Programming Booklet Showcasing a Variety of Reports and Tasks

A Collection of Code and Written Reports

Original University Module: **Programming**

Current Focus: **Developing Coding Ability**

A report originally created to fulfil the degree of

BSc (Hons) in Computer Science

University of Brighton

Originally Submitted 20/08/2020

Abstract

This paper is a combination of coding tasks and projects from a variety of programming languages, some of which include written reports that were submitted alongside them. The languages that have been predominantly worked with and therefore showcased in this booklet are Java and Python. Each topic is divided into separate sections, which has then been split up into separate tasks and reports.

Contents

1 Reports

1.1 Definitions

Listed below is a short list of keywords that are used throughout this booklet.

Algorithm - A set of instructions that are followed in order to solve a problem.

Argument - A way to provide more information to a function, which can then be used by it.

Arrays - A series of memory locations, each of which holds a single item of data, but with each box sharing the same name. All data in an array must be of the same data type.

Class - A possible structure to create an object in an object-oriented programming language. It defines a set of properties and methods that are common to all objects of one type.

Function - A sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

Inheritance - An ability of an object to take on one or more characteristics from classes of objects.

Method - A part of an object, allowing it to perform an action, in order to modify itself or to return a value. A specific type of function: it must be part of a "class", with access to the class' variables.

Pointer - An object in many languages that stores a memory address. This can be that of another value located in computer memory, or in some cases, that of memory-mapped computer hardware.

Private - A keyword that specifies access level and provides programmers with some control over which variables and methods are hidden in a class. The opposite of this keyword is public.

Register - A small amount of fast temporary memory that is quickly accessible within the processor where the ALU or the CU can both store and change values needed to execute instructions.

Script - A program or a sequence of instructions that is carried out by another program, rather than a computer processor as a compiled program, processing the steps line-by-line from top to bottom.

Static - A static variable is one that does not change in its lifetime. For example, an array is a “static data structure” because its size is set at the moment it is created and it cannot ever change.

Strongly Typed - A programming language that typically has stricter typing rules at compile time, implying that errors and exceptions are more likely to happen during compilation.

Structure - A way to organise code and data through a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

Weakly Typed - A programming language that typically has looser typing rules and may produce unpredictable or even erroneous results or may perform implicit type conversion at runtime.

1.2 Language Comparisons

Java and Haskell are two languages that consist of many contrasting properties and uses, both with their own strengths and weaknesses. The Java community is largely rooted in industry; decades of programming has resulted in a large set of libraries available for anyone to use. Haskell, however, is used more by those of an academic background, due to its heavy use of "pure" logic and abstraction. This report will compare and contrast the two languages in various ways, in order to determine how they are effective in solving problems that have arisen in software development over the years.

1.2.1 Documentation

Section 1.1.1 will give a brief overview of Java and Haskell. Section 1.1.2 will be a high-level insight into some of the various differences and similarities between the two languages, and the contrasting approaches each language takes. Section 1.1.3 will be a review that reflects on various implications. Section 1.1.4 will be the appendix, expanding on various examples shown throughout this report.

1.2.1.1 A Brief Overview of Java

The Java Language Specification describes Java as a "*general-purpose, concurrent, class-based, object-oriented language. It is designed to be simple enough that many programmers can achieve fluency in the language*" (Oracle, 2013). Like other popular languages, Java is an imperative language that uses control structures such as if, while and for loops. Java is used by over 40 million users and is the 3rd most popular language on GitHub (GitHub Inc., 2019) with over 64,000 major corporations using Java for desktop applications and backend web systems (Gupta K. , 2018).

One of the main factors attributed to Java's growth is arguably its scalability, popular among both enterprises and scaling start-ups alike; Twitter moved from Ruby to Java in 2010 to cope with its ever-increasing demand during the 2010 US election (Gade, 2011). Java's scalability is due to the language being statically typed, meaning all variables and expressions are known at compile time. Java's prominence in the industry is also due to the fact it is backwards compatible, allowing older versions to still run after newer versions are released. This is greatly beneficial for corporations, as they do not need to worry about rewriting all their code every time a new update is released.

1.2.1.2 A Brief Overview of Haskell

The Haskell 2010 Language Report describes Haskell as a high-level, purely functional language that has incorporated innovations from "*years of research on non-strict functional languages*" (Marlow, Haskell 2010 Language Report, 2010). Haskell is also a *statically typed* language, though like most modern functional languages, writing programs requires a slight change in perspective when compared to imperative languages.

One way to explain how Haskell works if by equivalating it to a more common example; anyone who has used a spreadsheet before has some experience of functional programming. In a spreadsheet, each cell has a specified value, either a standalone value or one that is expressed in terms of another cell values. It is taken for granted that the spreadsheet with compute cells in an order which respects their dependencies. Haskell takes a similar approach, specifying values as an expression, rather than a sequence of commands like an object-oriented language would in order to produce a final result.

1.2.2 Comparing Concepts

Both Java and Haskell are used as general-purpose programming languages; each have a managed environment that is of a higher level than assembly. This means that programmers do not need to worry about the state of the underlying machine, especially when dealing with freeing any unused memory. There are however some notable differences, which this section of the report will cover. This includes the primary goal of each language, how code reuse is managed, and how languages often influence each other over time as developers create various ways of solving certain issues.

1.2.2.1 Design Goals

The language best suited for you to use depends heavily on the overall aim of your program. The very first Java update came with the promise "*Write Once, Run Anywhere*" (Tech Insider, 2007) by allowing one set of code to be runnable anywhere on almost any machine. This is achieved via the *Java Virtual Machine* (*JVM*), taking code written in Java and 'translating' it into the specific machine code required for that particular computer. Instead of compiling directly into machine code, Java files are compiled to an in-between code called *bytecode*. The *JVM* then interprets this *bytecode* and converts it to the required output based on the operating system the code is running on. This allows developers to not need to worry about platform differences, only the team maintaining the *JVM* do.

An added bonus of the *JVM* is that since it is isolated from the operating system, no external process can access the application data unless the *JVM* allows it, which adds a whole extra layer of security to the language (Mikhalchenko, 2017). These are both major selling points for developers who desire to reach a large audience across multiple platforms, without needing to devote extra resources into creating or maintaining secure programs for lesser used systems. Haskell on the other hand was born out of a desire for a common, purely functional language. Whilst there were dozens of non-strict functional languages at the time, there was a consensus that overall progress was being held back by the "*lack of a common language*" (Marlow, Haskell 2010 Language Report, 2010).

When Haskell was first being designed back in 1987, a committee of various experienced functional programmers was formed to agree on a new standardised language. One of the main aims for this new language, named after the logician Haskell B. Curry whose work provided the basis for much of the incorporated logic within it, was for it to be "*suitable for teaching, research and applications, including building large systems*" (Marlow, Haskell 2010 Language Report, 2010). This led to Haskell utilising *lazy evaluation*, where expressions are only evaluated when needed in order to produce the program's output, and by extension never calculates things that it never needs.

1.2.2.2 Polymorphism

Both Java and Haskell were originally designed with some kind of polymorphism in mind, the general concept of which can be explained with a simple real-life analogy. The President of the United States employs polymorphism by having advisers, including legal, medical, and military advisers. In order to make sure everything runs smoothly everyone should only be responsible for one objective. When the president asks their advisers to advise them, they should all know how to respond accordingly.

1.2.2.2.1 Code Reuse

One major benefit of polymorphism is the ability to reuse any code or classes (Grinnell College, 2000). Java supports subtype polymorphism, a more restricted form of it since it efficiently limits the set of possible types to itself and following subtypes. Haskell handles this concept slightly different, via the use of parametric polymorphism. This is different to the previous Java example, as Haskell defines types that are generic over other types. The genericity can be expressed by using type variables for the parameter type to replace them explicitly or implicitly with specific types when necessary, thereby allowing the argument of a function to be made to accept any type instead of a specific one. Both subtype and parametric polymorphism are useful solutions to problems that have arisen in software development. Whilst there is no 'one size fits all' answer to which type is better, the advantages of parametric polymorphism are undeniably significant; so much so that they would be added to Java in a future update.

1.2.2.2.2 Generics

Before 2004, Java did not contain any other kind of polymorphism besides subtyping. This changed in September 2004 however when Java was updated to version 5 (also referred to as Java SE 5), introducing the ability to use generics. Java generics extend the language with type parameters and effectively introduced parametric polymorphism to the language. One of the developers behind this update was Philip Wadler, a computer scientist who helped develop Haskell back in 1998 (Marlow, Haskell 2010 Language Report, 2010), His knowledge on functional programming allowed him to incorporate similar ideas into Java, resulting in Java SE 5 being labelled as "the most significant release" (Nayuki, 2017).

This is one example of how powerful solutions to software development problems from various different languages have influenced each other over time. Generics introduced strong type checking, which enabled errors and exceptions to be more likely to be caught during the compilation of code. Type parameters may be used as type variables in the declaration of fields and methods; thus, it is possible to utilise parametric polymorphic types in Java, as seen in Figure 9 located in the appendix.

```
...
        public static
        <A> A id(A a) { return a; }
...
```

***1.2 - Figure 1**: A polymorphic id in Java.*

```
class List<A> {
       public void
       add(A element) { ... }
}
```

***1.2 - Figure 2**: A polymorphic list in Java.*

1.2.2.3 Modularity

Modularity is the idea that any problem can be broken down into smaller segments in order to make it easier to solve. Imagine you need a new chair for your home office; it is much easier to put one together via smaller individual parts compared to carving out a new chair top to bottom from scratch. The same basic concept applies to programming. One such way this is done in both Java and Haskell is through the use of pre-made libraries, which can be used whilst developing new software without having to rewrite the same sections of code from scratch every time. Since computers can only execute one task at a time (provided there is only one CPU core), most basic programs are coded to simply run each line in order. This results in code being executed sequentially without switching between tasks. However, there are other ways that tasks can be processed in order to achieve a greater efficiency, provided that the particular language supports it.

1.2.2.3.1 Sequential

Imagine you are sitting in your new office chair at home when suddenly you get an urgent call. Your boss has asked you to prepare a presentation that you will deliver tomorrow in a critical meeting in another country. Unfortunately, your passport is out of date, requiring you to travel up to London to renew it. If you were to complete the tasks sequentially, you could first drive to London for an hour, wait in line for 3 hours, drive back for another hour, then work on the presentation for 5 hours.

1.2.2.3.2 Concurrency

On the other hand, you could drive up to London, then whilst you are waiting in line for 3 hours, you could work on the presentation. Once you have collected your passport and driven home, you would only need to spend 2 more hours on the presentation, as you switched between tasks whilst waiting in line with nothing else to do. This concept in software development is called concurrency.

Java was not originally built with concurrency in mind; it was only until Haskell influenced Java SE 5 update that support for it was added, via the use of multiple threads that work independently of each other. Whilst this has allowed for more efficient code as seen in Figure 10, developers have consequently been introduced to "more and more threading-related bug reports (from older Java projects) so rife with concurrency bugs that they work only by accident" (Goetz, Brian; Peierls, Timothy; Bloch, Joshua J.; Bowbeer, Joseph; Holmes, D.; Lea, D.;, 2006). Haskell, however, was built with concurrency in mind, allowing programs to make use of it since its initial release. This is due to the fact it was developed by academics and innovates, focusing on new and immerging ideas.

1.2.2.3.3 Parallelism

Alternatively, you can split a task between two people, such as asking your assistant to work on the important presentation whilst you simultaneously drive up and get your new passport. This is known as parallelism, which requires multiple processers. In a single core CPU, you may get concurrency but not parallelism. Similar to concurrency, Java was not originally developed with parallelism in mind, as the ability to share a task simply was not a well-established feature present in the industry.

Like concurrency, Haskell was developed with parallelism as a core feature from the beginning, as they could see the growing demand for the need of multiple CPU cores (Tardi, 2019). This decision resulted in the developers requiring vast research into systems that had never been developed before, such as "a new parallel garage collector" (Marlow, Haskell 2010 Language Report, 2010), that would work across a greater number of CPU cores. Since then, many other languages including Java have benefited from the vast amount of work that Haskell have put into developing software that fully utilise parallelism and concurrency, in order to achieve a greater efficiency.

Sequential	***Concurrency***	***Parallelism***
Task 1 -----	Task 1 - -	Task 1 -----
Task 2 -----	Task 2 --- --	Task 2 -----
Time: 10 Hours	Time: 7 Hours	Time: 5 Hours

1.2 - Figure 3*: A simplified visualisation of the modular concepts, matching the presentation scenario.*

1.2.3 Conclusion of Report 1.2

Both Java and Haskell have developed numerous ways of overcoming various problems that have arisen in software development overtime. As shown in the previous comparisons, it is clear that neither language is outright better than the other, though this does not mean that they have the same target audience. Java incorporates well established ideas into its language over time, with continued backwards compatibility and a large number of resources available to the public. This is great for enterprises and new enthusiasts alike; old code does not need to be continuously rewritten after every new update, whilst beginners can benefit from the vast libraries already available. Java would not be the language it is today though if it were not for functional programming languages such as Haskell, as many of the established features taken for granted in software development have come from academics continuously pushing the boundaries on what is thought to be possible. In order for languages like Java to remain powerful and popular among enterprises and enthusiasts, they must always be willing to incorporate ideas from innovative languages such as Haskell.

