CA341

*Comparative Programming Languages*

***Student:*** *Nigel Guven*

***ID:****14493422*

Imperative Style v Object Oriented Style Analysis Report

**Due Date:** *10am 20th November*

1. **Table of Contents**
2. Programming Paradigm………………………………4
3. The Two Primary Paradigms…………………………5
   1. Imperative Paradigm
   2. Object-Oriented Style
4. Programming Requirements…………………………6
   1. Imperative Style – C++
   2. Object-Oriented Style – Java
5. Analysis – Imperative Style………………………….7
   1. C++ Code
   2. C++ Program Description
6. Analysis – Object-Oriented Style……………………10
   1. Java Code
   2. Java Program Description
7. Paradigm Differentiation…………………………….14
8. References…………………………………………...15
9. **Objective**

*Implement the following program in an* **imperative** style***and****an* **object-oriented** *style using any languages you choose provided they have an open-source implementation.*

*Implement a* ***to-do list*** *program that stores the list of things to do in a* ***queue*** *(first-in/first-out). Each item in the to-do list can be either a* ***task*** *or an* ***event****. Tasks have a* ***date****, a* ***start time****, a* ***duration*** *and a* ***list of people*** *assigned to the task. An event has a* ***date****, a* ***start time*** *and a* ***location****.*

1. **Programming Paradigm**

*Programming paradigms are a classification standard for programming languages based upon how they operate. Languages can have multiple paradigms of which there are many. Paradigms are classified by the following:*

* **Execution Model**

The execution model is how a programming language operates. It is implemented by the activities that occur once a program is compiled and invoked. Some programs like Java and C++ operate using a main method while more functional languages like Haskell are statically typed which allows the compiler to catch errors before allowing execution of the program.

* **Global Modification**

Global Modifications or side effects are interactions which allow functions modify variables, raise exceptions and read/write data to and from files. Side effects allow programs to interact globally i.e with filesystems, computers. Many high-level programming languages use side effects to increase their scope. For functional languages and assembly languages, side effects are restricted or not utilised because of the increased risk of creating bugs and IO errors.

* **Syntax and Semantics**

The syntax of a program defines the structural ordering of a paradigm for a programming language. Syntax is the set of rules and processes that create the structure of a language. Semantics refers to the signifiers of data types, keywords and symbols. SQL is a declarative language which is used to query a database in order to obtain information. It is structured by a set of specific clauses which can perform different operations on a database. Object-oriented languages like Python and Java are organised into classes which are hierarchical and form the basis for creating object.

1. **The Two Primary Paradigms**

**2.1 Imperative Paradigm**

Imperative programming is a paradigm of computer programming in which a program describes a sequence of instructions which can change during the run-time stage. Data is transmutable in that it can transform from its value at compilation to end of a program at run-time. The flow of an imperative program is controlled using loops, conditional statements and function calls where supported by the language. Imperative programming is reliant upon a step by step routine structure.

Some examples are:

* C++, Ruby, FORTRAN

**2.2 Object Oriented Paradigm**

The Object-Oriented Style of programming or OOP is the approach to programming where all computations are viewed as objects and how they can be manipulated. An object may be a simple date type like and integer or a float or it might be made up of multiple data types which define that specific object. A person could be visualised in an object via a string of characters for a name, an integer for age, weight and height and some functions which can be used to construct or instantiate and object, declare a statement which can alter the state of an object or even be a Boolean function to compare two different objects. OOP is based around the concept of the class. Classes are definitions for the format for the structure of an object. They are the underlying blueprint which states the variables and functions for an object.

