Programming and Systems Development Lab Exam

You will sit the actual lab exam during your scheduled lab session on Thursday 31 October, or during a different time by consultation with the Teaching Office.

During the exam

During the lab exam, you will use one of the lab machines in its default configuration. When you come into the exam, you will be able to bring nothing with you. You are also not allowed to access any previously written documents or code files from your home drive or the web. If you do, it will be considered plagiarism.

As in a normal examination, communication between candidates during the laboratory exam is strictly forbidden.

A candidate who wishes to print a document during the exam should summon an invigilator, who will collect it from the printer and deliver it.

A candidate may leave early, but only after summoning an invigilator, who will ensure that submission has taken place. Any candidate who experiences a hardware or software problem during the examination should summon an invigilator at once.

Leave the laboratory quietly when you are done; the exam may still be in progress for other groups.

Overview

This exam consists of two parts: part 1 requires you to write a Python program, and part 2 requires you to write a Java program. The instructions, allocated marks, and submission instructions are given below for each part. Note that each part has a total of **24 marks** allocated for correctness, as indicated; in addition, **1 mark** for each part will be allocated for appropriate coding style (commenting, formatting, variable names, etc).

Part 1: Python

You should write the code for each task below in a **separate python source file**, as indicated below. You should then submit **all source files** through Moodle as described at the end.

Python Task 1: Database [5 marks]

Create a new SQL database called **BookInfo** that will store a list of authors and the books they wrote. It will have two tables. The first table should be called Authors and contain the following data:

Name	Place of Birth
Agatha Christie	Torquay

J.K. Rowling	Bristol
Oscar Wilde	Dublin

The second table should be called Books and contain the following data:

ID	Title	Author	Data Published
1	De Profundis	Oscar Wilde	1905
2	Harry Potter and the chamber of secrets	J.K. Rowling	1998
3	The seven dials mystery	Agatha Christie	1929
4	The picture of Dorian Gray	Oscar Wilde	1890
5	Murder on the Orient Express	Agatha Christie	1934
6	Harry Potter and the prisoner of Azkaban	J.K. Rowling	1999

Save your solution to this problem in a file **python_task1.py**.

Python Task 2: Displaying database [7 marks]

Using the **BookInfo** database, display the list of authors and their place of birth. The program should ask the user to enter a place of birth and then show the title of the book, date published and author's name for all books by authors who were born in the location they selected.

Save your solution to this problem in a file **python_task2.py**.

Python Task 3: Querying database [5 marks]

The program should ask the user to enter a year and display all the books published after that year, sorted by the year they were published.

Save your solution to this problem in a file **python_task3.py**.

Python Task 4: Writing to file [7 marks]

The program should ask the user for an author's name and then save all the books by that author to a text file called **Booklist.txt**, each field separated by dashes so it looks as follows (for example):

> This file contents the following record(s):
>
> 2 - Harry Potter and the chamber of secrets - J.K. Rowling - 1998
>
> 3 - The seven dials mystery - Agatha Christie - 1929
>
> 4 - The picture of Dorian Gray - Oscar Wilde - 1890

Finally, the content of the text file should be displayed as above.

Save your solution to this problem in a file **python_task4.py**.

Python submission

Ensure your submission files are named exactly as they are supposed to (i.e. **python_task1.py, python_task2.py, python_task3.py, python_task4.py**).

Go to your moodle account, and submit your files at the appropriate submission link:

> *Moodle > Programming and System Design > Lab Exam > Python Submission*

Java

In this lab, you will create a set of classes representing playable characters in a video game, as well as a class representing a player of that video game. The following are the main features that you will implement – the remainder of this lab sheet describes in more detail how the classes should be created.

Every **Game Character** has two properties: a **name** (e.g., "Superman") and a set of **powers** (e.g., for Superman, this might include Flying, Invincibility, Speed, and Energy Blast).

A **Player** of a video game has a set of game characters which they have already purchased.