1.2.4 Appendix for Report 1.2

```
package genericsExample;

public class Drink {

	public static void main(String[] args) {

		Glass<Squash> g = new Glass<Squash>();
	Squash squash = new Squash();
g.liquid = squash;

		Squash s = g.liquid;

		Glass<Water> waterGlass = new Glass<Water>();
waterGlass.liquid = new Water();

		Water water = waterGlass.liquid;
	}
}
```

***1.2 - Figure 4**: Java Generics Example*

```
package concurrencyExample;

public class Main {

 Runnable runnable = () -> {	try

{
		String data = Thread.currentThread().getData();
		System.out.println("Ping. " + data);
		TimeUnit.SECONDS.sleep(1);
		System.out.println("Pong. " + data);
	}

	catch (InterruptedException e) {
e.printStackTrace();
```

```
            }
        };

        Thread thread = new Thread(runnable);
        thread.start();

}
```

***1.2 - Figure 5**: Java Concurrency Example*

```
data GenConfig = GenConfig
{ cfMsgChan :: TChan Msg
    -- ^ The channel connecting querying threads and the writing thread
  , cfRetrieved :: TVar Int
    -- ^ STM variable holding total number of entries retrieved
  , cfGenerated :: TVar Int
    -- ^ STM variable holding total number of entries produced
  , cfConnPool :: ConnectionPool
    -- ^ Database connection pool
  , cfQuery :: PriceRange -> Word -> IO (Either ServantError Listing)
    -- ^ Action that returns listing of items for given price range and offset
}
type Gen = ReaderT GenConfig IO

-- | Run the 'Gen' monad.
 runGen
  :: GenConfig          -- ^ Generation config
  -> Gen a              -- ^ The monad to run
  -> IO a
runGen cfg m = runReaderT m cfg

generate :: PostgresConf -> IO (Int, Int)
generate dbConfig = do   channel   <-
newTChanIO   retrieved <- newTVarIO 0
generated <- newTVarIO 0
  pool      <- createConnectionPool dbConfig
runGen GenConfig     { cfMsgChan   = channel
    , cfRetrieved = retrieved
    , cfGenerated = generated
    , cfConnPool  = pool
    , cfQuery     = undefined } $ void . mapConcurrently id $
csvWriter : (queryingAction 0 <$> priceRanges)   retrieved'
<- readTVarIO retrieved   generated' <- readTVarIO generated
return (retrieved', generated')

csvWriter :: Gen ()
queryingAction :: Word -> PriceRange -> Gen ()
```

***1.2 - Figure 6**: Haskell Concurrency Example – This code has been sampled from a university lecture.*

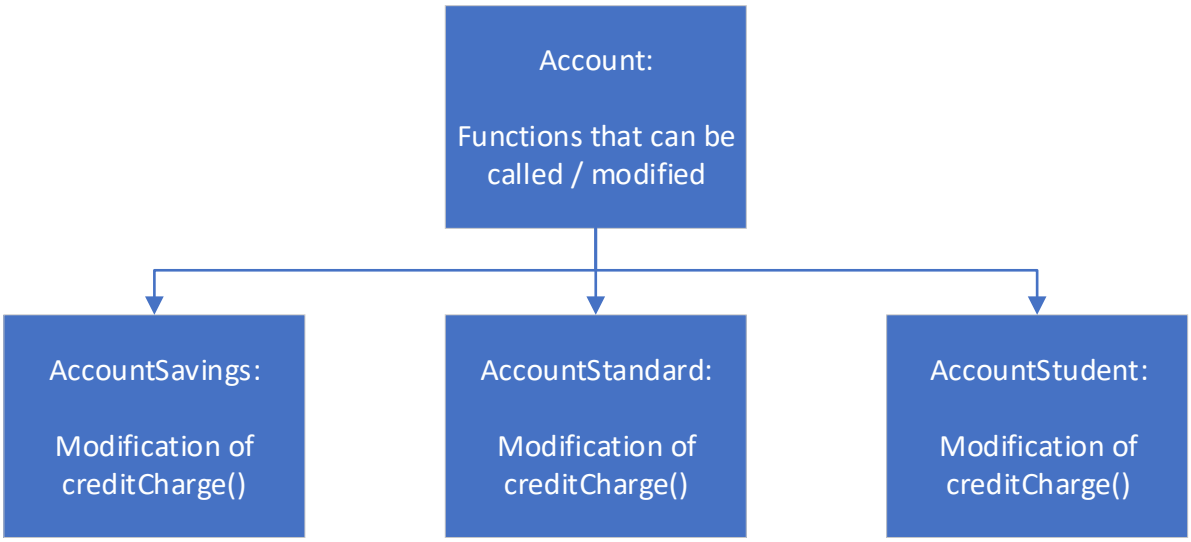
1.3 Inheritance in Coding

1.3.1 Understanding Inheritance

Inheritance is the mechanism of allowing an object or class to derive the features of another object or class. This allows for classes to be built upon those already existing, allowing the reuse of code, via public classes and interfaces for new objects to be created from already existing ones (Lemay, 1996). Savings, Standard and Student Accounts could all be classes that extend from a main Account class, allowing functions to be duplicated or changed if needed. This is not to be confused with subtyping, is also often known as interface inheritance, whilst inheritance is widely known as code inheritance or implementation inheritance (Tempero, Yang, & Noble, 2013). Inheritance as defined here is a commonly used mechanism for establishing subtype relationships (Hartley, 2003).

Inheritance is done by creating a superclass (also referred to as a base class or parent class), allowing extended subclasses (also referred to as an extended class or child class) to acquire pre-programmed properties and functions (JavaTPoint, 2018). A class consists of a group of objects that contain common properties and is a template from which objects can be created. Subclasses can inherit methods simply as they are, allowing the reuse of code, or write a new instance method that has the same signature as the one in its respective superclass. This allows the new instance to override the original method when called. Furthermore, subclasses can hide methods by writing a new static method that contains the same signature as the one in its respective superclass.

Every class created in Java has one direct superclass by default (expect the Object class), which implicitly is a subclass of Object unless otherwise stated (Oracle, 2019). A superclass can have any number of subclasses extending off of it, but a subclass can only have one superclass (unless interfaces are used, as will be seen later in this report). A subclass inherits all of the fields, methods, and nested classes from the superclass it extends from. Private members from a superclass are not inherited; however, these can be used by the subclass if the superclass has protected or public methods in order to access private fields. An example of inheritance can be seen in **Figure 1** below.



***1.3 - Figure 1**: One possible example of inheritance that can be utilised when creating classes in Java.*

In the previous example, there are three derived subclasses called *AccountSavings*, *AccountStandard* and *AccountStudent*, all of which extend the superclass Account. This allows each derived subclass to contain a copy of all the methods and felids functions from Account. All three of these subclasses, however, have separate *creditCharge()* functions. This can be seen in **Figure 2**, where *creditCharge()* is created in the superclass Account. The subclass *AccountStudent* as seen in **Figure 3** then overrides this function by changing the amount of credit limit allowed before interest is charged.

```
1  class Account {
2
3      private double theBalance = 0.00;
4      private double theOverdraft = 0.00;
5
6      public void creditCharge() {
7          if (getBalance() < 0) {
8              if (getOverdraftLimit() > (getBalance() + (getBalance() * 0.00026116))) {
9                  setOverdraftLimit((getBalance() + (getBalance() * 0.00026116)));
10             }
11             withdraw(-(getBalance() * 0.00026116));
12         }
13         return;
14     }
15
16     public void deposit(final double money) {
17         assert money >= 0.00;
18         theBalance = theBalance + money;
19     }
20
21     public double getBalance() { return theBalance; }
22     public double getOverdraftLimit() { return theOverdraft; }
23     public void setOverdraftLimit(final double money) { theOverdraft = money; }
24
25     public double withdraw(final double money) {
26         assert money >= 0.00;
27         if (theBalance - money >= theOverdraft) {
28             theBalance = theBalance - money;
29             return money;
30         } else {
31             return 0.00;
32         }
33     }
34 }
```

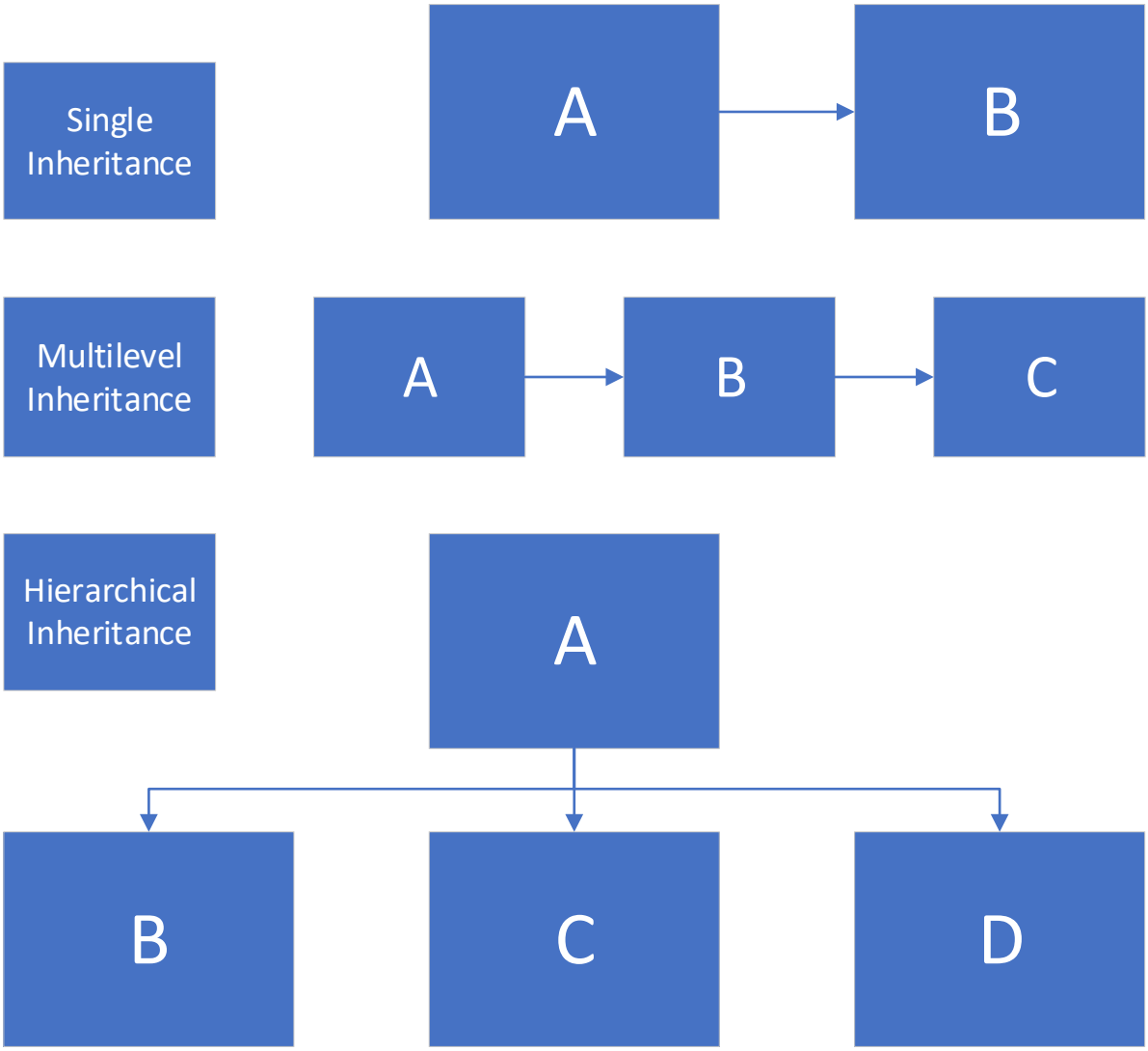
***1.3 - Figure 2**: A sample taken from the superclass Account, taken from **Section 2.2.1** of this booklet.*

```
1  class AccountStudent extends Account {
2
3      public void creditCharge() {
4
5          if (getBalance() < -5000) {
6              if (getOverdraftLimit() > (getBalance() + (getBalance() * 0.00026116))) {
7                  setOverdraftLimit((getBalance() + (getBalance() * 0.00026116)));
8              }
9              withdraw(-(getBalance() * 0.00026116));
10         }
11         return;
12     }
13 }
```

***1.3 - Figure 3**: Another sample of code, taken from subclass AccountStudent as seen in **Section 2.2.4**.*

1.3.2 Different Types of Inheritance

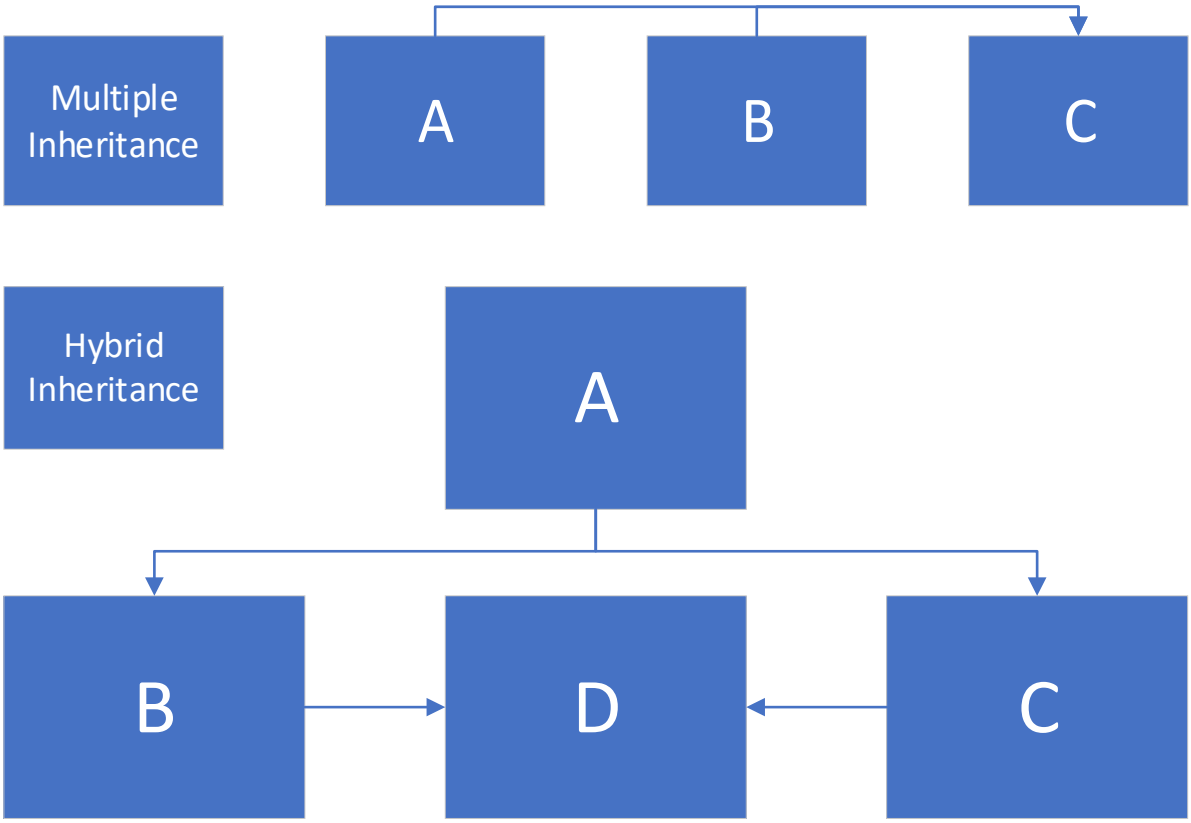
There are multiple types of inheritance, all of which contain a superclass and subclasses, but each comes with a variation of implementation (Gupta L. , 2019). The basic type is single inheritance, where the subclasses inherit features of the superclass it extends from. This is simplest type, where class A serves as a superclass for the derived subclass B. Another type is multilevel inheritance, where a derived class inherits a superclass as well as being the superclass for another subclass. In Java, a class cannot directly access its grandparent's content, so must do so via an intermediary class in-between. The last type is hierarchical inheritance. This is where one class is created to serve as a superclass for multiple subclasses, similar to the bank account example shown earlier.



***1.3 - Figure 4**: Three more examples of inheritance that can be utilised when creating classes in Java.*

Although languages such as Java do not currently support multiple types of inheritance used together with classes, it is possible via the use of interfaces. These are similar classes in the sense that it is a collection of abstract methods. Classes can implement an interface, which then in turn inherits all the abstract methods of it. Using this method, a class can inherit features from multiple interfaces. This can be seen in the example below, where class C is derived from interfaces A and B.

Similarly, a hybrid inheritance is a combination of inheritances used in conjunction with one another. Some languages like Java do not currently support multiple inheritance used together with classes as seen above; this can only be achieved through interfaces. An example can be seen in **Figure 5** below.



