University of St Andrews

School of Computer Science

CS1002 – Object Oriented Programming

Assignment: **W11 – UML Modelling and Implementation with Inheritance**

Deadline: 30 November 2018 *Credits*: 25% of coursework mark

MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Aim / Learning objectives

Your aims in this practical are to:

- improve your understanding of UML models
- implement a solution in Java conforming to the UML model using inheritance

By the end of this practical you should be able to:

- model a specification expressed in English using inheritance as a UML class diagram
- implement this model in Java using multiple classes

Introduction

This practical involves modelling a given scenario as a UML class diagram and implementing this model in Java as accurately as possible. You will need to decompose the problem and identify the set of classes and methods you will need to write. You will also need to test your solution carefully. It is possible to get a mark of 17 by producing a very good solution to the basic requirements along with an excellent report. Getting a mark above 17 will require completing one or more extensions in addition to an excellent basic solution.

Setting up

The first steps should be familiar:

- Log in to a machine booted into *Linux*.
- Check your email in case of any late announcements regarding the practical.
- Launch *Terminal*. Use appropriate commands to create a new directory called *W11-Practical* in your *cs1002* directory and move to it.
- Create a new *LibreOffice Writer* document for your report, add the appropriate header at the top, and save it as *W11-Practical-Report*.
- Create a *source* directory inside the *W11-Practical* directory and move into it. This is where your program source code should reside.

If you can't remember how to do any of these steps, refer back to the instructions for the Week 1 practical at:

https://studres.cs.st-andrews.ac.uk/CS1002/Practicals/W01/
https://studres.cs.st-andrews.ac.uk/CS1002/Lectures/

The Task

Practical specification

In this practical you will create a UML class diagram and then develop a Java program to satisfy the following specification.

You are asked to design, implement and test a software system for a company that sells holidays.

Each holiday offered by the company has a unique id, location, cost, starting date and duration. A holiday is one of 3 kinds:

- adventure (for eg, trekking in the Himalayas or the Rockies),
- relaxation (for eg, a beach holiday in the Canary Islands), or
- culture (for eg, sightseeing in Florence or Agra).

Adventure holidays have a difficulty level associated with them. Relaxation holidays have a description of the amenities available. Culture holidays have one or more aspects (art, architecture, music, etc) and may have a lecture on the aspects of interest as part of the deal.

The company employs a number of staff, each of whom has a staff id, name and salary. Members of staff belong to one of 2 categories: holiday guides and admin staff. Each holiday guide specialises in one type of activity (for eg, climbing or trekking) while admin staff have one or more skills.

Each holiday has a member of admin staff associated with it. In addition, each adventure holiday also has a holiday guide associated with it.

The program should support the following functionality:

- Print the names and salaries of all staff of the company in a tidy format of your choice;
- Calculate the total salary cost of the company;
- Give all staff a salary increment of a given percentage;
- Print the names and specialisations of all holiday guides;
- Print the names and difficulty level of all of adventure holidays that are offered by the company and the names of the guides associated with them;
- Print the details of all holidays offered by the company that fall within a date range;
- Find the most expensive holiday offered by the company;
- Print the details of all culture holidays which include a lecture.

UML design

You must design classes with suitable attributes and methods to represent the requirements specified above. Think about the most appropriate modifier for each attribute and represent it clearly.

Classes should contain at least one constructor and suitable getter and setter methods in addition to the functionality given in the specification above.

Use inheritance where appropriate.

Include multiplicity and navigability details for associations between classes.

You may use *draw.io* or another tool of your choice to create the class diagram. Remember to save your work as you go along. If you accidentally close your browser, you do not want to lose all your work! Include a *pdf* version of the model in your submission and insert it in the report if legible.

This design will be the blueprint for the implementation that follows.

Java implementation

Translate the UML design into Java code, creating corresponding classes and paying particular attention to associations.

You should then define another class called *W11Practical* in a file *W11Practical.java*, which will contain only a static *main* method. Inside the *main* method, you must create a number of objects of appropriate classes and invoke their methods to show that your code correctly carries out the required tasks.

You can compile your program by executing `javac *.java` from the source directory and run it by executing `java W11Practical`.

Guidelines

Plan ahead and decide a suitable set of classes. Where should each piece of information be kept and where do operations belong?

Look up pre-defined Java classes that might be useful to model certain concepts such as dates. These classes model the concepts better and will help reduce the chances of invalid data.