Some examples are:

* Java, Python, Eiffel

1. **Programming Requirements**

**3.1 Imperative Style – C++**

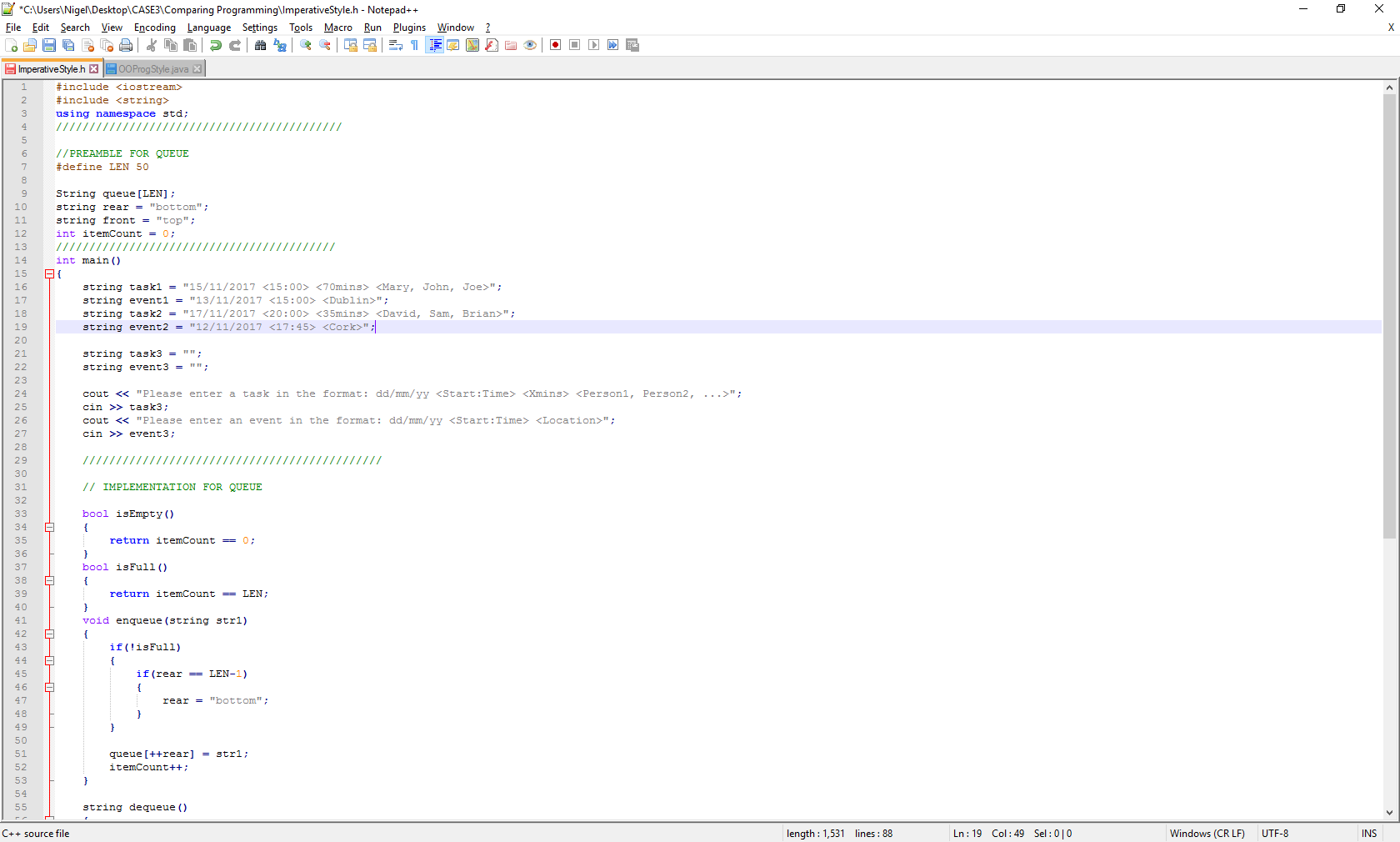
I decided to use C++ to display the imperative style of programming. C++ is a multi-paradigm programming language. It does utilise OOP style but for the task at hand, I focused on the imperative paradigm. C++ is an abstract language which contains features of both high level and low-level languages. It is used in kernels and driver scripting as well as for writing software applications. From the perspective of paradigms, C++ is stronger since it allows for multiple paradigms in the one program if need be. However, its classes are not as well designed as Java classes, therefore Java is stronger for Object-Oriented programming.