***1.3 - Figure 5**: Two different examples of inheritance that can only be used in Java through interfaces.*

1.3.3 Pros and Cons of Inheritance

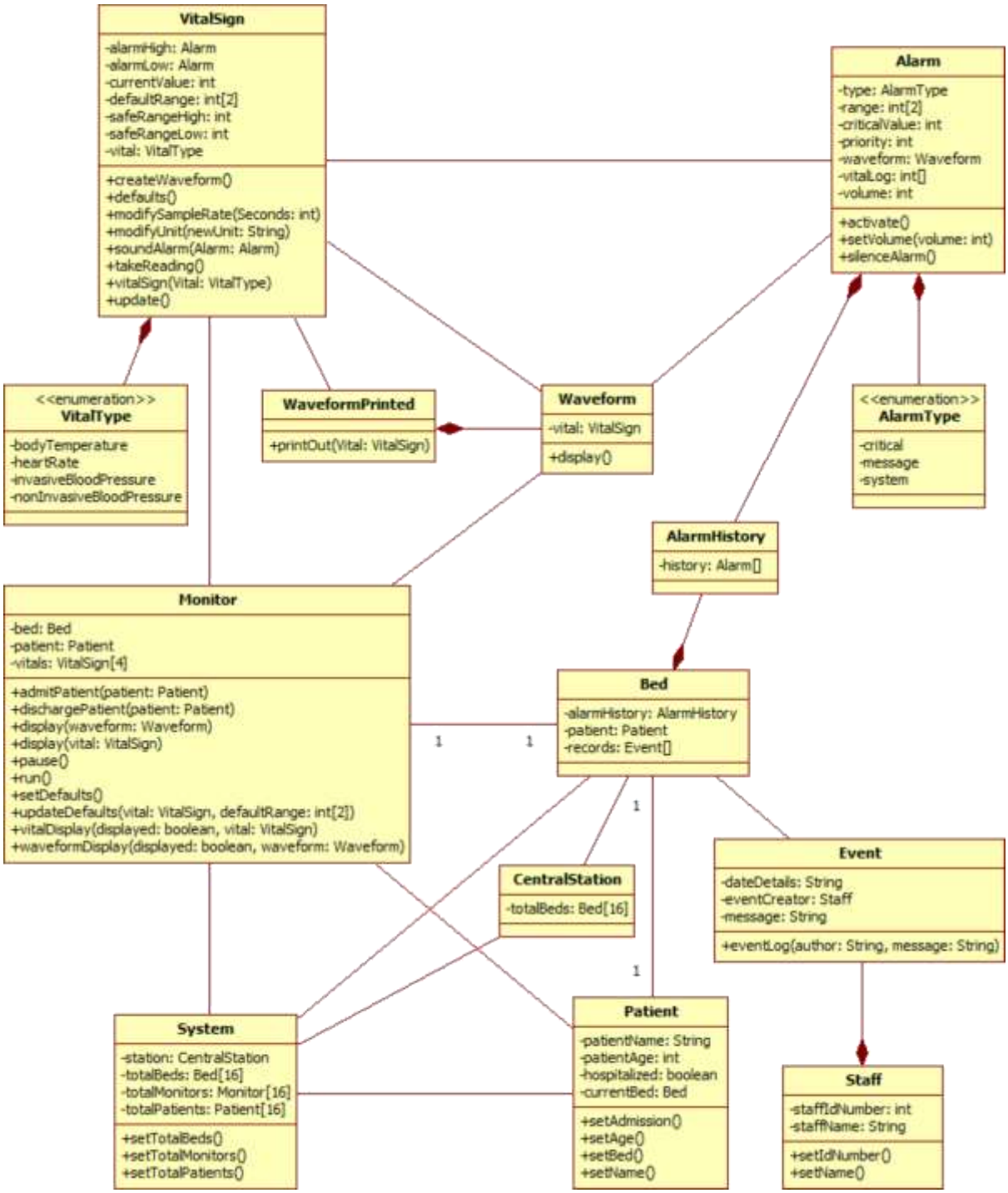
By reusing sections of already written code, we also avoid redundancy of code, as well as improve its overall reliability, as it will already have been tested and debugged. Furthermore, it provides a clear model structure which is easy to understand and manage. One such method this can be achieved through is the “Factory Method Pattern”, a design pattern that deals with the problem of objects being created without specifying the exact class that the object will be created. This is dealt with simply by creating objects via an interface and implementing them by subclasses, rather than by calling a constructer. Methods like these are vital when creating larger sized projects, such as video games that have multiple game modes, with each one following a standard set of rules.

Although there are many advantages, there are also some disadvantages with using inheritance. The main disadvantage is that a superclass and its subclasses get tightly coupled, meaning when code of the superclass is changed, it will affect all its derived subclasses as well. Although this can be useful when trying to change all the subclasses at once, it can also lead to certain variables being changed or overridden by mistake, leading to unexpected results. This is often referred to as the “Fragile Base Class Problem” (Yogen, 2018), where a programmer cannot determine whether a change to the superclass is safe simply by examining it in isolation without looking at the classes extending from it. In Java, this is handled by allowing inheritance or overriding of a class method to be prohibited, by labelling a declaration of a class or method with the keyword "final" (Aldrich, 2004).

1.4 Unified Modelling Language

1.4.1 UML Class Diagram Example

Figure 1 is a UML class diagram constructed to visualize an object-oriented for a hospital ward, with attributes and operations of each class, alongside the relationship between them. Each class below has a name at the top, attributes in the middle and operations or methods at the bottom.



***1.4 - Figure 1**: A UML class model, developed to match pre-set conditions for a predefined use case.*

1.4.2 Developing a Use Case Model

This task demonstrates requirements engineering to develop a use case model for a library to an agreed formal specification, through the use of formal methods and utilising discrete mathematics.

1.4.2.1 Membership Class (LM)

The model begins by defining an invariant for a membership, where members belong to a finite and initially empty set named *Member*. Each member identifier and associated information are specified as two fully abstract sets, $\mathbb{M}$ and $\mathbb{I}$. The initial condition for *LM*, together with its invariant, imply that all components are empty when first instantiated. A query is used to show information associated with a member, and another that lists them all. In both cases, their pre-conditions have been set to clarify these specifications by providing type information implicit from the invariant.

LM	
Queries and Events	Explanation
LM?showInfo(m → i)	Shows the information of a member.
LM?showMembers(→ M)	Shows all the registered members.
LM!NewMember(i → M)	Adds a new member to the system.
LM!UpdateInfo(m, i)	Updates the information of a member.
LM!RemoveMember(m)	Removes a member from the system.

1.4.2.2 Catalog Class (LT)

The library's catalog can be modelled as an initially empty set of titles, where each title has an associated description. This class has a specification that is very similar to the previous one, except this time modelling information around library titles as opposed to library members.

LT	
Queries and Events	Explanation
LT?showDesc(t → d)	Shows the description of a selected title.
LT?showCatalog(→ T)	Shows the catalog of all titles in the system.
LT!NewTitle(d → t)	Adds a new title into the system.
LT!UpdateDesc(t, d)	Updates the description of a title.
LT!RemoveTitle(t)	Removes a title from the system.

1.4.2.3 Collection Class (LC)

The next stage of the model creates a library collection by extending the previous catalog class. A function is created to record the current number of copies for every title in the system and define a partition over the set of such titles. Titles are considered to be in the in-collection if there is at least copy of the title available, otherwise they are in the ex-collection. The *showDesc* query is promoted directly from the previous class *LT*, as queries at this level may be similar to those provided for a library catalog. Other state-components and initial-conditions are also inherited.

The *showCatalog* query on the other hand is redefined in order to include not only information about library tiles, but also to include the number of copies as well as its description. The *NewTitle* event is also extended from *LT*, in order to fix the number of initial copies to 0. In order for the system to be able to change this value, a new event that adds or remove copies is included.

LC	
Queries and Events	Explanation
LC?showDesc(t → d)	Shows the description of a title.
LC?showNoCopies(t → n)	Shows the total number of copies of a title in the system.
LC?showInStockCollection(→ C)	Shows the titles currently in-collection.
LC?showOutStockCollection(→ C)	Shows the titles currently not in-collection.
LC!NewTitle(d, n → t)	Adds a new title into the system.
LC!AddOrRemoveCopies(t, n)	Alters the number of copies of a title.

1.4.2.4 Loan Class (LL)

A loan class is modelled by composing the previous classes *LM* and *LC*, where many of the queries and events at this level are simple promotions. These two classes are purposefully independent so that composing them will produce a consistent invariant. The model then extends their combined state, to define the set of all current loans, expressed mathematically in the model as a relation, and the numbers of available and loaned copies for each title in the system. A query is used to show the number of available copies for a given title, and another to update the *showCollection* query from *LC*. A query that shows the set of members borrowing copies of a given title is also provided.

The *NewTitle* event has been expanded to show that the number of available and loaned copies is initially zero. The *AddOrRemoveCopies* event is also promoted but expanded to show how the number of available copies is altered. It is also given a pre-condition; a removed copy must be an available copy. Finally, the expanded versions of the *LoanCopy* and *Return* events show how the numbers of available and loaned copies change.

LL	
Queries and Events	Explanation
LL?showInfo(m → i)	Shows the information of a member.
LL?showMembers(→ M)	Shows all the registered members.
LL?showDesc(t → d)	Shows the description of a title.
LL?showNoCopies(t → n)	Shows the total number of copies of a title in the system.
LL?availableCopies(t → n)	Shows the number of available title copies ready to be loaned.
LL?showInStockCollection(→ C)	Shows the titles currently in-collection.
LL?showOutStockCollection(→ C)	Shows the titles currently not in-collection.
LL?showLoans(t → M)	Shows the members currently borrowing copies of a title.
LL!NewMember(i → m)	Adds a new member to the system.
LL!UpdateInfo(m → i)	Updates the information of a member.
LL!NewTitle(d → t)	Adds a new title into the system.
LL!AddOrRemoveCopies(t, n)	Alters the number of copies of a title.
LL!LoanCopy(t, m)	Allows members to make loans of a title.
LL!Return(t, m)	Allows members to return loaned titles.

1.4.2.5 Reservation Class (LR)

LR is specified as a class parallel to *LL* to allow titles to be reserved. Later in the model, *LL* and *LR* will be composed together to produce a library system that is capable of both loans and reservations. To specify *LR*, the class LM and LC are exported and composed. A request queue function is created that delivers the sequence of members currently requesting a copy of a particular tile. Similar to loans in *LL*, reservations are defined as a relation. A query is created to show this information. The *NewTitle* event is extended at this level, in order to create a new and initially empty request queue.

An event that allows member to reserve a particular title, as well as an event to cancel such a reservation have also been created. These two events preserve the relative ordering of all other pending requests whenever one for a title and member is either reserved or cancelled. They also give its new or previous position in the queue. When a member is removed, the system also makes sure that any of their reservations are also removed.

LR	
Queries and Events	Explanation
LR?showRequests(t → Q)	Shows the request queue for a given title.
LR!NewTitle(d → t)	Adds a new title into the system.
LR!Reserve(t, m → p)	Adds a member to the request queue for a given title.
LR!Cancel(t, m → p)	Removes a member from the request queue for a given title.

1.4.2.6 Simple Library Class (LS)

LL and *LR* are finally composed to produce a simple library class, with the majority of queries and events being direct promotions from previous classes. To maintain consistency, the model imposes a further constraint; for each title, a member may have at most one loan or reservation at one time.

LS	
Queries and Events	Explanation
LS?showInfo(m → i)	Shows the information of a member.
LS?showMembers(→ M)	Shows all the registered members.
LS?showDesc(t → d)	Shows the description of a title.
LS?showNoCopies(t → n)	Shows the number of copies of a particular title in the system.
LS?showLoans(t → M)	Shows the members currently borrowing copies of a title.
LS?showRequests(t → Q)	Shows the request queue for a given title.
LS!NewMember(i → m)	Adds a new member to the system.
LS!UpdateInfo(m → I)	Updates the information of a member.
LS!AddOrRemoveCopies(t, n)	Alters the number of copies of a title.
LS!NewTitle(d → t)	Adds a new title into the system.
LS!Reserve(t, m → p)	Adds a member to the request queue for a given title.
LS!Cancel(t, m → p)	Removes a member from the request queue for a given title.
LS!LoanCopy(t, m)	Allows members to make loans of a title.
LS!Return(t, m)	Allows members to return loaned titles.

1.4.3 LaTeX Formal Use Case Model

Membership Class: LM

LM

$Member : \text{set } \mathbb{M}$ $info : \text{Member} \rightarrow \mathbb{I}$
$Member' = \emptyset$

$LM?showInfo(m \rightarrow i)$

$i := info(m)$

$LM?showMembers(m \rightarrow i)$

$M := info$

$LM!NewMember(i \rightarrow m)$

$i : \mathbb{I};\ m : \mathbb{M};\ m \notin Member$
$m \in Member';\ info'(m) = i$

$LM!UpdateInfo(m, i)$

$m : Member;\ i : \mathbb{I};\ i \neq info(m)$
$info'(m) = i$

$LM!RemoveMember(m)$

$m \in Member$
$m \notin Member$

Catalog Class: LT

LT

$Title : \text{set } \mathbb{T}$
$desc : Title \to \mathbb{D}$

$Title' = \emptyset$

$LT?showDesc(t \to d)$

$d := desc(t)$

$LT?showCatalog(\to T)$

$T := desc$

$LT!NewTitle(d \to t)$

$d : \mathbb{D};\ t : \mathbb{T};\ t \notin Title$

$t \in Title';\ desc'(t) = d$

$LT!UpdateDesc(t, d)$

$t \in Title$

$desc'(t) = d$

$LT!RemoveTitle(t)$

$t \in Title$

$t \notin Title$

Collection Class: LC

LC

LT
$nc : Title \to \mathsf{NAT}$
$\{InColl, ExColl\} : \mathsf{part}\ Title$
$InColl := \{t : Title \bullet nc(t) > 0\}$

$LC?showDesc(t \to d)$

$LT?showDesc(t \to d)$

$LC?showNoCopies(t \to n)$

$n := nc(t)$

$LC?showInCollection(\to C)$

$C : Title \pfun \mathsf{POS} \times \mathbb{D}$
$C = \{(t, n, d) : InColl \times \mathsf{POS} \times \mathbb{D} \bullet n = nc(t) \land d = desc(t)\}$

$LC?showOutCollection(\to C)$

$C : Title \pfun \mathbb{D}$
$C = \{(t, d) : ExColl \times \mathbb{D} \bullet d = desc(t)\}$

$LC!NewTitle(d, n \to t)$

$LT!NewTitle(d \to t)$

$nc'(t) = n$

$LC!AddOrRemoveCopies(t, n)$

$t : Title\ ;\ n : \mathsf{INT}\ ;\ nc(t) + n \geq 0$

$nc'(t) = nc(t) + n$

Loan Class: LL

LL

$$
\begin{array}{|l}
\hline
LM \; ; \; LC \\
loan : Title \nrightarrow Member \\
na, nl : Title \rightarrow \mathsf{NAT} \\
\forall t : Title \bullet \\
\quad nc(t) = na(t) + nl(t) \land \\
\quad nl(t) = \#\{m : Member \bullet t \mapsto m \in loan\} \\
\hline
\end{array}
$$

$LL?showInfo(m \rightarrow i)$

$$
\begin{array}{|l}
\hline
LM?showInfo(m \rightarrow i) \\
\hline
\end{array}
$$

$LL?showMembers(\rightarrow M)$

$$
\begin{array}{|l}
\hline
LM?showMembers(\rightarrow M) \\
\hline
\end{array}
$$

$LL?showDesc(t \rightarrow d)$

$$
\begin{array}{|l}
\hline
LC?showDesc(t \rightarrow d) \\
\hline
\end{array}
$$

$LL?showNoCopies(t \rightarrow n)$

$$
\begin{array}{|l}
\hline
LC?showMembers(\rightarrow M) \\
\hline
\end{array}
$$

$LL?availableCopies(t \rightarrow n)$

$$
\begin{array}{|l}
\hline
n := na(t) \\
\hline
\end{array}
$$

$LL?showInCollection(\rightarrow C)$

$$
\begin{array}{|l}
\hline
C : Title \pfun \mathsf{POS} \times \mathsf{NAT} \times \mathbb{D} \\
C = \{(t, n, l, d) : InColl \times \mathsf{POS} \times \mathsf{NAT} \times \mathbb{D} \bullet \\
\qquad n = nc(t) \land l = nl(t) \land d = desc(t)\} \\
\hline
\end{array}
$$

