Lab Session Week 12

Mock Exam

This session consists of a **Java implementation assignment** and three extra exercises on designing **UML diagrams**. The final exam will consist of an implementation assignment similar to this one (70% of the score) and **one** modelling exercise (30%) of the score.

# Practice Exam OOS: Programming Part

**This part of the practice exam covers Java programming. It is meant to be solved in 3 hours. You can use a 4-page summary and the Java documentation. The programming part counts for 70% of your total score on the exam.**

## Problem description

A camping site wants to automate the management of its reservations. Currently the camping provides pitches (empty places where visitors can place their own tent or caravan) and fully equipped tents. However, they want the system to be easily extendable to other types like bungalows. Every available place on the camping has a code (consisting of a character and a number) and a capacity (maximum number of people that can use this place). For pitches, they want to store whether or not electricity is available. For tents, a list of possible extras that can be added to the tent (baby bed, microwave…) needs to be stored (each extra is a String). Also for tents, the year of their construction is stored.

The camping is open from May 1st until September 30th (included). Every place keeps the information about which day it is already occupied. People can only make a reservation for the current season. It should be possible to check if a place is free for a certain period, to make a reservation and to cancel a reservation. Periods are defined by a start day and an end day (=day you leave the camping site, so the place stays free), where days are numbered as integers: May 1st = 0, May 2nd = 1 and so on till September 30th = 152. They also want to know how many days a place has been occupied. The base price of a pitch is €12/day, for tents €50/day. However, for tents you get 1 day free for every week you stay. It should be possible to calculate the price for a given period.

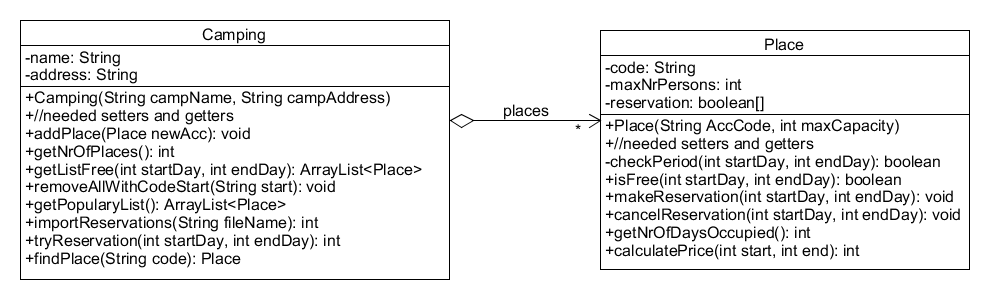
A camping stores its name and address. Of course, you should be able to add places (of whatever type) and to know the total number of places. Places with the same code cannot be added twice. To facilitate reservations, it should be possible to get a list of all free places during a certain period. There should also be the possibility to check if a place with a given code is available on the camping.

From time to time the organization of the camping site may change, so they want to have the possibility to remove all places where the code starts with a given combination. They are also interested in the popularity of the places, so they want a list of all places ordered by number of days occupied with least occupied first.

For regular customers, the camping allows to make reservations for the next year. This information is stored in a text file. So at the start of every new season, this file is used to import existing reservations. This method returns the number of imported reservations. If something goes wrong while importing it returns the negative value of the number of successful imports.

## Implementation

Start from the project ‘campingstart’ form Toledo, and from the following class diagram. ***You may NOT modify the signature of the provided methods nor the type of the fields. It is possible that you have to add extra attributes or methods to meet all the demands.***

******

As a start, no distinction is made between pitches and tents. Some more info about the tests.

* Test3 checks if you can make and cancel a reservation and checks if you can calculate the number of occupied days with getNrOfDaysOcuupied(). For every method in which you need to make a reservation, you have to make sure that the given period is valid. Implement the needed checks in a separate method checkPeriod() and call it where needed. Don’t forget to check if a place is still free (method isFree()) if you want to make a new reservation. If something goes wrong when making or cancelling reservations, print a message to the screen.
* The format of the import file is as follows  
   - code of place  
   - start day  
   - end day

This structure repeats any number of times. The method returns the number of imported reservation (= number of periods). If the method finds invalid information it returns the number of imports that has already been done as a negative number and skips the import.

* In test10 we get a list of all places ordered by popularity, with least popular places first. The order of places with the same popularity is unspecified.
* The final challenge is to implement the method int tryReservation(int startDay, int endDay). With this method it should be possible to find a solution, even if there is not one single place free for the whole period. It will look for a solution using multiple places. You may start from the first day and look for the place with the highest availability. Then you repeat this for the day following this period and so on till you made a reservation the whole period. The method returns the number of places needed. If really no solution is found, the method returns -1.
* Keep in mind if a certain test does not pass that the reason may be twofold: you did something wrong in your syntax (like = instead of ==) resulting in different behavior or the logic you implemented is not correct or complete. Use available tools to check where the problem occurs.

