CMPU4060
OBJECT ORIENTED SOFTWARE DEVELOPMENT
PROJECT ASSIGNMENT HANDOUT

DR ANEEL RAHIM

1. Introduction

This handout is the specification for your individual project assignment.

2. Developing your project

This project is designed to practice, develop and assess your programming skills and object oriented programming competence. Reflect upon the learning outcomes and that upon completion of the module the student should be able to:

(1) Design an object-oriented software application

(2) Implement a software application using an object-oriented programming language utilising core object-oriented programming concepts, and develop problem solving skills as part of this process

(3) Test and debug an object-oriented software application

(4) Implement basic algorithms and data structures using an object-oriented programming language

(5) Select and evaluate appropriate methods, including algorithms and patterns, for the implementation of object-oriented solutions

In developing your project, you are encouraged to interact with your classmates during and outside of the formal laboratory sessions. Through discussion, you can develop your understanding and awareness of techniques and methods for designing, developing and implementing your project. You are encouraged to carry out self-guided research and reading in addition to the suggested texts and web learning tools to augment your programming skills and knowledge surrounding the project problem. This will differentiate your solution and enhance your learning experience.

2.1. **Plagiarism.** By submitting your assignment, you are declaring that the project assignment is entirely your own work. Plagiarism is strictly forbidden and if you are unsure, please refer to the Appendix B of the student handbook (Appendix B: Information Leaflet In Relation To Cheating) if you need to refresh your memory regarding plagiarism. Your project should be all your own work. As a quick reminder and non-exhaustive list from a programming perspective plagiarism includes:

: Sharing code or using someone else's code

: Reworking other students or work found on the web and changing variable names

: Copying another students design, report, tests

: Getting someone else to write code for you

If there is any suspicion of code being shared or copied, students may be required to attend an oral defence of their assignment to discuss their submissions. Students should be able to explain their coding decisions and validate their understanding of their project submission.

3. Project Submission

Deadline: 23:59, Sunday 1 May 2016.

- Submit via Webcourses.
- Multiple submissions possible. Only last one is graded.
- Link WILL stay open after deadline.
- Late submissions be penalised by an absolute mark of 10% per day late (so after 10 days link will close).
- Marking Scheme below – remember to address each item to maximise marks
- Zip up your project directory and submit.

3.1. **Deliverables.** Project document, python code for program and tests, supporting files.

3.2. Marking Scheme.

3.2.1. *60% Project Implementation.*

(1) Project implements the specification and program has the ability to correctly use the data provided to perform the specified functionality

(2) Code quality and style: good programming practices displayed (e.g. using suitable algorithms, naming conventions, resource management, appropriate use of comments)

(3) Error and Exception Handling: Program is robust to unexpected data inputs and handles errors in an appropriate way; program makes appropriate use of Python exception handling in error management

(4) Tests: Appropriate tests to exercise the code at a system and unit level

(5) Jazz: Project goes beyond the minimum specifications and demonstrates students ability to use a variety of programming skills beyond the basic minimum defined scope of the project.

3.2.2. *20% Project Design.*

(1) Applying OOP concepts to solution (e.g. selection of suitable classes, modules, interfaces between classes, potential for code reuse)

(2) Class Design (e.g. appropriate methods, attributes, attribute visibility)

3.2.3. *20% Documentation.* This is the opportunity to demonstrate your understanding of the software development life cycle from design through implementation and testing. The document does not need to be long or verbose but should contain information about the program assumptions, inputs and outputs. The design of the solution should be addressed with details of the modules, classes and/or function used and data files used or created. The tests strategy should also be covered.

High marks in documentation will be achieved through:

(1) accuracy (matching the implementation)

(2) clarity (well structured, written and without typographical, spelling, grammar mistakes)

(3) succinctness (concise but understandable)

(4) completeness (covering the required sections)

4. Guidelines

4.1. **Specification.** Imagine you work for a small Irish software company that has won the tender to deliver a fuel management software solution to a large Irish Airline. The airline is entering the European air freight cargo business and wants a software tool to manage their fuel purchasing strategy. Each week, an aircraft will fly up to six trips. The airplane will start and end in the same home airport each week and can optionally visit one other airport twice. The company has a number of different aircrafts that have different fuel capacities and the cost of fuel varies from airport to airport. Given a list of 5 airports (including to the home airport) that a given plane needs to visit in a week, the most economic route must be found.