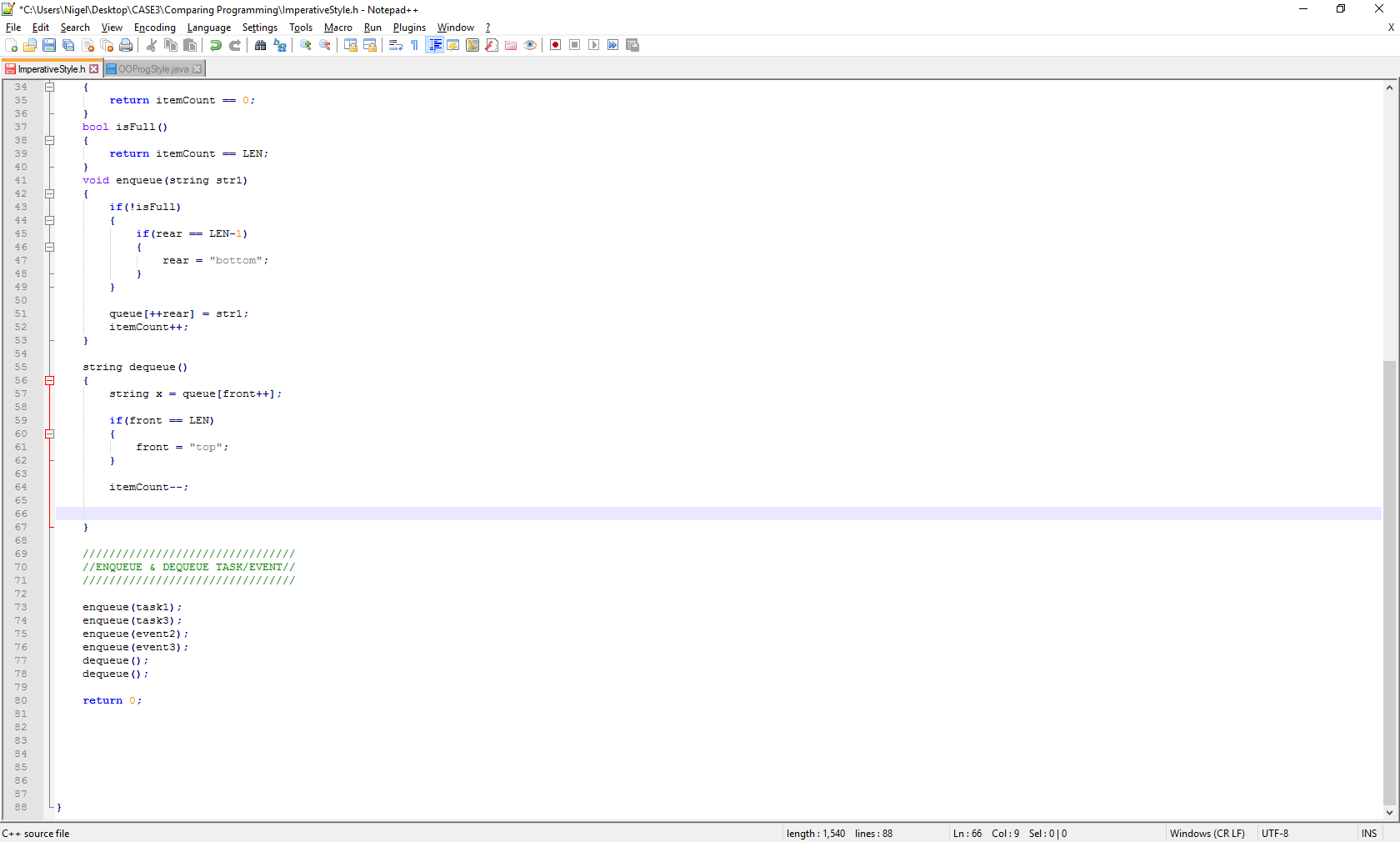
**3.2 Object-Oriented Style – Java**

I wanted to use Java as the Object-Oriented language. It is perfect for OOP paradigm style as the Java language is built around using objects and manipulating them in data structures. Java is indeed influenced by C syntax and can support multiple paradigms. However, its strength is also its weakness in that it has a bias towards the OOP style. Java is utilised in building software applications and in web development. Java is a hierarchical language with the use of interfaces and classes which make it more organised than C++ and from that it is easier to learn and debug.

