Object Oriented Software Development 2021-2022
Professors: S. Desmet and K. Gerling
Lab assistants: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken and J. Wauters.

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.

KU LEUVEN

Object Oriented Software Development 2021-2022

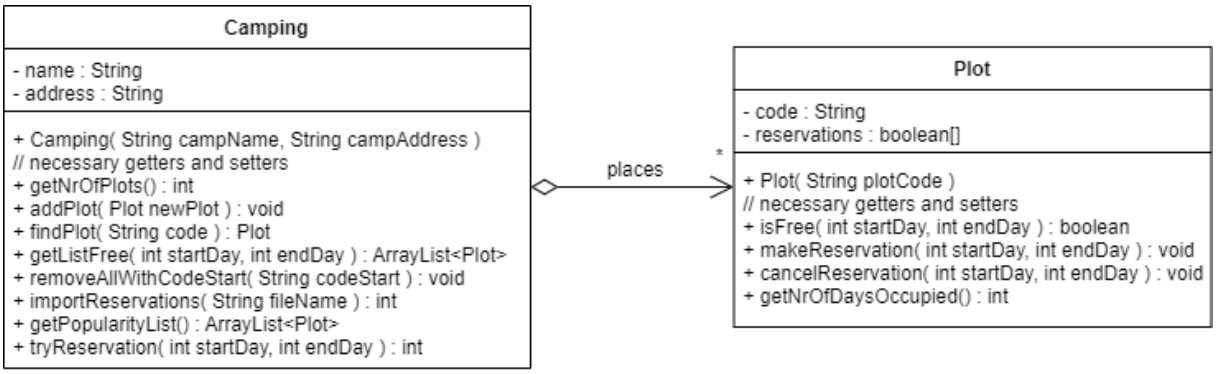
Professors: S. Desmet and K. Gerling
Lab assistants: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken and J. Wauters.

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 wants to automate the management of its reservations. The camping provides a number of plots where visitors can stay, each of which has a code and an array of booleans denoting the days on which this plot is reserved (true if still available, false if reserved). The camping is open between May 1st (index 0 in the array) and September 30th (index 152 in the array). This means that dates are always represented as integers between 0 and 152 for this exercise.



Test 1 [0.5 point] Tests the constructor of *Plot* and a few getters.

Test 2 [0.5 point] Tests if all days are marked as available after construction of a new plot.

Test 3 [1 point] Tests the method *isFree*, which returns true if the current plot is still available in the period between *startDay* (inclusive) and *endDay* (non-inclusive), and returns *false* if it is not. The method also returns *false* if the start and/or end day don't fall between the correct extremes. Additionally, we test the method *getNrOfDaysOccupied*, which counts the total number of days this plot has been reserved for.

Test4 [1 point] Tests if we can make and cancel a reservation. Of course it is impossible to make a reservation if (part of) the period we want to reserve isn't available. If we try to make a reservation on a day which is not available, an error message is printed (you can choose what it says).

Test5 [1 point] Tests the constructor of *Camping* and a few getters.

Test 6 [2 points] Tests if we can add new plots to the camping. Note that plots need to have a unique code. If we try to add a plot with a code that already exists, nothing happens.

Test7 [2 points] Tests if we can look up a certain plot based by its code. This method returns the resulting plot, or null if no plot exists with the given code.

Test8 [2 points] Tests if we can generate a list of all plots which are available between a certain start and end date.

Test9 [2 points] Tests if we can remove all plots whose code starts with a given string.

Test10 [2 points] Before your planning software existed, the camping's reservations were stored in text files. Here we test the method *importReservations*, which imports a number of reservations from such a text file. These text files have the following format:

- Code of the plot
- Start day of the reservation
- End day of the reservation

This structure is repeated a number of times. The method returns the number of imported reservations. If the method comes across invalid information (e.g. a code for a nonexistent plot), it returns the number of imports performed up until that point as a negative number, and then stops the import.

Test11 [2 points] The camping wants to make a distinction between two different types of plots: pitches, which are empty spaces where visitors can set up their own tent or caravan, and fully furnished tents. For pitches we store whether or not they have electricity. For tents we store the build year, plus an *ArrayList* of strings with possible extras that can be added to the tent (crib, microwave, …)

Implement this functionality through the use of inheritance. Check the test code to see which classes and methods you are expected to create.

Test12 [2 points] Tests whether we can produce a list of all plots, ordered by increasing popularity (in other words, the plot with the lowest amount of reservations is at the front of the list, followed by that with the second lowest amount, and so on). In case there are plots with an equal number of reservations, the order in which you add them to the list doesn't matter.

Test 13 [2 points] As a final challenge, you will create a method *tryReservation*. This method tries to spread a reservation over multiple plots if no single plot exists that can take the full reservation. Use the following approach: first you look for the plot with the longest free period from the start of the reservation, and you reserve the available time. Then you do the same again, starting from the first day which could not yet be reserved, and so on until the full period has been reserved. This method returns how many plots were needed to make your reservation. Remember that you can use extra helper methods if you deem this useful.

Object Oriented Software Development 2021-2022
Professors: S. Desmet and K. Gerling
Lab assistants: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken and J. Wauters.

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.

KU LEUVEN

Object Oriented Software Development 2021-2022

Professors: S. Desmet and K. Gerling
Lab assistants: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken and J. Wauters.

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.

KU LEUVEN

Object Oriented Software Development 2021-2022

Professors: S. Desmet and K. Gerling
Lab assistants: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken and J. Wauters.

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] 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 what an impact factor is and how it is calculated to decide on what type to use.