Comparing Imperative and Object-Oriented Programming

Name: Ethan Sharkey
Student Number: 17355756

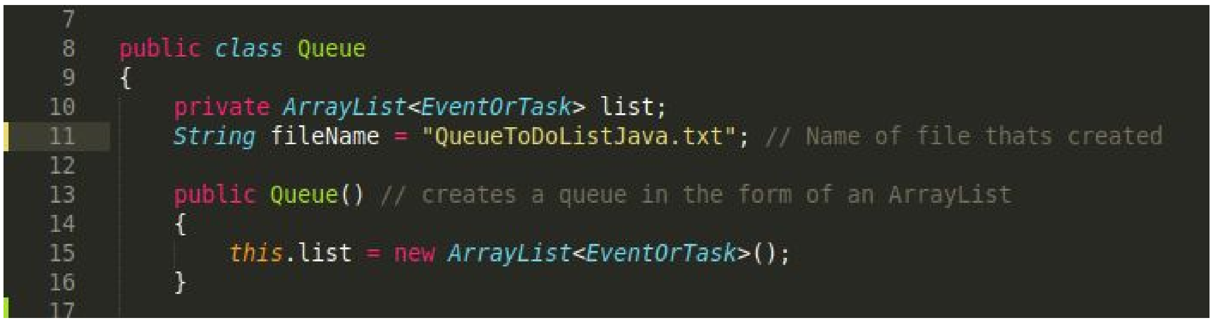
I chose to complete the object oriented side of this assignment in Java as I feel it is the language that I have the best knowledge of and I know it is very well suited for object orientation. For the imperative side of it I chose Python as it is also a language I have great knowledge of.

Object-Oriented Programming Advantages:

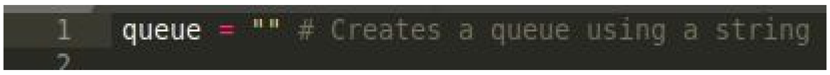
The object oriented side of this project immediately started off easier than the imperative part of this assignment as I had a lot more to work with variable wise as I was able to implement composite data types as well as primitive data types throughout my program whereas only primitive types were available to me to implement in the imperative programming part of the assignment. However, that doesn't mean there weren't any difficulties with the object oriented side of this assignment.

Starting with the advantages of this assignment being object oriented, being able to create a class Queue. The resulting instance of queue was hugely beneficial as I was able to then encapsulate all the methods and variables regarding Queue in the Queue class such as the main methods, enqueue() and dequeue() etc. In python for the imperative programming side, I created a variable called queue and assigned it an empty String. For the Object Oriented Queue, I used an ArrayList as the Queue and for the imperative side I used a string, a primitive data type, as the queue. I discovered it is a lot easier to manipulate and construct ArrayLists over Strings due to the fact that Strings are immutable whereas ArrayLists are mutable.

Difference between Creating a Queue: (OOP VS Imperative)



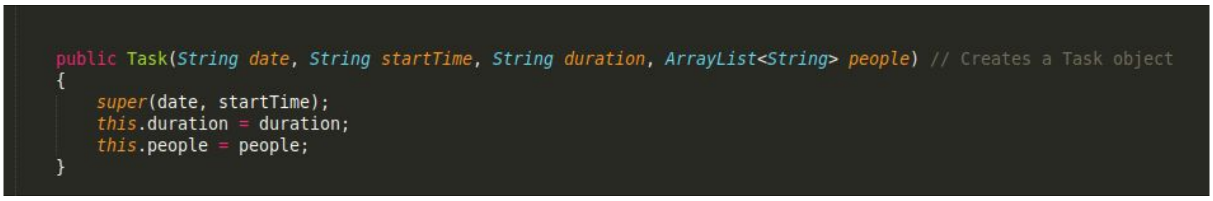
```java
public class Queue
{
    private ArrayList<EventOrTask> list;
    String fileName = "QueueToDoListJava.txt"; // Name of file thats created

    public Queue() // creates a queue in the form of an ArrayList
    {
        this.list = new ArrayList<EventOrTask>();
    }
```