1. **Analysis – Imperative Style**

**4.1 C++ Code**





**4.2 C ++ Program Description**

This C++ program contains a queue implementation which takes in strings. The two strings that are involved are tasks and events. The format for tasks and events are displayed in the following:

* Task

‘’dd/mm/yy <Start:Time> <Xmins> <Person1, Person2, PersonN>’’

* Event

‘’dd/mm/yy <Start:Time> <Location>’’

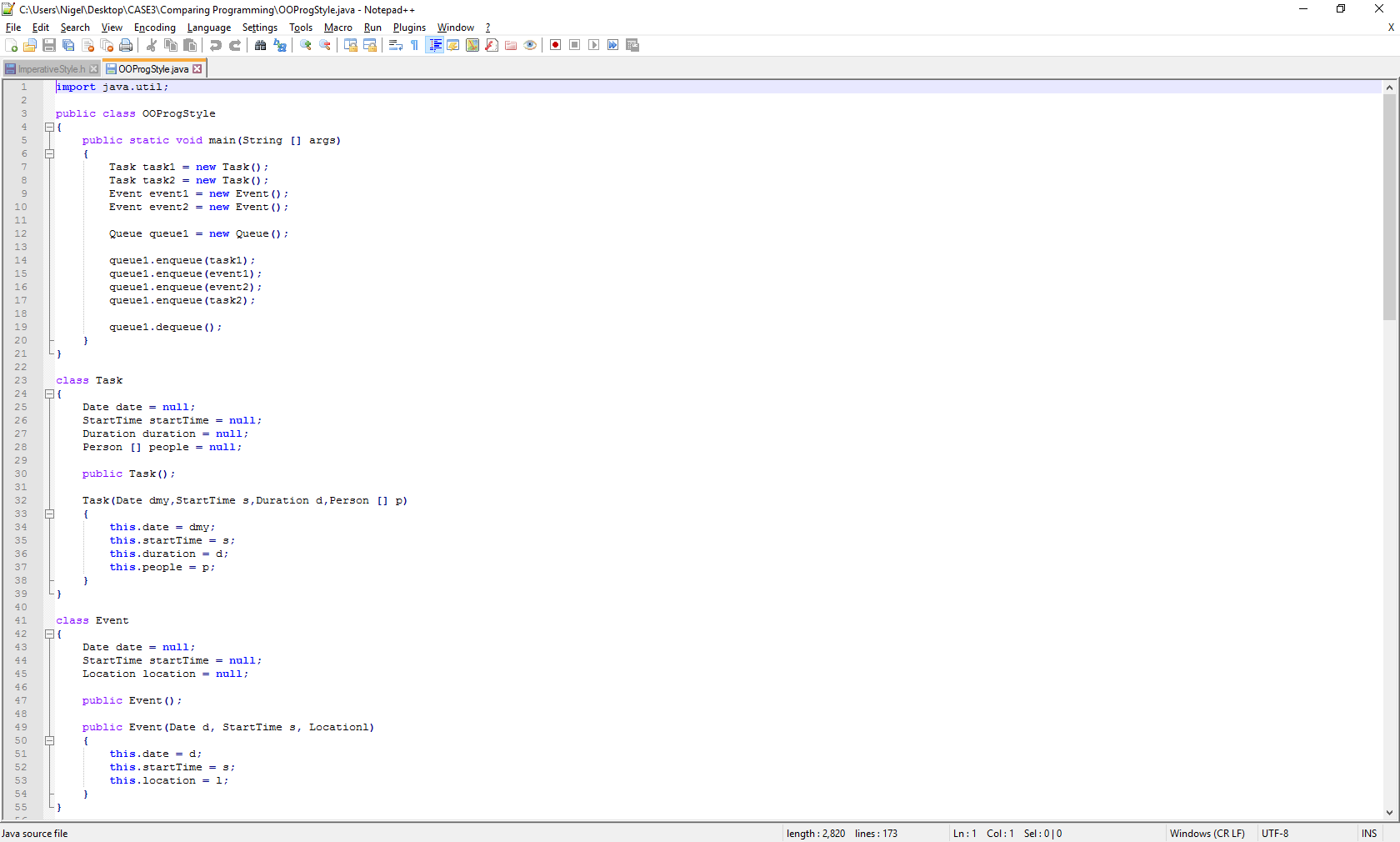
The tasks and events are treated as such as strings. The queue implementation is basic with isEmpty(), isFull() and enqueue()/dequeue() functions. The length of the queue must be defined as C++ treats the queue as an array. There is one task and one event that are undefined at compile-time. These then accept inputs of data to be given a value. This is an example of Imperative Style programming where date can change during run-time.

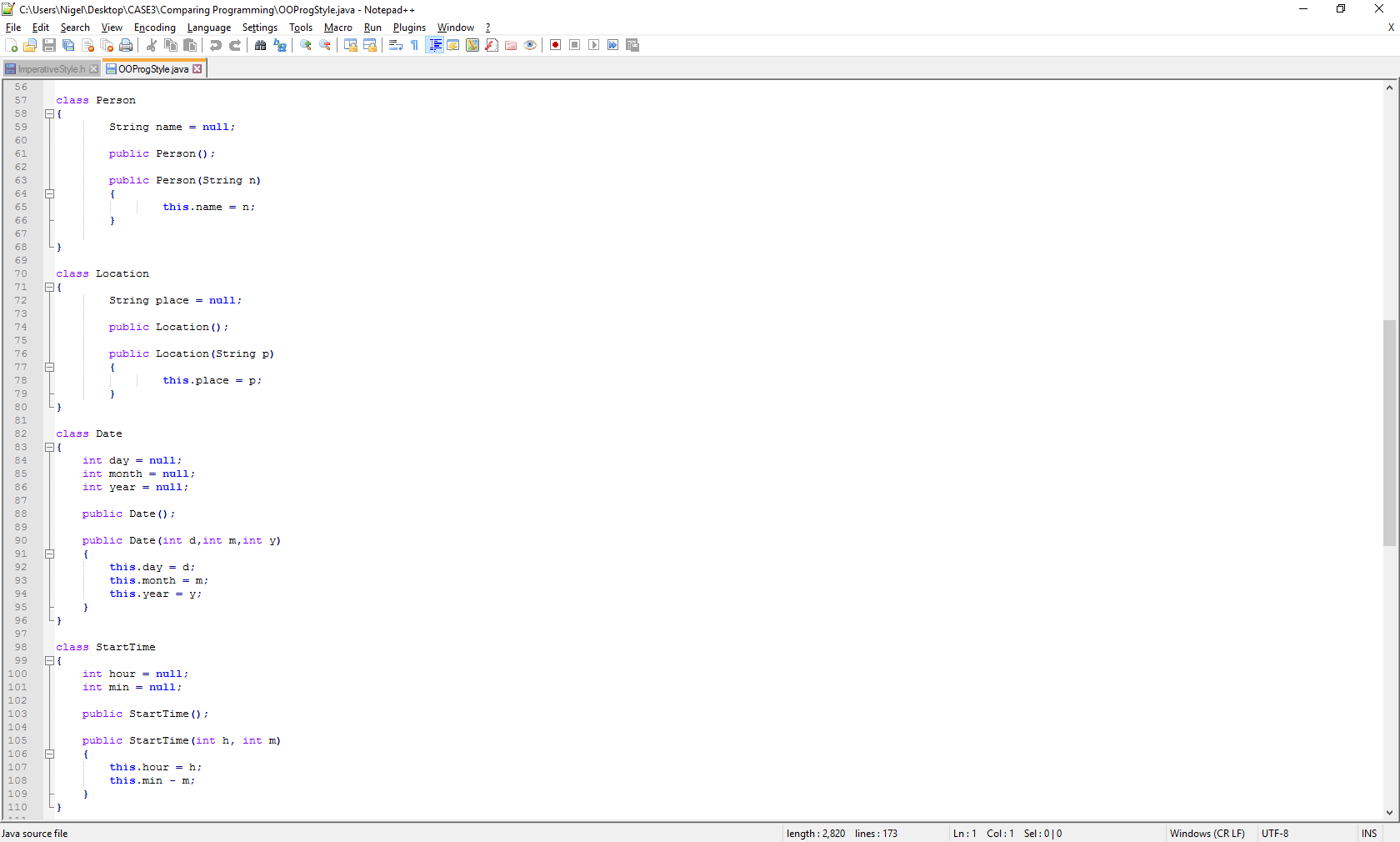
The program runs in a step-by-step state. Therefore, any functions that are necessary must be defined before the main function. C++ does not support functions written after a main method unless they are defined in the preamble. I did not define my functions in the preamble to show that this style is imperative which runs in a step-by-step state.