$LL?showOutCollection(\rightarrow C)$

$$
\begin{array}{|l}
\hline
C : Title \pfun \mathbb{D} \\
C = \{(t, n, l, d) : ExColl \times \mathbb{D} \bullet l = nl(t) \land d = desc(t)\} \\
\hline
\end{array}
$$

$LL?showLoans(t \rightarrow M)$

$$
\begin{array}{|l}
\hline
M := \{m : Member \bullet t \pfun m \in loan\} \\
\hline
\end{array}
$$

$LL!NewMember(i \rightarrow m)$

$$
\begin{array}{|l}
\hline
LM!NewMember(i \rightarrow m) \\
\hline
\end{array}
$$

$LL?UpdateInfo(m, i)$

$LM!UpdateInfo(m, i)$

$LL!NewTitle(d, n \to t)$

$LC?NewTitle(d, n \to t)$

$LL!AddOrRemoveCopies(t, n)$

$LC!AddOrRemoveCopies(t, n)$

$LL!LoanCopy(t, m, n)$

$t : Title\ ;\ m : Member\ ;\ n : nl(t$
$t \mapsto m \notin loan$
$na(t) > 0$
$nl(t) > 0$

$t \mapsto m \in loan'$

$LL!Return(t, m)$

$t \mapsto m : loan$

$t \mapsto m \notin loan'$

$LL?memberLoaning(m \to T)$

$T := t : Title \bullet m \mapsto t \in loan$

$LL!RemoveMember(m)$

$m \mapsto t \notin loan$

$LM!RemoveMember(m)$

$LL!RemoveTitle(t)$

$t \mapsto m \notin loan$

$LT!RemoveTitle(t)$

Reservation Class: LR

LR

LM ; LC

$requestQ : Title \to int \cdot \mathsf{seq}\ Member$

$reserve : Title \nrightarrow Member$

$\mathsf{cf}\ reserve = \mathsf{cod} \circ requestQ$

$nQ := (\#) \circ requestQ$

$LR!NewTitle(d, n \to t)$

$LC?NewTitle(d, n \to t)$

$requestQ'(t) = \langle\rangle$

$LR!Reserve(t, m \to p)$

$t : Title$; $m : Member$; $p : \mathsf{POS}$

$t \mapsto m \notin reserve$; $p = nQ(T) + 1$

$requestQ'(t) = (requestQ(t))\langle m\rangle$

$nQ'(t) = p$

$t \mapsto m \in reserve'$

$LR!Cancel(t, m \to p)$

$t : Title$; $m : Member$; $p : \mathsf{POS}$

$t \mapsto m \in reserve$

$Q_1\langle m\rangle Q_2 := requestQ(t)$

$p = \#Q_1 + 1$

$requestQ'(t) = Q_1\langle\rangle Q_2$

$nQ'(t) = nQ(t) - 1$

$t \mapsto m \notin reserve'$

$LR!RemoveMember(m)$

$m \mapsto t \notin reserve$

$LM!RemoveMember(m)$

$LR!RemoveTitle(t \to m)$

$LR!Cancel(t, m \to p)$

$LT!RemoveTitle(t)$

Simple Library Class: LS

LS

$$LL\,;\ LR$$
$$loan \cap reserve = \emptyset$$

$LS?showInfo(m \rightarrow i)$

$$LM?showInfo(m \rightarrow i)$$

$LS?showMembers(\rightarrow M)$

$$LM?showMembers(\rightarrow M)$$

$LS?showDesc(t \rightarrow d)$

$$LT?showDesc(t \rightarrow d)$$

$LS?showNoCopies(t \rightarrow n)$

$$LC?showMembers(\rightarrow M)$$

$LS?showNoCopies(t \rightarrow n)$

$$LC?showMembers(\rightarrow M)$$

$LS?showLoans(t \rightarrow M)$

$$LL?showLoans(t \rightarrow M)$$

$LS?showRequests(t \rightarrow Q)$

$$LR?showRequests(t \rightarrow Q)$$

$LS?showInCollection(\rightarrow C)$

$$C : Title \pfun \mathsf{POS} \times \mathsf{NAT} \times \mathsf{NAT} \times \mathbb{D}$$
$$C = \{(t, n, l, q, d) : InColl \times \mathsf{POS} \times \mathsf{NAT} \times \mathsf{NAT} \times \mathbb{D} \bullet$$
$$n = nc(t) \land l = nl(t) \land q = nQ(t) \land d = desc(t)\}$$

$LS?showOutCollection(\rightarrow C)$

$$C : Title \pfun \mathbb{D}$$
$$C = \{(t, n, l, q, d) : ExColl \times \mathbb{D} \bullet l = nl(t) \land q = nQ(t) \land d = desc(t)\}$$

$LS!NewMember(i \to m)$

$LM!NewMember(i \to m)$

$LS!RemoveMember(m)$

$LR!RemoveMember(m)$

$LS?UpdateInfo(m, i)$

$LM!UpdateInfo(m, i)$

$LS!AddOrRemoveCopies(t, n)$

$LL!AddOrRemoveCopies(t, n)$

$LS!NewTitle(d, n \to t)$

$LL!NewTitle(d, n \to t)$
$LR!NewTitle(d, n \to t)$

$LS!RemoveTitle(t \to m)$

$LR!RemoveTitle(t \to m)$

$LS!Reserve(t, m \to p)$

$LR!Reserve(t, m \to p)$
$t \mapsto m \notin loan$

$LS!Cancel(t, m \to p)$

$LR!Cancel(t, m \to p)$

$LS!LoanCopy(t, m)$

$LL!LoanCopy(t, m)$
$n : (NAT)\,;\ na(t) > n$

$t \mapsto m \notin reserve\,;\ n = nQ(t)$

$LR!Cancel(t, m \to p)$
$n = p - 1$

$LS!Return(t, m)$

$LL!Return(t, m)$

1.4.4 Evaluation of Report 1.4

The formal model that I have developed allows users members and their relevant information to be added, modified, or removed from a library system. The system can also add, modify, or remove any book alongside its relevant information. The user can check to see how many of any given book title is currently in stock. The system also allows members to loan or reserve library books through the use of queues. All of these functions are all split up into multiple classes to allow for easy promotion between them when required. Each of the models queries and events are then unified into one class at the end of the model for ease of use for any user of the system. All of this has been completed in order to meet the requirements for creating a formal use case model for a library lending system.

1.4.5 Reflection of Report 1.4

Upon reflection of my work, I feel that I have improved my understanding of the concepts and constraints of the provided formal model. I was able to achieve this by implementing various queries and events in order to provide required features for the model to function. I also learned how to use LaTeX in order to produce the layout of the use case, as seen in **Appendix A**. If I were to attempt a task similar to this one again in the future, I would research further into how dates can be used to implement other features, such as titles being listed as lost if unreturned after a predefined period. Overall, I am happy with the new modelling techniques I have developed, by formalizing system requirements and developing a formal model of an example library system at an abstract level.

1.4.6 Appendix for Report 1.4

Appendix A –LaTeX Code for Creating the Formal Use Case Model

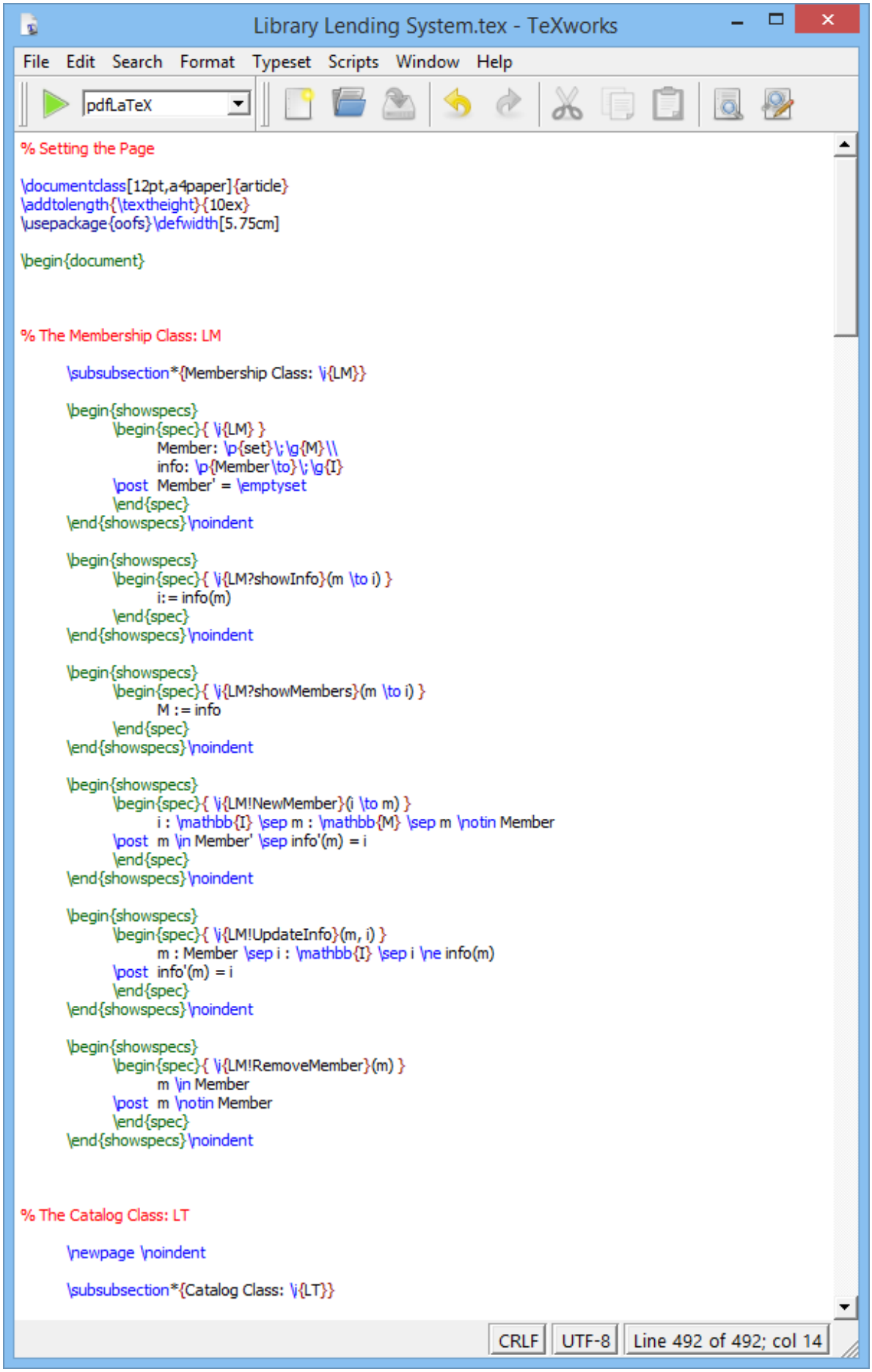
Library Lending System.tex - TeXworks

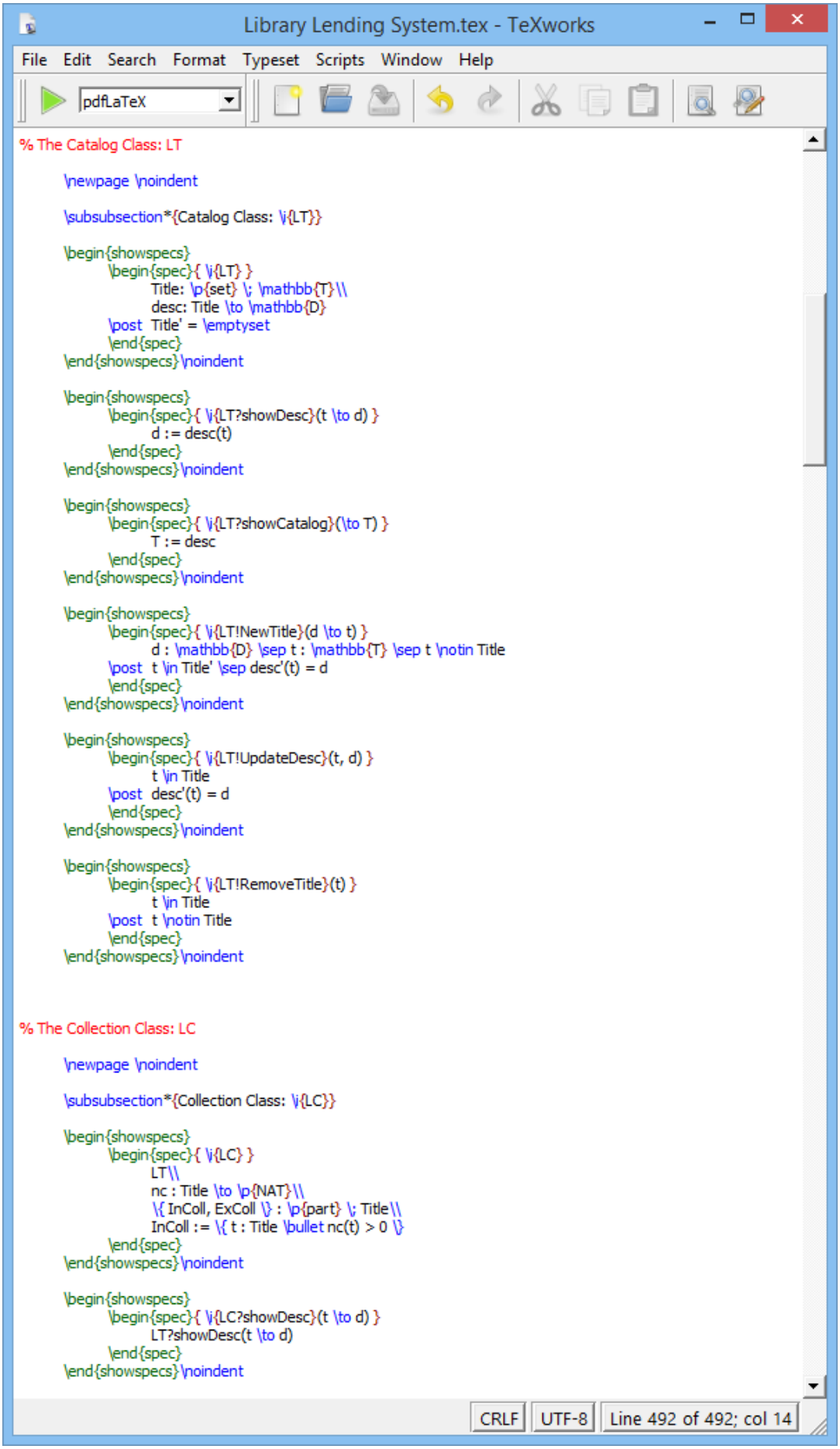
File Edit Search Format Typeset Scripts Window Help