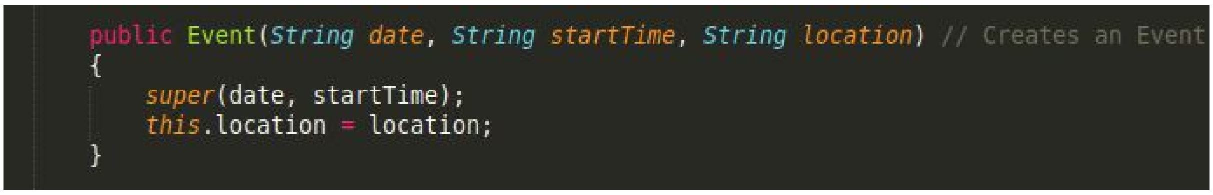
```python
queue = "" # Creates a queue using a string
```

An additional advantage that came with writing up the object oriented part of the assignment was that the Events and Tasks that were being added to the queue were their own object whereas compared to the imperative side, I ended up creating them as strings. There was no way to differentiate between an Event or a Task in the imperative part however, there was when it came to the object oriented side. Since this was possible it also made the code a lot cleaner, more human readable and encapsulation became a factor once more. Coupling between the classes is also evident.

Event and Task Object in Java: (Object Oriented)

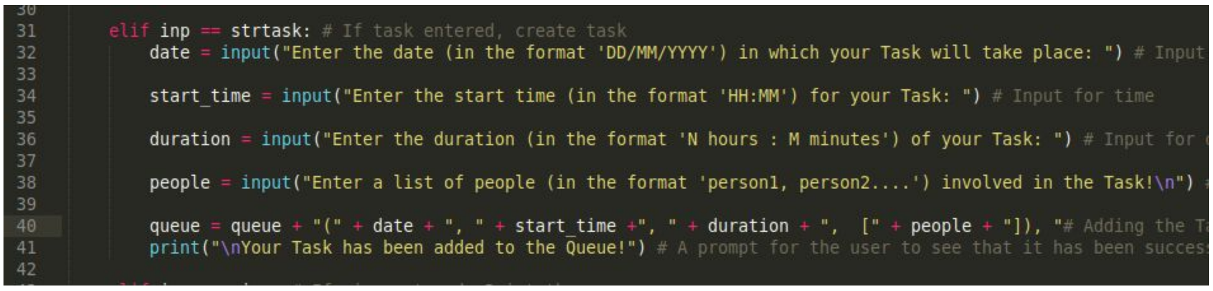


```java
public Task(String date, String startTime, String duration, ArrayList<String> people) // Creates a Task object
{
    super(date, startTime);
    this.duration = duration;
    this.people = people;
}
```

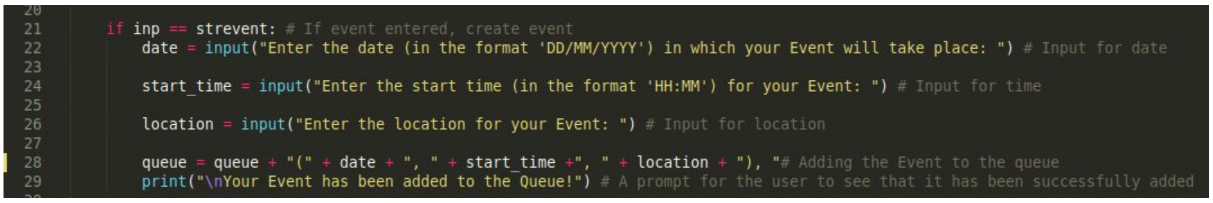


```java
public Event(String date, String startTime, String location) // Creates an Event
{
    super(date, startTime);
    this.location = location;
}
```

Event and Task String in Python: (Imperative)



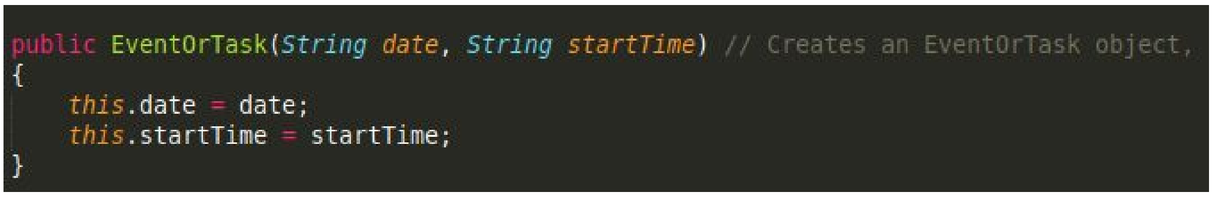
```python
    elif inp == strtask: # If task entered, create task
        date = input("Enter the date (in the format 'DD/MM/YYYY') in which your Task will take place: ") # Input

        start_time = input("Enter the start time (in the format 'HH:MM') for your Task: ") # Input for time

        duration = input("Enter the duration (in the format 'N hours : M minutes') of your Task: ") # Input for

        people = input("Enter a list of people (in the format 'person1, person2....') involved in the Task!\n")

        queue = queue + "(" + date + ", " + start_time +", " + duration + ",  [" + people + "]), "# Adding the T
        print("\nYour Task has been added to the Queue!") # A prompt for the user to see that it has been succes
```