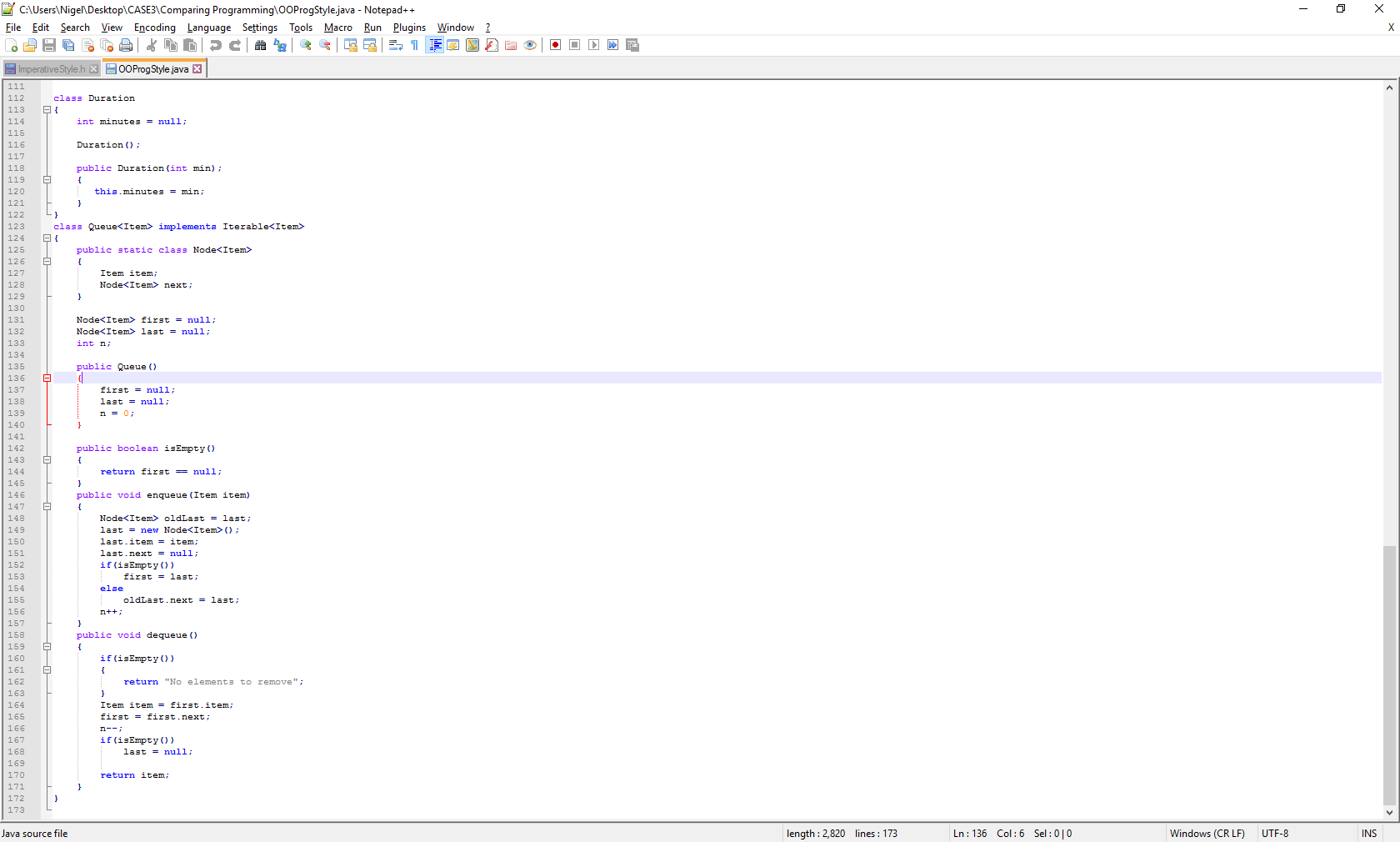
The queue enqueues the tasks and events from the back and dequeues the items at the front. This is the standard design of a queue implementation. The end of the program is reached when the main functions returns the value of zero. This effectively terminates the program and again is an imperatively designed function.