During the game, a player will encounter a series of levels – and each level requires a set of characters with particular powers. For example, one level might need characters with powers including Flight, Strength, and Science. To begin such a level, a player must choose a set of characters that supply all of the necessary powers. For example, if a level needs character with Flight, Strength, and Science, the player might choose Superman (to provide Flight and Strength) along with Doctor Octopus (to provide Science). If a player does not have characters that are able to supply the necessary powers, they are not able to start the level.

Example

Assume that the player has the following characters in their set:

- Robin (providing Weapons)
- Starfire (providing Flight and Energy Blast)
- Cyborg (providing Strength and Weapons)
- Beast Boy (providing Transformation)
- Raven (providing Magic)

The following are the game characters the player should choose in various circumstances

- If the required powers are Weapons and Strength, they should choose Cyborg
- If the required powers are Flight, Strength, and Transformation, they should choose Starfire, Cyborg, and Beast Boy.
- If the required powers are Transformation, Magic, and Science, then this player cannot start the level as they do not have any characters that provide all of the required powers.

Java Task 1: Power (2 marks)

*Note about implementation: all classes created in this exam should be put in the **superhero** package.*

You should create an enumerated type **Power** with the following values:

> CLONING, COMPUTER, ENERGY_BLAST, FLIGHT, INVINCIBILITY, MAGIC, MAGNETISM, SCIENCE, SMALL, SPEED, STRENGTH, TELEPATHY, TRANSFORMATION, WEAPONS, WATER

Java Task 2: GameCharacter (6 marks)

You should then create a class **GameCharacter** representing a game character including the following properties:

- name (a String)
- powers (a set of Power objects)

You should create a constructor to set the value of the above fields with the following signature (note that this constructor uses **varargs** for its final parameter):

> **public GameCharacter(String name, int cost, Power... powers)**

In addition, your **GameCharacter** class must satisfy the following requirements:

- Your class must be **immutable** – that is, it should not be possible to change any of the fields of the class in any way after it is created.
- The class must have a complete set of **get** methods for all parameters, but no **set** methods
    - Be sure that none of your **get** methods violate the immutability of your class
- You should provide appropriate implementations of **equals()**, **hashCode()**, and **toString()**.
    - It is fine to use the auto-generated versions of **equals()** and **hashCode()** if you want, but you must write your own (non-auto-generated) **toString()** method.
- You must write **descriptive Javadoc comments** for the class and for all class members (fields and methods).

Java Task 3: Player (6 marks)

Once your **GameCharacter** class is finished, you should implement a **Player** class to represent a player of the game, making use of the **GameCharacter** class defined above. **Player** should a single field:

- The set of game characters that they currently own (represented as **GameCharacter** objects) – you can choose whatever data type you want for this field

Player should also have the following methods:

- A constructor that initialises the set of characters correctly
- An appropriately named method to add a character to the set
- A **get** method to retrieve the field value.

Java Task 4: chooseCharacters (10 marks)

In your **GameCharacter** class, implement one additional public method, as follows:

- **public Set<GameCharacter> chooseCharacters(Power... neededPowers)**

Note that you are free to add any number of **private** helper methods that you want, but the above should be the only **public** method in the class other than the constructor and the **get** method.

This method should implement the behaviour shown above in the example: given a set of powers, this should return the set of characters required to provide all of the requested powers, or **null** if it is not possible to provide the powers. If no set of characters can be found to cover all of the required powers, this method should return **null.** If more than one set of characters can provide the required powers, any valid set of characters can be returned (e.g., in the example, a level requiring Weapons and Magic could return Raven and Robin, or Raven and Cyborg).

Java submission

Ensure your submission files are named exactly as they are supposed to – i.e., **Power.java**, **GameCharacter.java**, **Player.java**. Also **be sure that all classes are in the *superhero* package.**

Go to your moodle account, and submit all of the above files at the appropriate submission link:

Moodle > Programming and System Design > Lab Exam > Java Submission