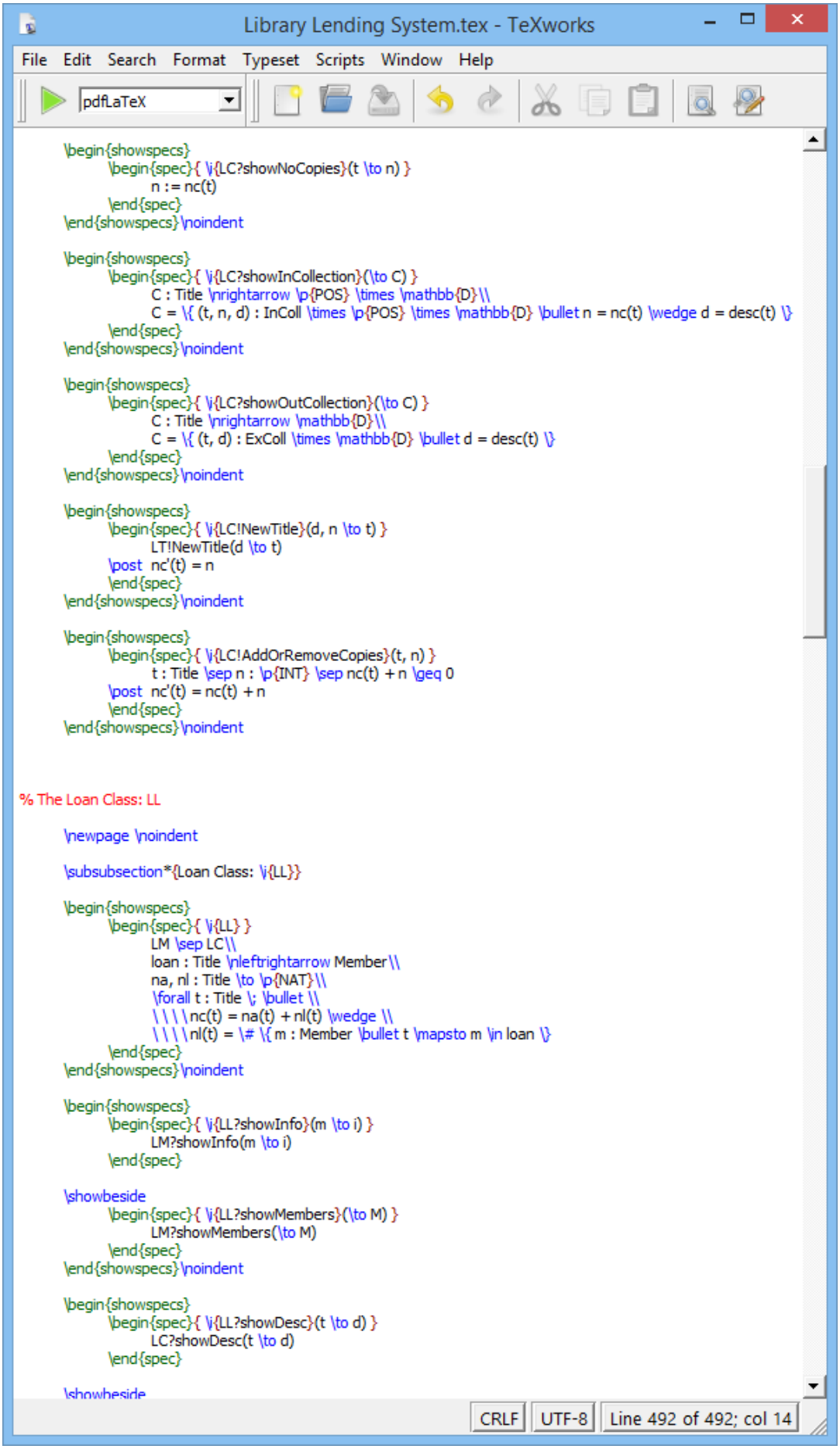
pdfLaTeX

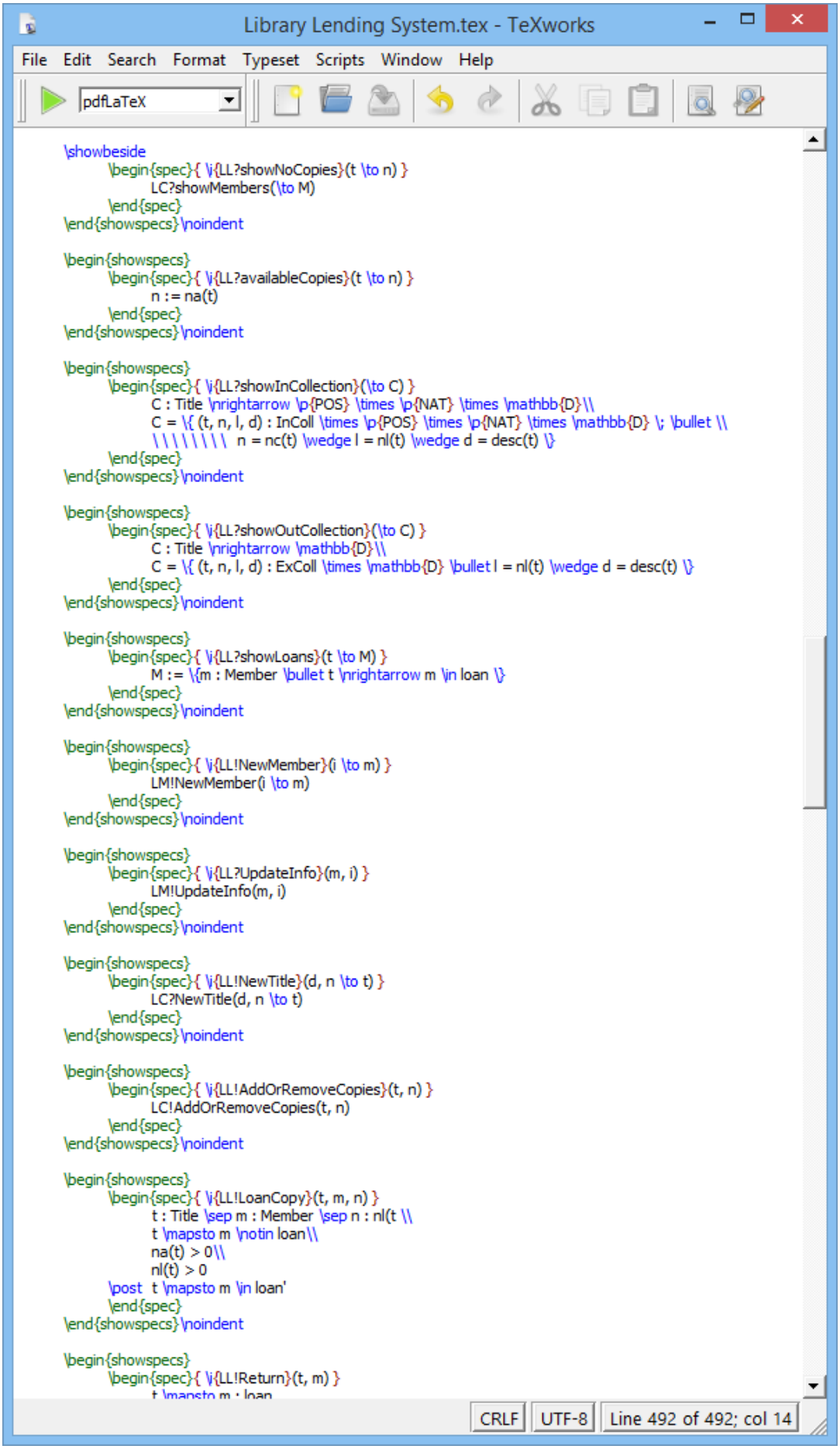
```
% Setting the Page

\documentclass[12pt,a4paper]{article}
\addtolength{\textheight}{10ex}
\usepackage{oofs}\defwidth[5.75cm]

\begin{document}



% The Membership Class: LM

        \subsubsection*{Membership Class: \i{LM}}

        \begin{showspecs}
                \begin{spec}{ \i{LM} }
                        Member: \p{set}\;\g{M}\\
                        info: \p{Member \to}\;\g{I}
                \post  Member' = \emptyset
                \end{spec}
        \end{showspecs}\noindent

        \begin{showspecs}
                \begin{spec}{ \i{LM?showInfo}(m \to i) }
                        i:= info(m)
                \end{spec}
        \end{showspecs}\noindent

        \begin{showspecs}
                \begin{spec}{ \i{LM?showMembers}(m \to i) }
                        M := info
                \end{spec}
        \end{showspecs}\noindent

        \begin{showspecs}
                \begin{spec}{ \i{LM!NewMember}(i \to m) }
                        i : \mathbb{I} \sep m : \mathbb{M} \sep m \notin Member
                \post  m \in Member' \sep info'(m) = i
                \end{spec}
        \end{showspecs}\noindent

        \begin{showspecs}
                \begin{spec}{ \i{LM!UpdateInfo}(m, i) }
                        m : Member \sep i : \mathbb{I} \sep i \ne info(m)
                \post  info'(m) = i
                \end{spec}
        \end{showspecs}\noindent

        \begin{showspecs}
                \begin{spec}{ \i{LM!RemoveMember}(m) }
                        m \in Member
                \post  m \notin Member
                \end{spec}
        \end{showspecs}\noindent
```

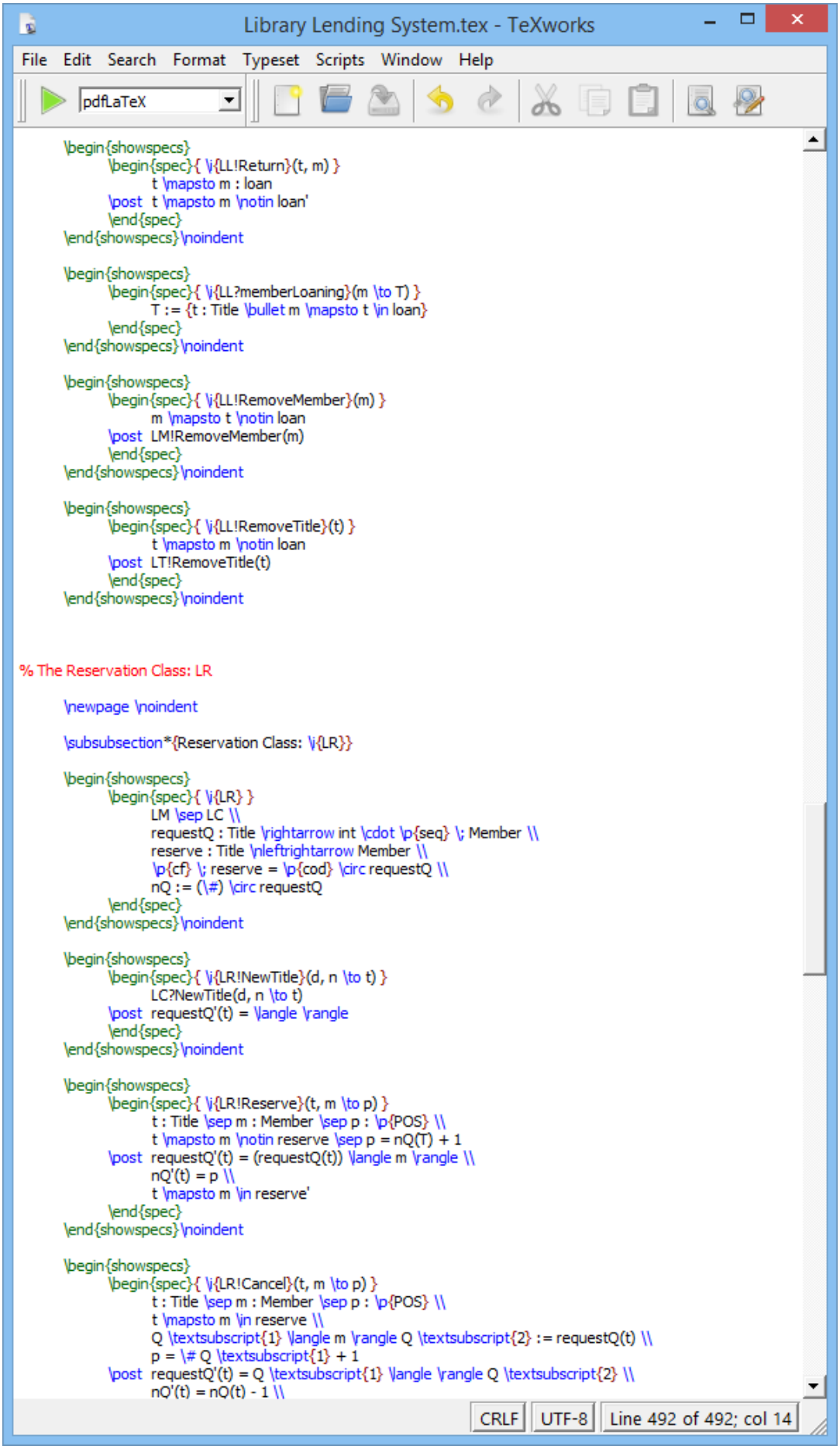
CRLF UTF-8 Line 492 of 492; col 14

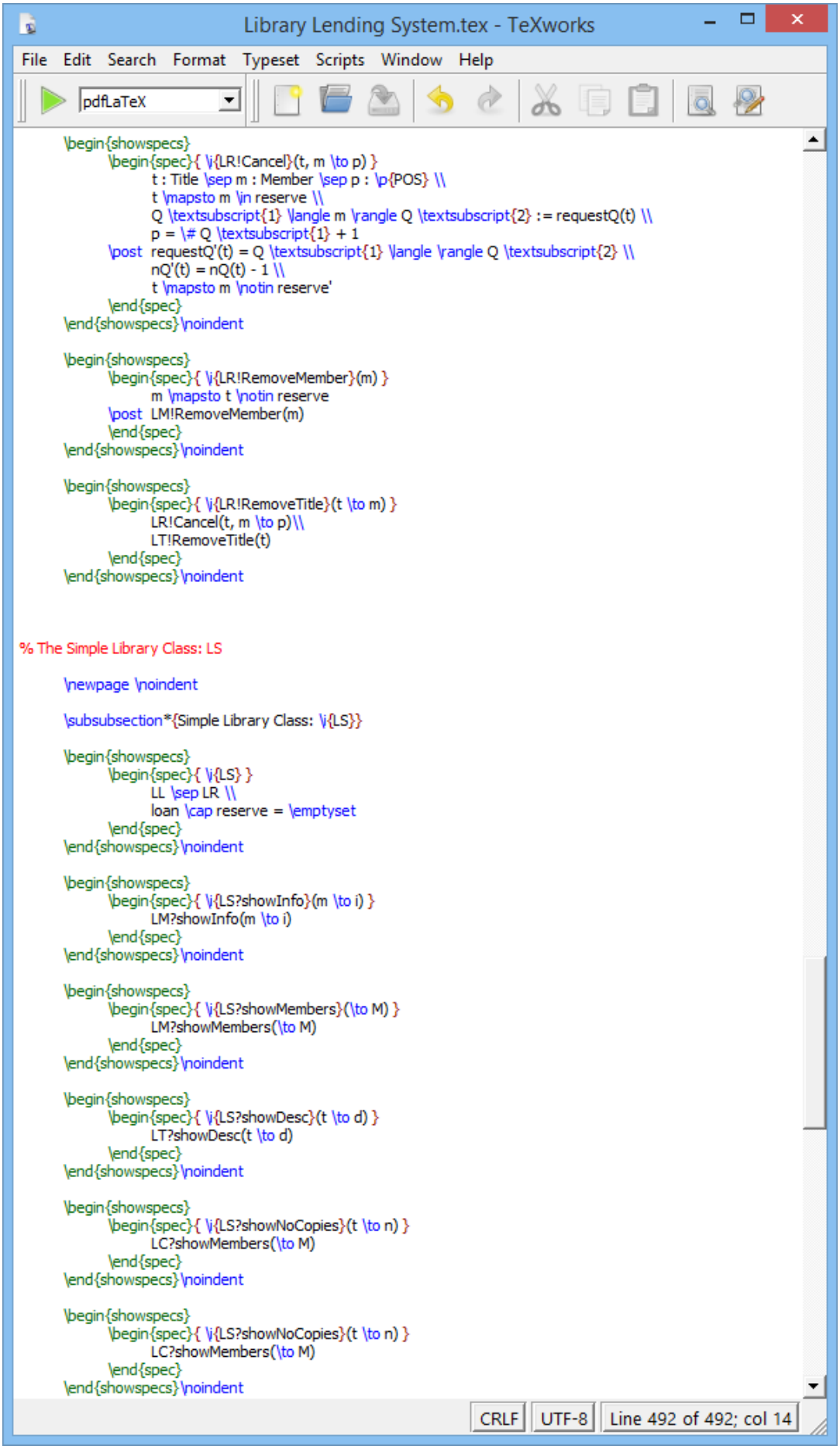

Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
% Setting the Page
\documentclass[12pt,a4paper]{article}
\addtolength{\textheight}{10ex}
\usepackage{oofs}\defwidth[5.75cm]
\begin{document}
% The Membership Class: LM
\subsubsection*{Membership Class: \j{LM}}
\begin{showspecs}
\begin{spec}{ \j{LM} }
Member: \p{set}\;\g{M}\\
info: \p{Member\to}\;\g{I}
\post Member' = \emptyset
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \j{LM?showInfo}(m \to i) }
i:= info(m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \j{LM?showMembers}(m \to i) }
M := info
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \j{LM!NewMember}(i \to m) }
i : \mathbb{I} \sep m : \mathbb{M} \sep m \notin Member
\post m \in Member' \sep info'(m) = i
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \j{LM!UpdateInfo}(m, i) }
m : Member \sep i : \mathbb{I} \sep i \ne info(m)
\post info'(m) = i
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \j{LM!RemoveMember}(m) }
m \in Member
\post m \notin Member
\end{spec}
\end{showspecs}\noindent
% The Catalog Class: LT
\newpage \noindent
\subsubsection*{Catalog Class: \j{LT}}
CRLF
UTF-8
Line 492 of 492; col 14

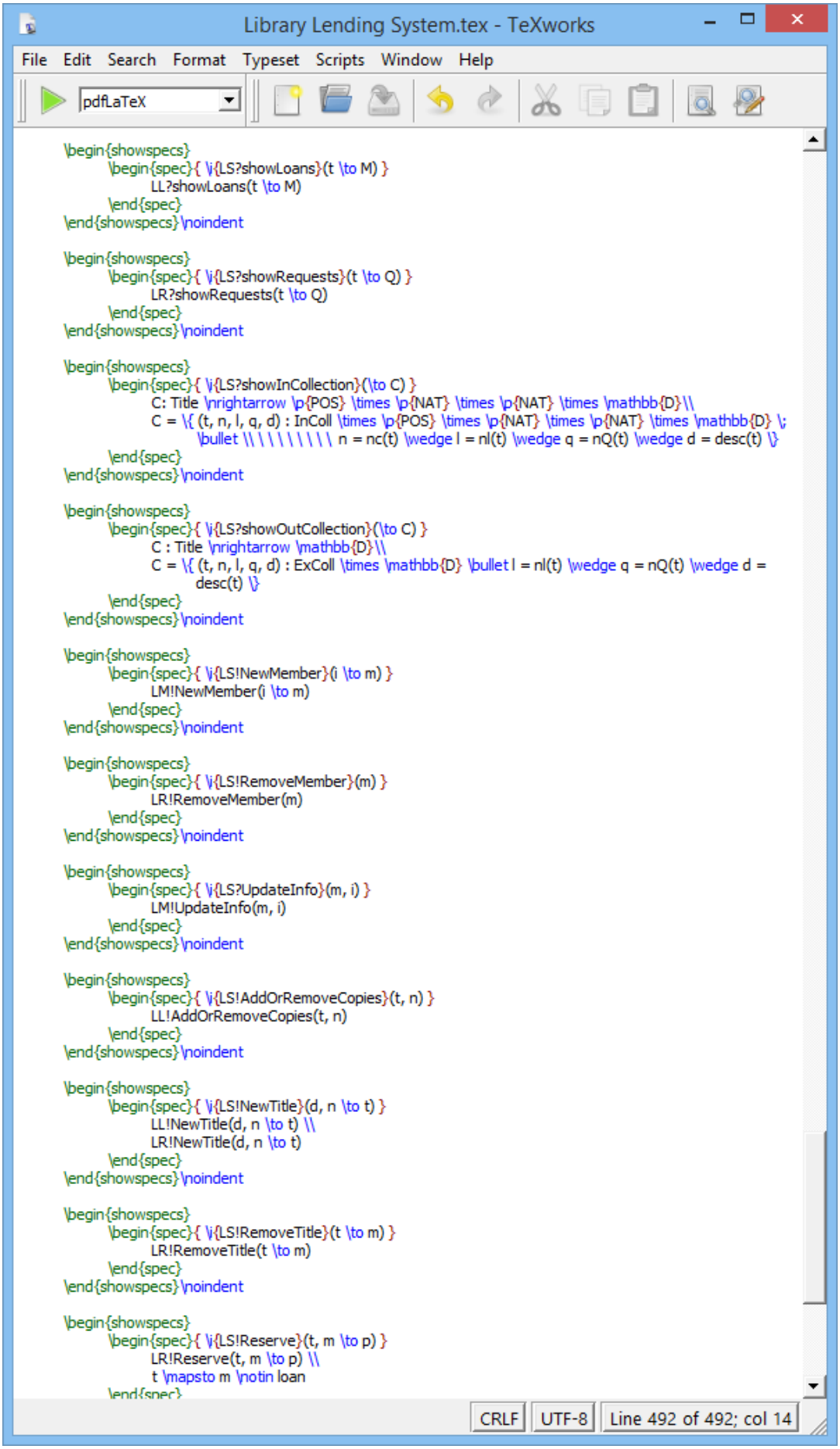

Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
% The Catalog Class: LT
\newpage \noindent
\subsubsection*{Catalog Class: \i{LT}}
\begin{showspecs}
\begin{spec}{ \i{LT} }
Title: \p{set} \; \mathbb{T}\\
desc: Title \to \mathbb{D}
\post Title' = \emptyset
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LT?showDesc}(t \to d) }
d := desc(t)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LT?showCatalog}(\to T) }
T := desc
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LT!NewTitle}(d \to t) }
d : \mathbb{D} \sep t : \mathbb{T} \sep t \notin Title
\post t \in Title' \sep desc'(t) = d
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LT!UpdateDesc}(t, d) }
t \in Title
\post desc'(t) = d
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LT!RemoveTitle}(t) }
t \in Title
\post t \notin Title
\end{spec}
\end{showspecs}\noindent
% The Collection Class: LC
\newpage \noindent
\subsubsection*{Collection Class: \i{LC}}
\begin{showspecs}
\begin{spec}{ \i{LC} }
LT\\
nc : Title \to \p{NAT}\\
\{ InColl, ExColl \} : \p{part} \; Title\\
InColl := \{ t : Title \bullet nc(t) > 0 \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LC?showDesc}(t \to d) }
LT?showDesc(t \to d)
\end{spec}
\end{showspecs}\noindent
CRLF
UTF-8
Line 492 of 492; col 14

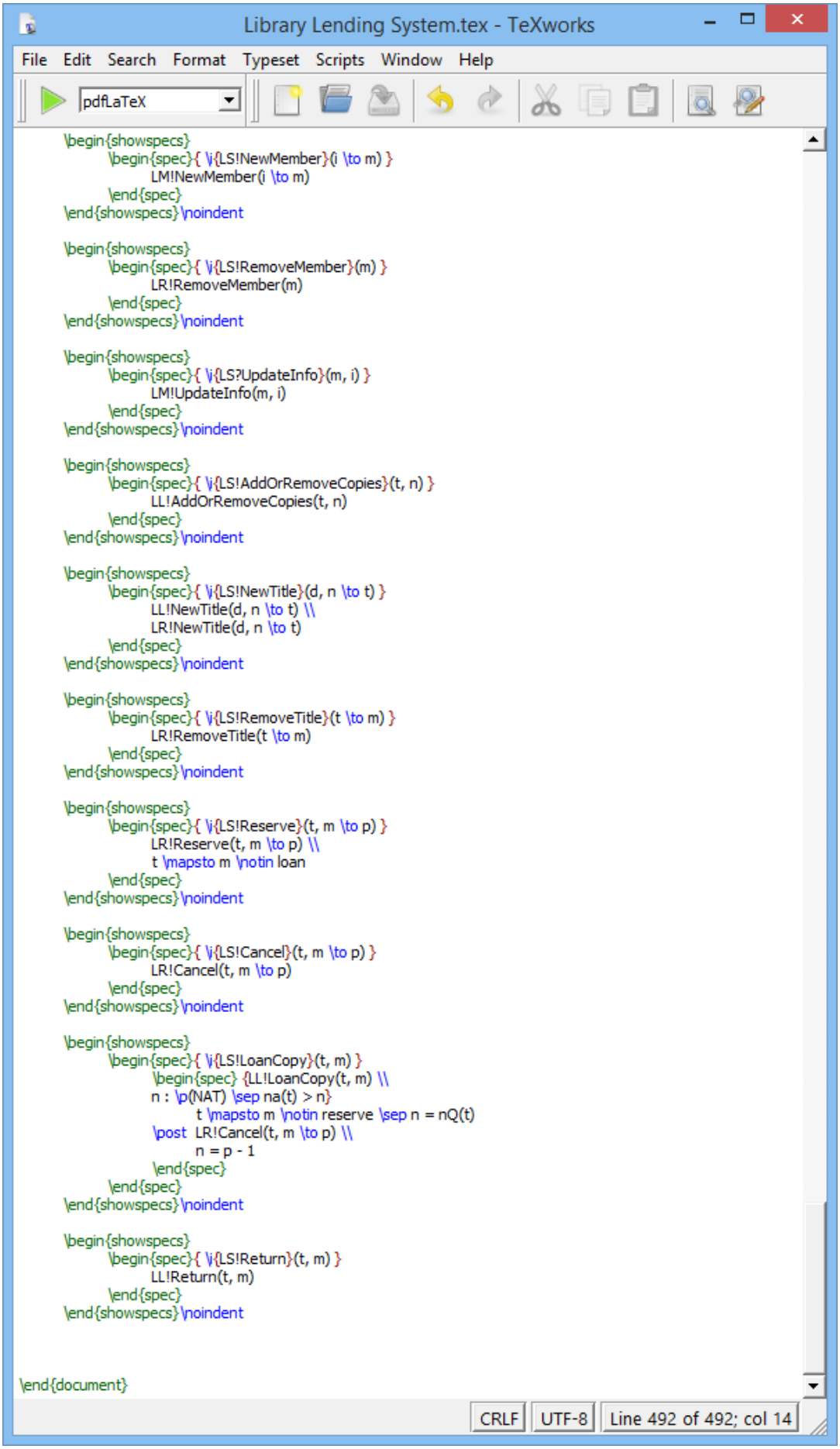

Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
\begin{showspecs}
\begin{spec}{ \i{LC?showNoCopies}(t \to n) }
n := nc(t)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LC?showInCollection}(\to C) }
C : Title \nrightarrow \p{POS} \times \mathbb{D}\\
C = \{ (t, n, d) : InColl \times \p{POS} \times \mathbb{D} \bullet n = nc(t) \wedge d = desc(t) \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LC?showOutCollection}(\to C) }
C : Title \nrightarrow \mathbb{D}\\
C = \{ (t, d) : ExColl \times \mathbb{D} \bullet d = desc(t) \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LC!NewTitle}(d, n \to t) }
LT!NewTitle(d \to t)
\post nc'(t) = n
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LC!AddOrRemoveCopies}(t, n) }
t : Title \sep n : \p{INT} \sep nc(t) + n \geq 0
\post nc'(t) = nc(t) + n
\end{spec}
\end{showspecs}\noindent
% The Loan Class: LL
\newpage \noindent
\subsubsection*{Loan Class: \i{LL}}
\begin{showspecs}
\begin{spec}{ \i{LL} }
LM \sep LC\\
loan : Title \nleftrightarrow Member\\
na, nl : Title \to \p{NAT}\\
\forall t : Title \; \bullet \\
\ \ \ \ nc(t) = na(t) + nl(t) \wedge \\
\ \ \ \ nl(t) = \# \{ m : Member \bullet t \mapsto m \in loan \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?showInfo}(m \to i) }
LM?showInfo(m \to i)
\end{spec}
\showbeside
\begin{spec}{ \i{LL?showMembers}(\to M) }
LM?showMembers(\to M)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?showDesc}(t \to d) }
LC?showDesc(t \to d)
\end{spec}
\showbeside
CRLF UTF-8 Line 492 of 492; col 14


Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
\showbeside
\begin{spec}{ \i{LL?showNoCopies}(t \to n) }
LC?showMembers(\to M)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?availableCopies}(t \to n) }
n := na(t)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?showInCollection}(\to C) }
C : Title \nrightarrow \p{POS} \times \p{NAT} \times \mathbb{D}\\
C = \{ (t, n, l, d) : InColl \times \p{POS} \times \p{NAT} \times \mathbb{D} \; \bullet \\
\ \ \ \ \ \ \ \ n = nc(t) \wedge l = nl(t) \wedge d = desc(t) \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?showOutCollection}(\to C) }
C : Title \nrightarrow \mathbb{D}\\
C = \{ (t, n, l, d) : ExColl \times \mathbb{D} \bullet l = nl(t) \wedge d = desc(t) \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?showLoans}(t \to M) }
M := \{m : Member \bullet t \nrightarrow m \in loan \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!NewMember}(i \to m) }
LM!NewMember(i \to m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?UpdateInfo}(m, i) }
LM!UpdateInfo(m, i)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!NewTitle}(d, n \to t) }
LC?NewTitle(d, n \to t)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!AddOrRemoveCopies}(t, n) }
LC!AddOrRemoveCopies(t, n)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!LoanCopy}(t, m, n) }
t : Title \sep m : Member \sep n : nl(t \\
t \mapsto m \notin loan\\
na(t) > 0\\
nl(t) > 0
\post t \mapsto m \in loan'
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!Return}(t, m) }
t \mapsto m : loan
CRLF
UTF-8
Line 492 of 492; col 14


Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
\begin{showspecs}
\begin{spec}{ \i{LL!Return}(t, m) }
t \mapsto m : loan
\post t \mapsto m \notin loan'
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL?memberLoaning}(m \to T) }
T := {t : Title \bullet m \mapsto t \in loan}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!RemoveMember}(m) }
m \mapsto t \notin loan
\post LM!RemoveMember(m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LL!RemoveTitle}(t) }
t \mapsto m \notin loan
\post LT!RemoveTitle(t)
\end{spec}
\end{showspecs}\noindent
% The Reservation Class: LR
\newpage \noindent
\subsubsection*{Reservation Class: \i{LR}}
\begin{showspecs}
\begin{spec}{ \i{LR} }
LM \sep LC \\
requestQ : Title \rightarrow int \cdot \p{seq} \; Member \\
reserve : Title \nleftrightarrow Member \\
\p{cf} \; reserve = \p{cod} \circ requestQ \\
nQ := (\#) \circ requestQ
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LR!NewTitle}(d, n \to t) }
LC?NewTitle(d, n \to t)
\post requestQ'(t) = \langle \rangle
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LR!Reserve}(t, m \to p) }
t : Title \sep m : Member \sep p : \p{POS} \\
t \mapsto m \notin reserve \sep p = nQ(T) + 1
\post requestQ'(t) = (requestQ(t)) \langle m \rangle \\
nQ'(t) = p \\
t \mapsto m \in reserve'
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LR!Cancel}(t, m \to p) }
t : Title \sep m : Member \sep p : \p{POS} \\
t \mapsto m \in reserve \\
Q \textsubscript{1} \langle m \rangle Q \textsubscript{2} := requestQ(t) \\
p = \# Q \textsubscript{1} + 1
\post requestQ'(t) = Q \textsubscript{1} \langle \rangle Q \textsubscript{2} \\
nQ'(t) = nQ(t) - 1 \\
CRLF
UTF-8
Line 492 of 492; col 14


Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
\begin{showspecs}
\begin{spec}{ \i{LR!Cancel}(t, m \to p) }
t : Title \sep m : Member \sep p : \p{POS} \\
t \mapsto m \in reserve \\
Q \textsubscript{1} \langle m \rangle Q \textsubscript{2} := requestQ(t) \\
p = \# Q \textsubscript{1} + 1
\post requestQ'(t) = Q \textsubscript{1} \langle \rangle Q \textsubscript{2} \\
nQ'(t) = nQ(t) - 1 \\
t \mapsto m \notin reserve'
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LR!RemoveMember}(m) }
m \mapsto t \notin reserve
\post LM!RemoveMember(m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LR!RemoveTitle}(t \to m) }
LR!Cancel(t, m \to p)\\
LT!RemoveTitle(t)
\end{spec}
\end{showspecs}\noindent
% The Simple Library Class: LS
\newpage \noindent
\subsubsection*{Simple Library Class: \i{LS}}
\begin{showspecs}
\begin{spec}{ \i{LS} }
LL \sep LR \\
loan \cap reserve = \emptyset
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showInfo}(m \to i) }
LM?showInfo(m \to i)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showMembers}(\to M) }
LM?showMembers(\to M)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showDesc}(t \to d) }
LT?showDesc(t \to d)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showNoCopies}(t \to n) }
LC?showMembers(\to M)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showNoCopies}(t \to n) }
LC?showMembers(\to M)
\end{spec}
\end{showspecs}\noindent
CRLF UTF-8 Line 492 of 492; col 14


Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
\begin{showspecs}
\begin{spec}{ \i{LS?showLoans}(t \to M) }
LL?showLoans(t \to M)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showRequests}(t \to Q) }
LR?showRequests(t \to Q)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showInCollection}(\to C) }
C: Title \nrightarrow \p{POS} \times \p{NAT} \times \p{NAT} \times \mathbb{D}\\
C = \{ (t, n, l, q, d) : InColl \times \p{POS} \times \p{NAT} \times \p{NAT} \times \mathbb{D} \;
\bullet \\ \\ \\ \\ \\ \\ \\ n = nc(t) \wedge l = nl(t) \wedge q = nQ(t) \wedge d = desc(t) \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?showOutCollection}(\to C) }
C : Title \nrightarrow \mathbb{D}\\
C = \{ (t, n, l, q, d) : ExColl \times \mathbb{D} \bullet l = nl(t) \wedge q = nQ(t) \wedge d =
desc(t) \}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!NewMember}(i \to m) }
LM!NewMember(i \to m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!RemoveMember}(m) }
LR!RemoveMember(m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?UpdateInfo}(m, i) }
LM!UpdateInfo(m, i)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!AddOrRemoveCopies}(t, n) }
LL!AddOrRemoveCopies(t, n)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!NewTitle}(d, n \to t) }
LL!NewTitle(d, n \to t) \\
LR!NewTitle(d, n \to t)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!RemoveTitle}(t \to m) }
LR!RemoveTitle(t \to m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!Reserve}(t, m \to p) }
LR!Reserve(t, m \to p) \\
t \mapsto m \notin loan
\end{spec}
CRLF
UTF-8
Line 492 of 492; col 14


Library Lending System.tex - TeXworks
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX
\begin{showspecs}
\begin{spec}{ \i{LS!NewMember}(i \to m) }
LM!NewMember(i \to m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!RemoveMember}(m) }
LR!RemoveMember(m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS?UpdateInfo}(m, i) }
LM!UpdateInfo(m, i)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!AddOrRemoveCopies}(t, n) }
LL!AddOrRemoveCopies(t, n)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!NewTitle}(d, n \to t) }
LL!NewTitle(d, n \to t) \\
LR!NewTitle(d, n \to t)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!RemoveTitle}(t \to m) }
LR!RemoveTitle(t \to m)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!Reserve}(t, m \to p) }
LR!Reserve(t, m \to p) \\
t \mapsto m \notin loan
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!Cancel}(t, m \to p) }
LR!Cancel(t, m \to p)
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!LoanCopy}(t, m) }
\begin{spec} {LL!LoanCopy(t, m) \\
n : \p(NAT) \sep na(t) > n}
t \mapsto m \notin reserve \sep n = nQ(t)
\post LR!Cancel(t, m \to p) \\
n = p - 1
\end{spec}
\end{spec}
\end{showspecs}\noindent
\begin{showspecs}
\begin{spec}{ \i{LS!Return}(t, m) }
LL!Return(t, m)
\end{spec}
\end{showspecs}\noindent
\end{document}
CRLF
UTF-8
Line 492 of 492; col 14

1.5 Mobile Application Design

Showcased below is a group project, where we were given the task to design and analyse a mobile application that could help not only students navigate in and around all of the campuses in the university, but also for any visitors who may visit throughout the year. After long discussions on the possible subjects which we could cover involving the subject we have decided as a group that the best way to achieve this would be to create and design an interactive map; the mobile app will be designed in order to assist both current and newly accepted students and visiting bodies to easily locate and manoeuvre through the buildings and campuses.

1.5.1 Required Research

To help us design our application we researched the key human computer interaction principles and rules. This included Don Norman's Principles and Schneiderman's Eight Golden Rules. Below are the two lists of principles that we have strived to follow:

1.5.1.1 Don Norman's Principles

Visibility - The more visible functions are, the more likely users will be able to know what to do next. When functions are "out of sight," it makes them more difficult to find and know how to use.

Feedback - Feedback is about sending back information about what action has been done and what has been accomplished, allowing the person to continue with the activity. Various kinds of feedback are available for interaction design-audio, tactile, verbal, and combinations of these.

Constraints - The design concept of constraining refers to determining ways of restricting the kind of user interaction that can take place at a given moment. There are various ways this can be achieved.

Mapping - The relationship between controls and their effects in the world. Nearly all artifacts need some kind of mapping between controls and effects, whether it is a flashlight, car, power plant, or cockpit. An example of a good mapping between control and effect is the up and down arrows used to represent the up and down movement of the cursor, respectively, on a computer keyboard.

Consistency - This refers to designing interfaces to have similar operations and use similar elements for achieving similar tasks. In particular, a consistent interface is one that follows rules, such as using the same operation to select all objects. For example, a consistent operation is using the same input action to highlight any graphical object at the interface, such as always clicking the left mouse button. Inconsistent interfaces, on the other hand, allow exceptions to a rule.

Affordance - An attribute of an object that allows people to know how to use it. For example, a mouse button invites pushing (in so doing acting clicking) by the way it is physically constrained in its plastic shell. At a very simple level, to afford means "to give a clue". When the affordances of a physical object are perceptually obvious it is easy to know how to interact with it.

1.5.1.2 Schneiderman's Golden Rules

Strive for Consistency - Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.

Enable Shortcuts - As the frequency of use increases, so do the user's desires to increase the pace and reduce the number of interactions. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

Informative Feedback - For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

Design Dialog for Closure - Sequences of actions should be organized into groups with a clear beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators a sense of relief, the satisfaction of accomplishment, the signal to drop contingencies from their minds, and an indication that it is clear to prepare for the next group of actions.

Simple Error Handling - As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.

Easy Reversal of Actions - This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

Internal Locus of Control - Experienced operators strongly desire the sense that they are in charge of the system and that it responds to their actions. Design the system to make users the initiators of actions rather than the responders.

Low Short-Term Memory Load - The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

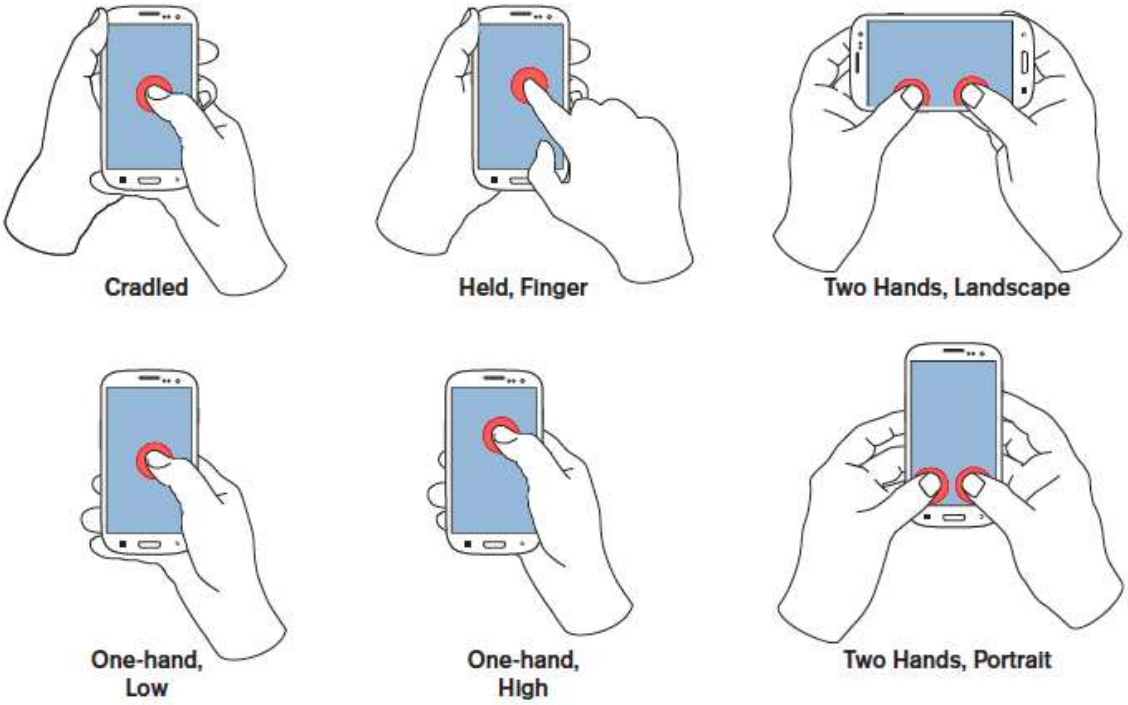
1.5.1.3 Applying the Principles

These overarching design principles of human computer interaction when applied will help to create a user interfaces that allow a user to:

- Avoid or easily solve errors. (Simple Solutions).
- Easily navigate with shortcuts and actions reversals. (Control, Shortcuts, Reversal).
- Intuitively understand how to interact with the application (Visibility, Affordance, Mapping).
- Inform the user when the application is interacted with and the result and know the action is complete (Feedback, Closure).

In addition to Don Norman’s principles and Schneiderman’s rules, the app design field also has many UX design and functionality principles and conventions unique to the field:

- Design screens for fingers.
- Give specific reason when asking for permissions.
- Inform users of controls and functions unique to our app.
- Reduce UI elements to the most fundamental to avoid clutter.
- Minimalist design to not distract from the key tasks of the user.
- Legible text and visible buttons, with deliberate colour and contrast choices.
- Minimize number of actions required, sticking to one primary action per screen.
- Taking advantage of unique functionality (working offline, notifications, GPS, gyroscope).



1.5 Figure 1 *– Six different ways the user will be able to interact with the application on their phone.*

Google's lead UX designer, Jenny Grove, details 25 principles of design specific to the creation of application in a series of posts on think with google. These principles are very good at correlating some of the overarching principles and rules into more concrete methods or just create good user interactions. Some of the most relevant to our application are:

- Filter and sort options.
- Prominent search fields.
- Asking permissions in context.
- Calls to action are front and centre.
- Distinct categories with no overlapping.
- Showing its value by addressing the task clearly.
- Allowing the user to manually change of location.
- Responsive visual feedback after significant actions.

The permissions and location change principles are especially important as our application will require access to the user location.

1.5.1.4 User Research

The users of our application will most likely be first year student that are unfamiliar with the university campus'. Using the university of Brighton demographics from the website we can estimate that a third of the undergraduate students would be viable to use the application. Therefore, we have roughly 5,700 potential users, 20.33% of the university population. A survey would be a good way of determining the user's opinion on the idea of this application as well as informing the user's needs and requirements. However, the survey questions asked must not be leading so that any answer backs up the survey producer's agenda.

1.5.1.5 Competition Research

The most popular navigation application on the market is google maps. Due to its ease of use when users navigate between places and how much information it can offer. However, google maps is for general navigation use between places rather than navigation within campus. Despite this difference we can use some of the map conventions google sets and build off users existing knowledge of using similar map applications. The Brighton university website also hosts a map of their own, indicating the various buildings, scattered across Brighton, Eastbourne and Hastings which are part of the university. This general overview of campus builds could be implemented within our application for users to select the campus they wish to view.

1.5.2 Application Analysis

1.5.2.1 Mobile Application

For this project, the app that we have decided to design is an interactive map of the Brighton University campuses. The reason for this is due to the fact that as new students in university, we have knowledge on how easy it can be for new students to get lost or not be able to find their way through the campus therefore we have created “MapUs”. The purpose and aim of the application is to assist not only university students locate destinations in the campus, but also those who may be visiting the campus for other purposes such as open days, events and so forth. The app will provide information that is relevant and required to aid the users in locating the room or area they need to traverse to. The app will illustrate information such as room timetables, which is especially useful for students so that they may know when a room is free and to reassure them that they are heading for the correct place. In addition to this, the app will also display information concerning the important rooms in the campus’ such as library opening times and closing times, the location of cafes and if events are being held in the campus, the app will also disclose where and when the event is.

The application will also include numerous features such as a search feature so that the user can input what room they are looking for and have the information displayed instantly. Furthermore, the app will also assist the users in finding relevant locations such as unused classrooms, lifts, green areas, libraries, cafes, and toilets. Through the app, when the user searches for something, when the information is returned, it will be highlighted, ensuring that the user can easily find the information that is relevant to them. There will also be a final feature that is important for the application and that is the fact that the app will have a shareable geolocation. This will allow the users to locate their friends and fellow course members with ease and will reduce the hassle of having to call or text their friends and waiting for their reply before being able to start their journey.

For security purposes, the application will have a security measure which we have implemented through the inclusion of a gateway access through the use of student ID’s. This means that to access higher level information and more detailed and in-depth information on the university campus’, the user will have to log in through the app using their student ID. Guests will have access to basic information such as room locations but to be able to access timetables of rooms they will need higher level access.

1.5.2.2 Scenarios and User Journeys

For our scenarios we decided to make them for 3 different types of users we could be getting. They are as follows: a student, a guest visiting the university and a user with a disability.

Our first scenario was for a student named Chad that attends the university and wants to find a computer room that does not have a scheduled class in. To do this, he opens the application and is then prompted to login using his university login details. During this period, the application needs to display all necessary UI and run the required checks in the background. The student then needs to search for computer rooms. The app then simply needs to retrieve all of the room data and display all of the computer rooms on all floors. And to differentiate the free rooms from the rooms in use the app needs to display the free rooms a different colour from the rest.