Reuse methods to implement other methods where applicable.

You do not have to get user input to test your implementation. You can make up suitable values in your *main* method.

Testing

Now think about how you are going to test your program. What test cases will you need to test the different possibilities?

Program presentation

As always you need to consider the presentation of your program code—see the section on **Program presentation** in Practical W03 if you cannot remember what to do.

https://studres.cs.st-andrews.ac.uk/CS1002/Practicals/W03/W03-Simple_Calculation.pdf

In addition to what you did in Practical W03, you should also include comments to say what your methods do.

The Autochecker

This assignment will **not** make use of the School's autochecker. You must carry out and document testing of your own to demonstrate that your solution satisfies the required functionality.

Report

Your report **must** be structured as follows:

- **Overview**: Give a brief overview of the practical: what were you asked to do?
- **Design**: Describe the design of your model and code. Justify the decisions and choices you made. In particular, include a copy of the class diagram, a brief explanation of why you designed your model in the way that you did, how you translated the model to Java and any interesting features of your implementation in Java.
- **Testing**: Describe how you checked that your model is complete and how you tested your program. Your report should include the output from a number of test runs to demonstrate that your program satisfies the specification; however, do **not** include tests for every possible test condition.
- **Research:** Research and write a short paragraph each on UML activity diagram and UML use case diagram. Remember to acknowledge your sources.
- **Evaluation**: Evaluate the success of your model and program against what you were asked to do.
- **Conclusion**: Conclude by summarising what you achieved, what you found difficult, and what you would like to do given more time.

Don't forget to add a header including your matriculation number, tutor, and the date.

Upload

Package up your *W11-Practical* folder, a **pdf** version of your UML model and a **pdf** copy of your report into a zip file as in previous weeks, and submit it using MMS, in the slot for Practical W11.

Extension Activities

The activities in this section aren't compulsory, though you need to do at least one of them to achieve a mark above 17. Try them if you're interested and have spare time. As in the previous practical, separate your extension work by copying and pasting your work in new files and then developing them separately from the original solution. *Do not attempt any extensions until you have produced a complete, good solution to the basic requirements*.

- Extend your solution (both model and program) so that you capture the notion of clients who book holidays with the company. Each client has an id, name, email and one or more interests. Implement the functionality to match the interests of the client with the holidays offered by the company and suggest holidays for a given a client.
- For each holiday, implement the functionality to book the holiday and keep track of bookings made. This should allow the solution to print the number of bookings for each holiday and the total revenue of the company. Explore different ways (textual, graphical, etc) of presenting the number of bookings for all the holidays to the user.
- Research and find a construct in Java that allows you to define a type with a set of values specified by you, as opposed to a pre-defined set of values. For example, the type *boolean* has two pre-defined values *true* and *false*. However in some cases, you may want to define a type whose values aren't pre-defined in Java. For example, you may define a type *Day* that only allows the values *Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday*. Find out how

to model this in UML. Then incorporate this construct into your model and implementation for this assignment wherever applicable.

- Can you think of any other extensions that would enhance the program? If you can, implement them and highlight them clearly in your report.

Please note

Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in next section). Some specific descriptors for this assignment are given below:

Mark range	Descriptor
1 - 6	Very little evidence of work; software does not compile or run, or crashes before doing any useful work. You should seek help from your tutor immediately.
7 - 10	A decent attempt which gets part of the way toward a solution, but has serious problems such as failure to compile, crashing during execution, or a lack of object-oriented structure.
11 - 13	A solution which is mostly correct and goes some way towards object-orientation using inheritance, but has major issues such as errors in operations, poor placement of attributes and methods in classes, incorrect associations between classes or inheritance relations, poor readability, or a weak report.
14 - 15	A correct solution using inheritance accompanied by a good report, but which can be improved in terms of code quality, for example: poor method structure, poor encapsulation and use of modifiers, lack of comments, or lack of reuse.
16 - 17	A correct solution, which demonstrates excellent design and code quality, good method decomposition, good design of required classes with appropriate use of inheritance, elegant implementation of methods with reuse where appropriate, and is accompanied by a well-written report.
18 – 19	As above, but implementing one or more extensions, accompanied by an excellent report.
20	As above, but with multiple extensions, outstanding code decomposition and design, and accompanied by an exceptional report.

Policies and Guidelines

Marking
See the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty
The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good academic practice
The University policy on Good Academic Practice applies: