Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student submitting an assignment.

Name:	Michael Savage
Programme:	CASE3
Module Code:	CA341
Assignment Title:	Comparing Imperative and Object-Oriented Programming
Submission Date:	22/11/19
Module Coordinator:	David Sinclair

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the referencing guidelines found at http://www.dcu.ie/info/regulations/plagiarism.shtml, https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

Name: Michael Savage Date: 21/11/19

Comparing Imperative and Object-Oriented Programming

Introduction

The first assignment from the ca341 module deals with the comparison of an imperative and object-oriented approach to a to-do list. In my case, I am comparing an Imperatively written Java program with an object-oriented Python program.

I intuitively thought of using Java for the object-oriented approach as it follows that paradigm very closely. Java encourages the use of objects and their attributes and methods but I chose to reverse the roles and attempt to use Python for this approach and write a Java to-do list imperatively as best as I could.

Files

The python program include the objects queue.py, task.py, and event.py and they are manipulated by main.py. An optimal scenario is observed by running test.py. The empty file __init__.py is needed to import the object files. The java program includes Todo.java and a text file called “toDoList.txt.”

Code

Each implementation starts by printing the usage in the terminal and then interprets user input. The python commands are called from the Queue class and the Java commands are called from a function called help.

Python

```python
def commands(self):
    print("Available commands.\n" +
    "'add e' to add an Event.\n" +
    "'add t' to add a Task\n" +
    "'job' to print first job.\n"+
    "'all' to show all jobs.\n" +
    "'help' to print commands.\n" +
    "'r' to remove.\n" +
    "'q' to quit.")
```

Java

```java
public static String help(){

    return "Available commands.\n" +
        "'add e' to add an event.\n" +
        "'add t' to add a task.\n" +
        "'job' to print the most recent job.\n" +
        "'all' to print the all jobs.\n" +
        "'help' to print commands.\n" +
        "'r' to remove most recent job.\n" +
        "'q' to quit.";
}
```

List of commands printed to terminal.

Adding Event/ Tasks

The Queue is a Linked-List that helps to organise the Events and Tasks. Every to-do job is enqueued after the user inputs the relevant data related to the Task or Event. In Java, Functions are coded in every step required to solve the problem. Adding an event in my Java implementation gets the file ready to be written to then calls another function to get user input.

Python

```python
def addEvent(q):

    #Event has a date, start time, and location
    newEvent = Event(input("Event Name: "),
        input("Date: "),
        input("Start Time: "),
        input("location: "))
    q.enqueue(newEvent)
```

Java

```java
    public static void addEvent(){

        try(FileWriter f = new FileWriter(toDo, true);
            BufferedWriter bw = new BufferedWriter(f);
            PrintWriter file = new PrintWriter(bw)){

// An event has a date, a start time and a location.

            String data = getUserInput("event");
            file.println(data);

        }
        catch (IOException e){
            System.out.println("Unable to write to file.");
        }
    }
```

Adding an Event to the Queue.

I decided to use separate files for the objects in Python so that I could satisfy one of the four main principles of object-oriented programming; Abstraction. In comparison, the java to-do program has all the code in a single file. The imperative approach deals with global variables and public functions so that everything is visible throughout the program.

An understanding of the functions is necessary, rather than a reliance on models that are able to solve it. The focus of imperative programming is how the problem should be solved. This means the Java program should be very easy to understand from glance.

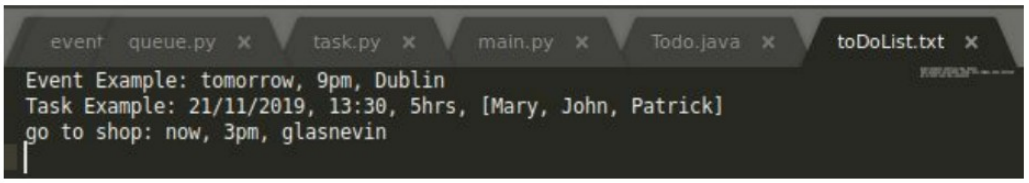
Storing data

In the Python approach I used a Linked-List to store the jobs. When running test.py we can see that the list items are printed one line at a time. while in the Java approach I used a file to read and write to.

Python

```
1. Show all jobs:
Football, 25/12/2024, 21:00, 4hrs, ['Pat']
Buy milk, 12/1/2039, 11:00, 4.30hrs, ['John', 'Mary', 'Ben']
New Years, 1/1/2020, 12:00, Dublin
```

Java



Storing the jobs

Removing data

Reading and writing to a file in Java was an arduous task, especially removing the top line of the file which represented the top of the ‘queue’. I had to skip past the first line and write to a temporary file and rename it to the original. The temporary file was then deleted.

Unlike this way, The python approach did not cause a lot of effort. It was a matter of simply pointing the head node to the next node in the list.

Python

```
temp = self.head.item
self.head = self.head.next
try:
    self.head.prev = None
    return temp
```

Java

```
// temp file is called toDoList.temp
File tempFile = new File(todo.getAbsolutePath() + ".tmp");
BufferedReader br = new BufferedReader(new FileReader(todo));
PrintWriter pw = new PrintWriter(new FileWriter(tempFile));
String line = null;

String remove = br.readLine();
while ((line = br.readLine()) != null) {
    pw.println(line);
    pw.flush();
}
pw.close();
br.close();

//Delete the original file
if (!todo.delete()){
    System.out.println("Could not delete file");
    return;
}

//Rename the new file to the filename the original file had.
if (!tempFile.renameTo(todo))
    System.out.println("Could not rename file");
}
```

Remove the top job.

Conclusion

Even though Java is inherently object-oriented, I found that it was interesting trying to find a solution other than a list for the to-do program. Writing imperatively made me think in depth about the functions I used and it gave me a better understanding of my code. If I were to repeat the assignment, I would use a formatted string with new-line characters instead of a file. Using files in Java meant importing many utilities and error handling.

Before I had started this assignment, I knew that I would be biased towards using Python. It is a simple to read language that I used extensively for problem-solving in different modules. I am happy with my implementation of The Task, Event, and Queue objects but I imagine more object-oriented principles could have been used like polymorphism.

Overall, a to-do list would be best implemented in Java through object-oriented programming. It is a small program with many variations of Task and Event objects.