- The distance between airports is calculated as the great circle distance between them
- The cost of fuel is assumed to be 1 euro per 1 litre at Airports where the currency is Euros
- The cost in of fuel in airports where the local currency is not euros is assumed to be the exchange rate from the local currency to euro. e.g. in you travel from London to Dublin and the exchange rate is GBP1 = €1.4029 and you purchase 1000 litres of fuel it will cost €1402.

4.2. **Product Features.** The program with take file inputs and give file outputs using CSV formats. It should also have a basic GUI to allow input of an individual itinerary. At a minimum, the program should work with a command line interface to interact with the user- this is not necessary if you have file and/or GUI interfaces.

4.3. **Code style and efficiency.** The program code should be created in a logical and efficient manner. This includes: efficient use of data structures (think for loops rather than repeated code), code re-use in functions and classes (breaking code up into logical chunks that keep code-blocks and files reasonable length i.e. no 1000 line files!), efficient storage and searching of data (think dictionaries)

4.4. **Class Model Structure.** Create classes that model the problem and abstract the main program from the underlying data structure. For example, the mainline code shouldn't need to understand anything about parsing a CSV file or calculating great circle distance or currency rates calculation. This means it should be apparent that you have you created classes that model the problem and assigned useful attributes and methods to the classes. The solution should define classes for holding data that will be used in the problem, e.g. there will need to be a class for Currency and Airport? (HINT: Yes, there will!)

4.5. **Errors and Exception Handling.** How robust is your program to errors? Have you implemented error checking and exception handling, especially to handle unexpected user inputs or data?

E.g. do you have error handling to identify and warn if the wrong number of airports are contained in the input, or one or more of the airports is not a valid airport in your list?

4.6. **Comments.** Is all of your code well commented? Are the variable names and methods/functions meaningful?

4.7. **Testing.** Test code to run on and demo the program under a variety of valid and error conditions that will exercise your program and its exception handling.

4.8. **Jazz.** Anything I didn't ask for but is a useful feature for the product, e.g. a map that displays the route

4.9. **Documentation.** Document should be no more than 4–5 pages. Your name and student number should be in the header of each page.

(1) Function Specification: Describe what does your program does. (max. 1 page)
  - What product features have you implemented?
  - What features from the handout have you left out.
  - How do I use your program? (Command line UI/GUI or via files?)
  - What (if any) jazz should I look out for?

(2) Design: Assumptions, Inputs and Outputs. Describe/Illustrate your object model and how your program was designed (max. 1 page text and 2 page diagrams)
  - Class diagram
  - Block diagram of main algorithm(s)
  - Text description to explain diagrams

(3) Testing: How have you tested your program. Have you got a suite of tests that exercise the key components? (max. 0.5 page)

4.10. **Supporting materials.** CSV files for airports, countries, currencies and aircraft

Example Airport Data:

```
596,Cork,Cork,Ireland,ORK,EICK,51.841269,-8.491111,502,0,E,Europe/Dublin
597,Galway,Galway,Ireland,GWY,EICM,53.300175,-8.941592,81,0,E,Europe/Dublin
599,Dublin,Dublin,Ireland,DUB,EIDW,53.421333,-6.270075,242,0,E,Europe/Dublin
```

Example Currency Data:

```
Iraq,Iraq,IQ,IRQ,368,IRQ,iq,IQ,IRQ,964,IRQ,IZ,118,IRQ,IQD,IRAQ,3,Iraqi Dinar,368,Yes
Ireland,Irlande,IE,IRL,372,IRL,ie,IE,IRL,353,IRL,EI,119,IRL,EUR,IRELAND,2,Euro,978,Yes
```

Example Currency Rates:

```
Australian Dollar,AUD,0.7253,1.3791
Euro,EUR,1,1
US Dollar,USD,0.9488,1.0541
```

Example Aircraft Data:

```
code,type,units,manufacturer,range
A319,jet,metric,Airbus,3750
A320,jet,metric,Airbus,12000
747,jet,imperial,Boeing,9800
MD11,jet,imperial,McDonall Douglas,12670
```