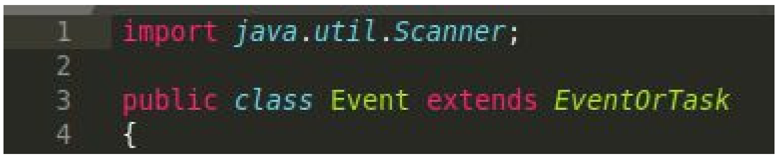
```python
    if inp == strevent: # If event entered, create event
        date = input("Enter the date (in the format 'DD/MM/YYYY') in which your Event will take place: ") # Input for date

        start_time = input("Enter the start time (in the format 'HH:MM') for your Event: ") # Input for time

        location = input("Enter the location for your Event: ") # Input for location

        queue = queue + "(" + date + ", " + start_time +", " + location + "), "# Adding the Event to the queue
        print("\nYour Event has been added to the Queue!") # A prompt for the user to see that it has been successfully added
```

With this advantage it also becomes a disadvantage with the object oriented side which was that I had to create a parent class called EventOrTask and its child classes, Event and Task extended from it. I implemented this as two different objects could not be placed in the same ArrayList so I had to make the two of them inherit from the same parent class to make it possible for them to be in the same ArrayList and therefore the same queue.

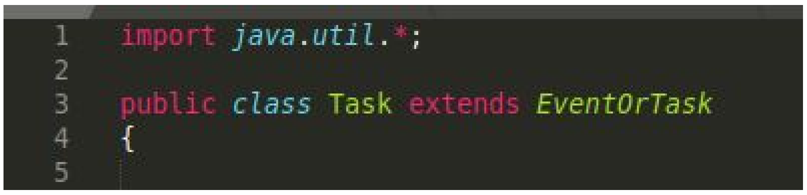
Showing Parent Class and Child Classes Extending from It:



```java
public EventOrTask(String date, String startTime) // Creates an EventOrTask object,
{
    this.date = date;
    this.startTime = startTime;
}
```



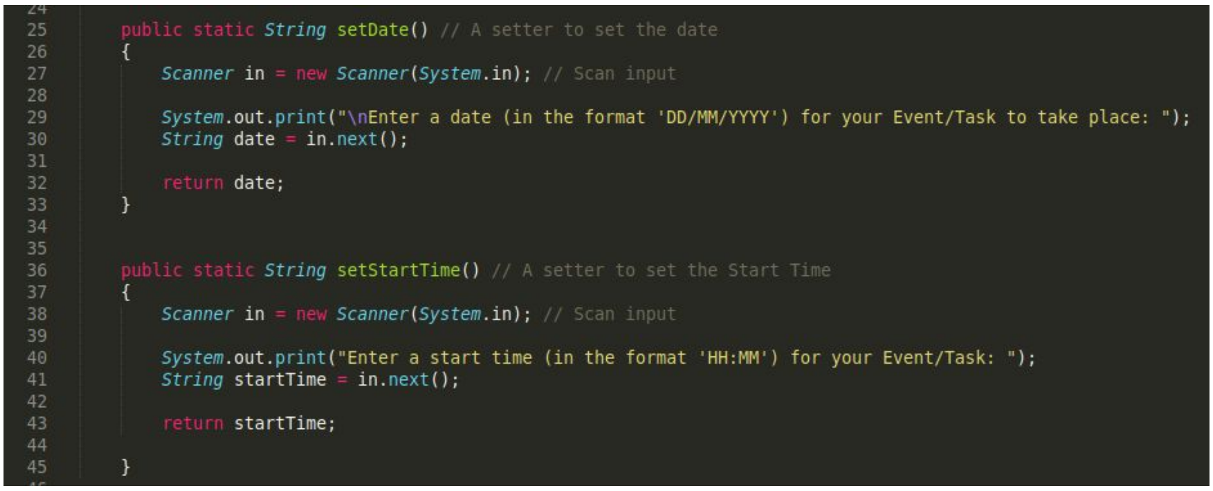
```java
import java.util.Scanner;

public class Event extends EventOrTask
{
```



```java
import java.util.*;

public class Task extends EventOrTask
{