1. **Analysis – Object-Oriented Style**

**5.1 Java Code**







**5.2 Java Program Description**

The Java program is based around multiple classes which treat the tasks and events as objects. The task object comprises of a date, start time, duration and an array of people which are constructors which define the task object. The event object consists of a date, start time and location. Each of their constructors is an object.

The task and event objects inherit the other objects as constructors. Each class has an empty constructor method and a full constructor method.

The queue is further complicated in that it must record two different types of objects into its data structure. To do this it must become a generic class taking in some item which is defined in a Java implementation as a generic object. If the queue did not have a generic type, then it would generate an exception which would not allow two different data types to fill the queue.

The queue has similar features in its implementation to the imperative solution. The queue is different in that instead of taking strings, it records each object in its list as a node which has a value and a link to the next Node in the queue.

The design of the program is non-imperative as can be seen in that the classes are listed after the main method. This is a key difference between Java and C because it is unnecessary to declare functions before a main function in Java.

In the main method each object must be instantiated with values before they can be used. For the purpose of this program, the constructor values were set to null.

1. **Paradigm Differentiation**

I analysed both programs following the implementation of the queue and task/event design. I came up with the following conclusions:

* Object-Oriented programming is much more organised and neat. Imperative style can be tangled and hard to debug. It would be more beneficial to use OOP style as there would be less debugging to do.
* Imperative style is much simpler for writing short programs. It uses up less memory and therefore can be more efficient given the circumstance.
* OOP requires much more time to code as can be seen by the fact that each class constructors must be given a value and then instantiated in a main method.
* OOP style is more versatile and is more useful in that it creates the concept of large data structures which can hold countless objects.
* Imperative programming can be versatile in that it can allow program data to evolve. The use of functions and methods also give the imperative paradigm an advantage.
* OOP is abstract which permits its data to be transmutable in that objects can be extended via interfaces. This creates a hierarchy of data which can be further expanded if needed. OOP supports multiple inheritance allowing objects to inherit the design of multiple objects.

1. References

<http://www.computing.dcu.ie/~davids/teaching.shtml>

<https://www.computerhope.com/jargon/i/imp-programming.htm>

<http://cs.lmu.edu/~ray/notes/paradigms/>

<http://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP>

<http://www.geeksforgeeks.org/queue-data-structure/>

<https://en.wikipedia.org/wiki/Object-oriented_programming>

<https://en.wikipedia.org/wiki/Imperative_programming>

<https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B>

END OF ASSIGNMENT