# Practice Exam OOS: UML Exercises

**The following exercises serve as extra practice in translating a problem description into a class diagram. You will get one similar task on the exam for 30% of your total grade. Example solutions to these exercises will be published on Toledo.**

## Album

**Create a detailed class diagram starting from following problem description: define the needed classes, their attributes, the signature of their methods and the relations between the different classes.**

Since you are the programming expert of the family, they ask you to create an application which is capable of creating interactive albums of their holiday visuals (currently only photos and movies) with some extra features.

All visuals have a name, a size (integer amount of bytes) and a date on which they are taken in the format “dd/mm/yyyy”. There is a possibility to add a rating (an integer number higher than zero and maximum 10) and to add maximum 3 tags to each visual. It needs to be possible to check if a visual contains a certain tag. For movies, you should also store their play time in seconds. Movies can be played and paused, photos can only be shown, but it should be possible to show a photo with or without a fade-in effect.

An album has a name and you can add an unlimited number of photos and movies to it. Take care that an album cannot contain 2 visuals with same name and same date. Create the option to search for the highest rating available in the album. You should also be able to select all visuals with a certain tag, by generating a collection of all visuals which contain this tag. To limit the size of an album, you need a way to simply remove all visuals with a rating lower than a certain value. Keep in mind that some visuals may not have a rating yet (their rating value is still 0) and those may never be removed. Provide a way to check how many visuals have been deleted. You also need the functionality to show a slideshow where all visuals with at least a certain rating are played or shown in a random order. The order should be different every time you call this method. The method returns the names of the visuals in the order they were played.

Another requirement is the option import visuals from a text file into an album. Assume the file contains all necessary info. Provide a check to see how many visuals have been read from file.

A last requirement is to search an album for all visuals with a size smaller than a given limit. Here the total size of all selected visuals is returned.

## Festival

**Create a detailed class diagram starting from following problem description: define the needed classes, their attributes, the signature of their methods and the relations between the different classes.**

To create an optimal experience for all visitors of a festival, the organizers want to experiment with a system where at the moment that you buy your ticket, you also choose the acts you want to see. In this experimental version only an integer number which represents the act will be stored. However, they want to distinguish between 2 types of tickets. With a standard ticket you may select maximum 10 acts, a VIP ticket has no limitations. Also the price of both is different and when visitors want to cancel their ticket, also the pay-back policy is different. Standard tickets are not refundable, VIPs should have the option to ask for a refund, which returns the total money they paid. Every ticket stores information about the name of the owner, his national number and the price, which depends on the number of selected acts. It should be possible to add acts to a ticket (currently only the number which represents the act) and you may assume that the given numbers are always valid. Every act you add to a ticket increases the total ticket price. Since you don’t know when a ticket is complete (=all acts have been added), you need a way to check this. A standard ticket should automatically be set to ‘complete’ once 10 acts have been selected.

Tickets can then be added to a festival. A festival has a fixed number of acts that will take place. Only tickets which are “complete” will be accepted and to avoid that the same person would buy multiple tickets, there is also a check if a ticket having the same national number is not yet available. When a ticket is added to the festival, the number of attendants for all chosen acts will be updated automatically. You need also the functionality to ask for the act number which has currently the highest number of attendants. Visitors may cancel their ticket, depending on the type of ticket the amount of pay-back is returned and the attendant numbers are updated.

Since the national number is constructed by the date of birth followed by a number in the format yyyymmdd-nr the organizers want the possibility to check if there is at least one ticket owner who is celebrating his birthday on a given date. To be able to execute some quick simulations with the system, they want to be able to import “dummy” tickets from a text file with all the necessary information.

## Bibliography

**Create a detailed class diagram starting from following problem description: define the needed classes, their attributes, the signature of their methods and the relations between the different classes.**

You are asked to develop an application to facilitate the administration of a bibliography. Your first client has only paper publications and web publications, but make your solution easily extendable to other types of publications.

Each bibliography is identified by the name of a certain editor. Every publication is identified by a title, the year of publication and a unique identification code. It should be possible to store the author and all co-authors of a publication. For a paper publication the name of the journal where it has been published and the impact factor[[1]](#footnote-1) of this journal is stored, for a web publication the URL and the number of views. To facilitate input of data, it should be possible to make a basic copy of an existing publication. Currently this feature is only implemented for paper publications.

You need to be able to add publications to the bibliography. It should be possible to get a string with an overview of all publications in the bibliography, where per publication at least title and year are shown. It should be possible to search for the oldest publication, to remove all publications from a given (co-)author and return the removed publications as a list. Since impact factors may change over time, it should be possible to update in one function call all the impact factors from a given journal and you want to know the sum of all impact factors for all publications. Add a way to print an overview of the publications in chronological order (oldest first); all publications in the same year are in alphabetical order on title.

To facilitate the start-up of the system, there should be the possibility to import a list of all publications from a given text file.

1. [Look up](https://en.wikipedia.org/wiki/Impact_factor) what an impact factor is and how it is calculated to decide on what type to use. [↑](#footnote-ref-1)