of solutions until **Monday, 10.02.2025 at 14:00**
at `moodle.uni-trier.de`

This is a bonus assignment. Please pay attention the changed deadline on **Monday**!
It will not increase the total number of assignments required for exam qualification.
If you still need Home Assignments points, the bonus points will of course count for you.

The assignment will be discussed in the final tutorial on 10.02.2025.

### Exercise 1 (Evaluation: Text)

This task deals with the creation, use and interaction of *packages* in Java:

1. Define a package `optimizers`. In it, create an interface `Optimizer` with one method

   ```
   List<String> optimize()
   ```

   using the `java.util.List` interface.

2. Define a package `optimizers.address`. In it, implement the following two classes:

   - `AddressUtil` must not be extensible or instantiable. It should be only available in its own package. Implement one static method

     ```
     String[] splitAddress(AddressOptimizer ao)
     ```

     which splits a string containing postal code and city (e. g., "54296 Trier" or "02999 Knappensee-Koblenz Gem. Lohsa") into its two parts using *regular expressions* both for detecting a correctly formatted address string and for extracting the two address parts from the string. If the string has an invalid format, return **`null`**.

     A valid postal code always has five digits (leading zeros are permitted). A city name may consist of upper- and lower-case letters ('a' to 'z' and 'A' to 'Z') or space characters ' ' or commas ',' or hyphens '-' or dots '.' or round brackets '(' and ')'.

   - `AddressOptimizer` should implement the `Optimizer` interface and have a constructor which gets passed one `String` argument. This string should be stored in a field only accessible from members of the same package and must not be modifiable after its initial assignment.

     The `optimize()` method should return the output of the `splitAddress()` method by taking into consideration the involved data types. This method always has to return a `List` and must never return **`null`**.

3. Define a package `de.uni_trier.dbis.eocs.assignment13`. In it, implement the class `MainExecutable` as follows:
   - The class should contain an "appropriate" `main(...)` method.
   - It should open a text file called `addresses.txt` (from the current directory) and read it in line by line. Each line represents one address.
   - For each address read, run the `optimize()` method (from `AddressOptimizer`). If its result is empty, output "`empty result`". Otherwise, print out the result, each element on a new line.

**Note**: For this task, no placeholder files have been created on Moodle. You have to use the "*New*" and "*Rename*" commands from the Moodle toolbar in order to create files with the correct paths and names. You will need to create a total of four "`.java`" files.

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

The class `Person` is provided and stores the name (`String`) and nationality (`String`) of a person. All instance fields are declared **`public`**, so there is no need for getter methods.

The provided class `Evaluation` contains the `main` method which first reads in the total number of persons (**`int`**) and then creates `Person` instances with the read in names and nationalities. These objects are then stored in a `List<Person>`.

Implement the following three static methods:

1. `Map<String, Integer> countNationalities(...)`
   The method `countNationalities()` has one input argument:

   ```
   List<Person> peopleList
   ```

   For each nationality in the list, the method should determine how many `Person` objects with this nationality exist. The result should be stored in a `Map<String, Integer>` using the nationality as *key* and the count as *value*.

2. **`int`** `getNumberOfPeople(...)`
   The method `getNumberOfPeople()` has two input arguments:

   ```
   Map<String, Integer> map
   String nationality
   ```

   For the given `nationality`, this method should return the number of people with the given nationality in the provided `map`. This map contains the result of a previous invocation of the `countNationalities()` method.

3. `List<Person> getPeople(...)`
   The method `getPeople()` has two input arguments:

   ```
   List<Person> personList
   String nationality
   ```

   This method should store all `Person` references from the given list in a new list (and return it), but only if their `nationality` matches the nationality given as argument.