```

Although I mentioned above that this is a disadvantage that more than one object could not be stored in the same queue, It resulted in an advantage later on. Throughout the assignment, it saved me from writing duplicates of code down the line as Event and Task have similar characteristics so the code that would've been duplicated only had to be written once in their parent class, EventOrTask. It also enabled me to implement inheritance throughout my program which is a huge feature of object oriented programming. None of this was possible with the imperative part of the assignment as it had to be one large program in one file using just primitive data types.

Example of code that would've had to be written twice:



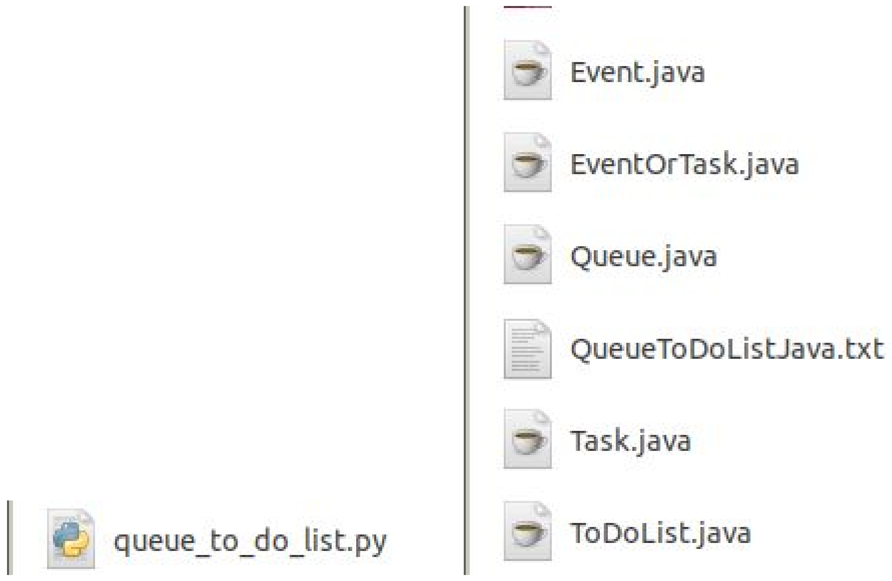
```java
    public static String setDate() // A setter to set the date
    {
        Scanner in = new Scanner(System.in); // Scan input

        System.out.print("\nEnter a date (in the format 'DD/MM/YYYY') for your Event/Task to take place: ");
        String date = in.next();

        return date;
    }


    public static String setStartTime() // A setter to set the Start Time
    {
        Scanner in = new Scanner(System.in); // Scan input

        System.out.print("Enter a start time (in the format 'HH:MM') for your Event/Task: ");
        String startTime = in.next();

        return startTime;

    }
```

The benefits that come with an object oriented design is evident throughout my program as it contains encapsulation, inheritance and coupling throughout. The resulting product from this is human readable, easy to interpret, reusable code. Compared to the imperative part of the assignment these are huge benefits, however there were advantages when it came to writing the imperative part of this assignment.

Imperative Programming Advantages:

The most visible advantage with the imperative side of this assignment is that it is significantly shorter compared to the object oriented side of it. There was also the benefit of implementing the program with one file whereas with the object oriented part every class had to be in its own file. Therefore, multiple files are used throughout the program which becomes messy within the directory if you have multiple classes. In some ways it is easier to read the code as it essentially reads from top to bottom of the file compared to the object oriented part where the program jumps from file to file to execute upon the methods that are called.

Example of the file difference:



queue_to_do_list.py

Event.java
EventOrTask.java
Queue.java
QueueToDoListJava.txt
Task.java
ToDoList.java

Overall I believe that completing this assignment with an object oriented design was much more beneficial than doing it with an imperative design and throughout this comparison, it is quite evident that there are a lot more advantages creating it with an object oriented design over an imperative design. The main distinction between the two is that you can work with a variety of data types the object oriented way compared to only primitive data types the imperative way. This made the most impacting difference when deciding which style was more practical for this assignment.