The next scenario was made for someone who does not know their way around the campus and needs a little help, we called her Dorothy. She is currently located in the Cockcroft building, Moulsecoomb campus. Dorothy has an event which she is running with permission from the university and does not know how to get to the Watts building. She is able to access the MapUs application which prompts her to either login via the use of student details or as a guest, as Dorothy does not have student credentials, she will only be able to login to the app as a guest. This will limit the level of access and information which she is shown however still allow her access to the map. She selects the campus she wants via the menu, so the application needs to retrieve and display the relevant maps. Dorothy then needs to input the watts building into the search bar the app will then need to display the user's location and the route to the selected destination.

Our third scenario was made for Brian who is a disabled user. Brian has downloaded the app in order to find all and also the best route he could use for disabled users, as he is not a student Brian can only login at a guest level. When he has logged in, he will be able to change settings and preferences to suit him best in this case prioritising only the routes which adhere to his needs and by doing so the app will change the current pathing algorithm to best suit Brian's options.

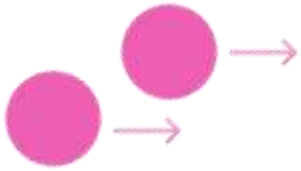
1.5.3 Developing Designs

1.5.3.1 Interface Designs

Once a user logs into the application, a 3D moveable rendition of their chosen campus will appear. We wanted the map to be in the centre of the screen whenever possible, only moving up or to the side when needed, such as when an options menu is open. This allows for easy navigation, resulting in good visibility and yield closure and reducing the amount of short-term memory load required.

There are 3 buttons we ended up settling on, which were the Settings, Key and Me buttons. The Settings button opens up a drop-down menu, pushing the 3D down so that it is still usable, but made smaller so that the settings take priority. From here the user can change many options, including which campus they want to look at, and disability options. One of these is colour blind mode, which changes the colour of the icons so that they are easier for the user to see if they have difficulty differentiating certain colours. This allows for good visibility and feedback, as each option they choose clearly changes the way the app behaves, allowing the user to see their changes take place.

The second button is the Key, which when opened expands up to half the size of the screen, listing what all the different icons are. The final one is the Me button, which repositions the map around where you are currently in the campus, provided you have geolocation enabled in the settings. All 3 buttons and the search bar are all on the screen at all times, with the one exception of when a user is searching for a room. Instead, possible room options fill the screen, in order to make it quicker and easier for the user to select rooms. This makes the app consistent to use, allowing shortcuts to speed up the users experience when quickly navigating through the application.

Mapped Interactive Movements		
Gesture	**Movement**	**Description of Effect**
Touch		Interacts with map elements, prompting effects.
Swipe		Moves the perspective in the horizontal plane.
Pinch Open		Adjusts the perspectives scale down to zoom out.
Pinch Close		Adjusts the perspectives scale up to zoom in.
Two Finger Swipe		Tilts the perspective up or down depending on path.
Rotate		Rotates the perspective either clockwise or anti-clockwise.

1.5 Figure 2 *– The six ways the user can interact with the 3D university buildings in the application.*

1.5.3.2 Prototype Designs

The prototype's main strength lays within the UI design, as it is simplistic and clear of unnecessary clutter, leaving the centre of the page clear so that the user can view the map fully. Information only appears when they interact with an element, keeping the user focused on the task. The map is also very minimalistic graphics wise while still looking elegant, in order to provide a clear display for the user. The application UI also has good affordances, as there are very few buttons on the screen at a time, and it is always easy to understand what each of them due to the word overlaying the button.

Due to the nature of the application being a map, our prototype does not display how the map will look in the final version, as this would require a full 3D model of a building on campus. Another weakness is that you cannot select by floors, which can be overcome by search for the room you want in the search bar, however this would be a nice extra to make the app even easier to use.

1.5.4 Evaluation of Report 1.5

In preparation, we had to find someone who has just entered their first year of life at university and would likely use the application. The potential user that we found to interview was someone who had moved to Brighton to study here and had no previous knowledge of the layout of the campus. We had asked the user if he could still remember his first few days and weeks at university before carrying out the interview due to the fact that if the user no longer had knowledge of this, then the interview would be redundant.

The user was asked questions such as “Which of these university campus’ do you regularly visit?” and “Do you think having a map of the campus would have been helpful?”. The session took place in a room where the potential user can easily be heard and can voice out their opinions. The potential user was given the survey to fill in and answer before being given the chance to fully voice out their opinion about the app.

Post interview, the potential user voiced out his opinions about the application and indicated that the applications layout is good and the fact that we had included so many features was a great bonus, although the image of the campus in the background can sometimes be off putting and also some colour would have been pleasing, instead of having a monotone app. In addition to this, the potential user said that he would highly recommend the application to new students that are about to attend university due to the fact that having an application like that would be really useful and would ease the struggle of finding rooms for the students.

After receiving this feedback, we did consider changing the design of the application but decided in the end that it would be better to leave it monotone for now and that although the fact that the picture of the campus is in the background and is not the most pleasing, it helps remind the users which campus they are currently viewing.

1.5.5 Reflection of Report 1.5

Throughout this project we kept on encountering new ideas which could be implemented into our application, some more useful than others, although we did not implement all of the aspects and conventions into our designs, we currently have been able to show the ideas which could be used if we were to ever turn our prototype into a real functioning application.

One point which will heavily restrict us on turning our application into a real functioning app is the university itself; in order to grant students access to their privileges we will have to discuss with the university on allowing us access to the entire database which may have some legal issues - Data Protection Act 1998 - running beside it as they would be sharing the students data without currently granting permission and even if they were to have permission it may not be from every student body but rather only from those open to having their data shared.

We followed the user-centred design guidelines however we did not create it to interact with our users as much as we could have in order to receive feedback which would have allowed us to create a greater level of personas and scenarios. Another problem which was brought to light to us when creating conventions was how we would be able to implement a way to accurately position a device within a space as the technology may not be currently available.

1.5.6 Appendix of Report 1.5

Appendix A – Personas

Persona 1	
Name	Alex
Age	20
Gender	Female
Occupation	Second year Brighton University student.
Description	Alex is already studying at Brighton University, and has therefore already had some experience with similar map layouts for the various buildings. They know how to navigate around practically every building on their campus, except for very obscure rooms that students usually do not have direct access to.
Experience	Has never used the application before but has studied all campus maps.
Persona Types	Quiet and Curious
Key Drivers	Looking to find the fastest way around campus.
Platform	Android

Persona 2	
Name	Brian
Age	18
Gender	Male
Occupation	College student studying for A-Levels.
Description	Brian is a soon to be first year student who has attended events held throughout the introductory days and open evenings but has a rough understanding of how to navigate from building to building. He still requires some level of assistance when trying to navigate between rooms and also between various facilities.
Experience	Experienced user of applications in general but not of this particular one.
Persona Types	Busy and Productive
Key Drivers	Interest in attending the university, needs to know the disabled access points.
Platform	IOS

Persona 3	
Name	Chad
Age	20
Gender	Male
Occupation	Student studying generic course second year.
Description	Chad is a second year who does not require very much assistance when navigating and locating areas and facilities around his campus however is unfamiliar with the other campuses
Experience	Was a frequent user during the start of the university but has slowly used less and less
Persona Types	Latest and Greatest
Key Drivers	Currently studying near Brighton.
Platform	IOS

Persona 4	
Name	Doris
Age	65
Gender	Female
Occupation	Retired
Description	Dorothy is a retired event handler who has planned to run an event within the university campus. Dorothy does not know any of the room locations in the campus and will require heavy assistance in traversing through the campus.
Experience	Rarely uses mobile applications and has never used the map before.
Persona Types	Social and Curious
Key Drivers	Interested in hosting an event in the campus.
Platform	Android

Appendix B – Scenarios

Scenario 1		
User *- All Application Users*		
User Goal *- Setting up application options for users frequently using the application.*		
User Intentions	**System Responsibility**	**Backend Processing**
Open up the app	Prompts user to either enter their student login details, use the guest login or login if you have previously logged in with a student login.	Display application UI and check previous log in status.
Logging into the app	Capture any required input data.	Authenticate student details, or restricted access to guest login.
Set up the application	Prompt user to set location settings (a set map on the start-up screen, so that the user will not need to keep inputting a frequent map).	Stores user selection to application.
Determining location	Allow location sharing.	Tracks user current location and shares to selected users.
Access any required disability options	Offer key option settings that would change the experience, such as disability access, colour-blindness, UI scale, etc.	Allow user to alter settings to adhere to user preferences.

Scenario 2		
User *- Dorothy*		
User Goal *- Find a way from the Cockcroft building to the Watts building.*		
User Intentions	**System Responsibility**	**Backend Processing**
Open up the app	Prompts user to either enter their student login details, use the guest login or login if you have previously logged in with a student login.	Display application UI and check previous log in status.
Selecting campus via the menu	Choose campus you wish to view.	Retrieve map details and display on screen.
Search for building in search bar	Retrieve inputted data and display user location and selected destination.	Process data input and send data search.

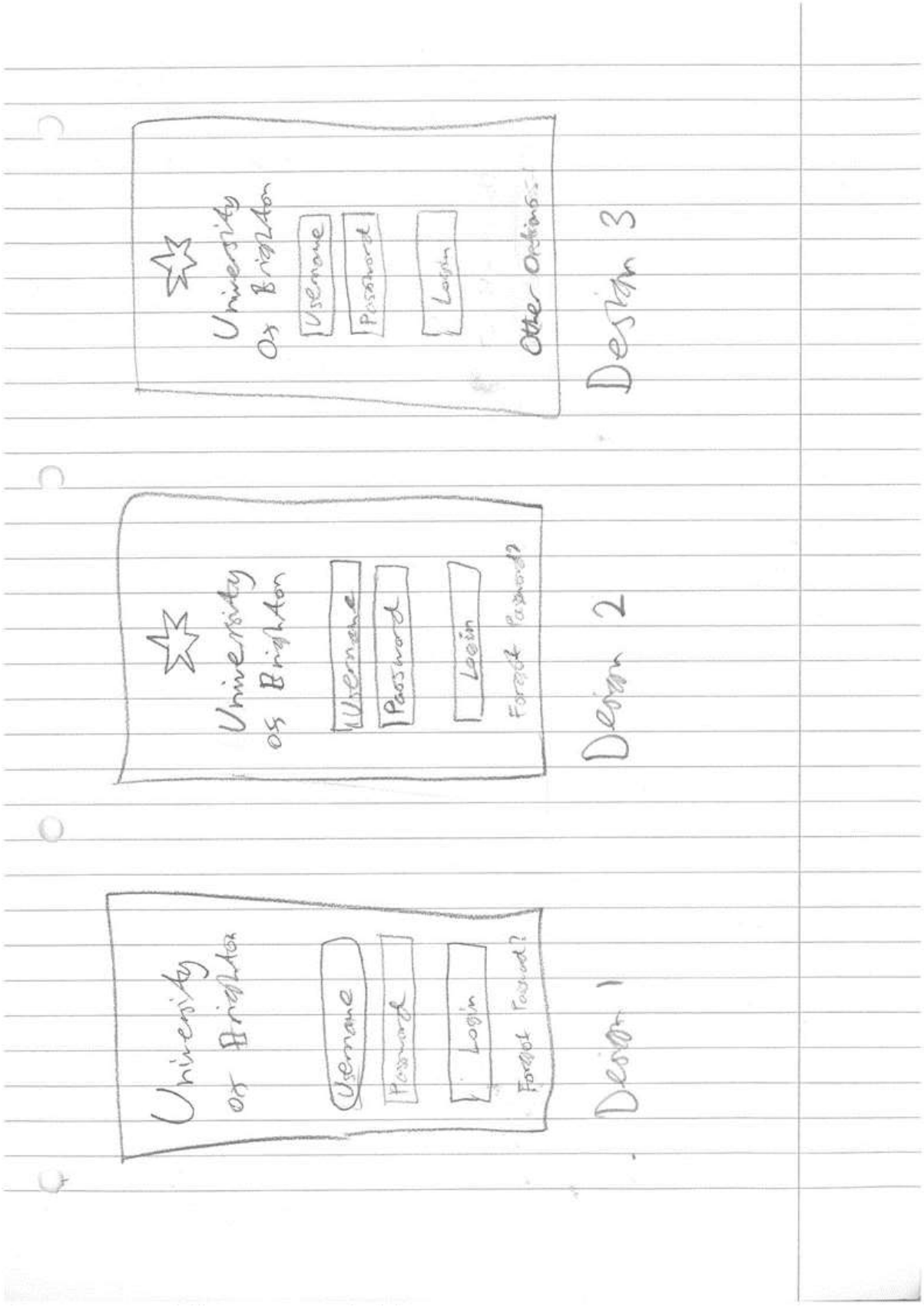
Scenario 3		
User *- Alex*		
User Goal *- Find a computer room that does not have a scheduled class to work in.*		
User Intentions	**System Responsibility**	**Backend Processing**
Open up the app	Prompts user to either enter their student login details, use the guest login or login if you have previously logged in with a student login.	Display application UI and check previous log in status.
Searches computer rooms in search bar	Display floor and room schedule.	Retrieve data of all rooms from each available floor.
Look for a marked free room among the results of the search	Display free rooms a different colour from the ones currently in use.	Filter rooms available from rooms currently with lessons.

Scenario 4		
User *- Chad*		
User Goal *– Locate their group members through the app location options.*		
User Intentions	**System Responsibility**	**Backend Processing**
Open up the app	Prompts user to either enter their student login details, use the guest login or login if you have previously logged in with a student login.	Display application UI and check previous log in status.
Allow the app to use location of the phone	Display the current location of the user.	Retrieve data from geo-location from the phone.
Search individual names of people in his group	Display location of the member if their location setting is turned on.	Retrieve and display current location of member.

Scenario 5		
User *- Brian*		
User Goal *- Navigate throughout the university campus whilst in a wheelchair for open day.*		
User Intentions	**System Responsibility**	**Backend Processing**
Open up the app	Prompts user to either enter their student login details, use the guest login or login if you have previously logged in with a student login.	Display application UI and check previous log in status.
Select disabled access	Prompts user to use the setup to select the disabled mode.	Change the pathing algorithm to display best disabled access.

Scenario 6		
User *- Dorothy*		
User Goal *- Find the location of the conventions going on in the campus.*		
User Intentions	**System Responsibility**	**Backend Processing**
Open up the app	Prompts user to either enter their student login details, use the guest login or login if you have previously logged in with a student login.	Display application UI and check previous log in status.
Search for a currently hosted event	Bring up the current event on the map.	Compare current events in database and retrieve data.

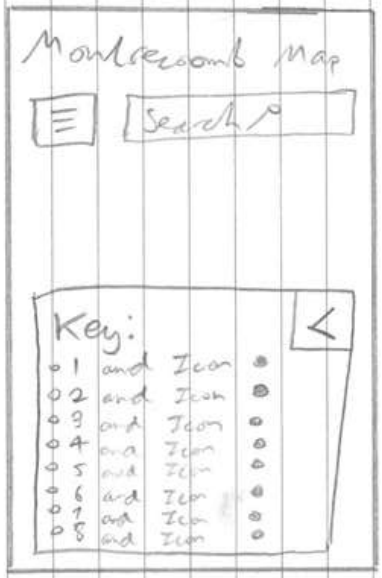
Appendix C: Low Fidelity Designs on Paper



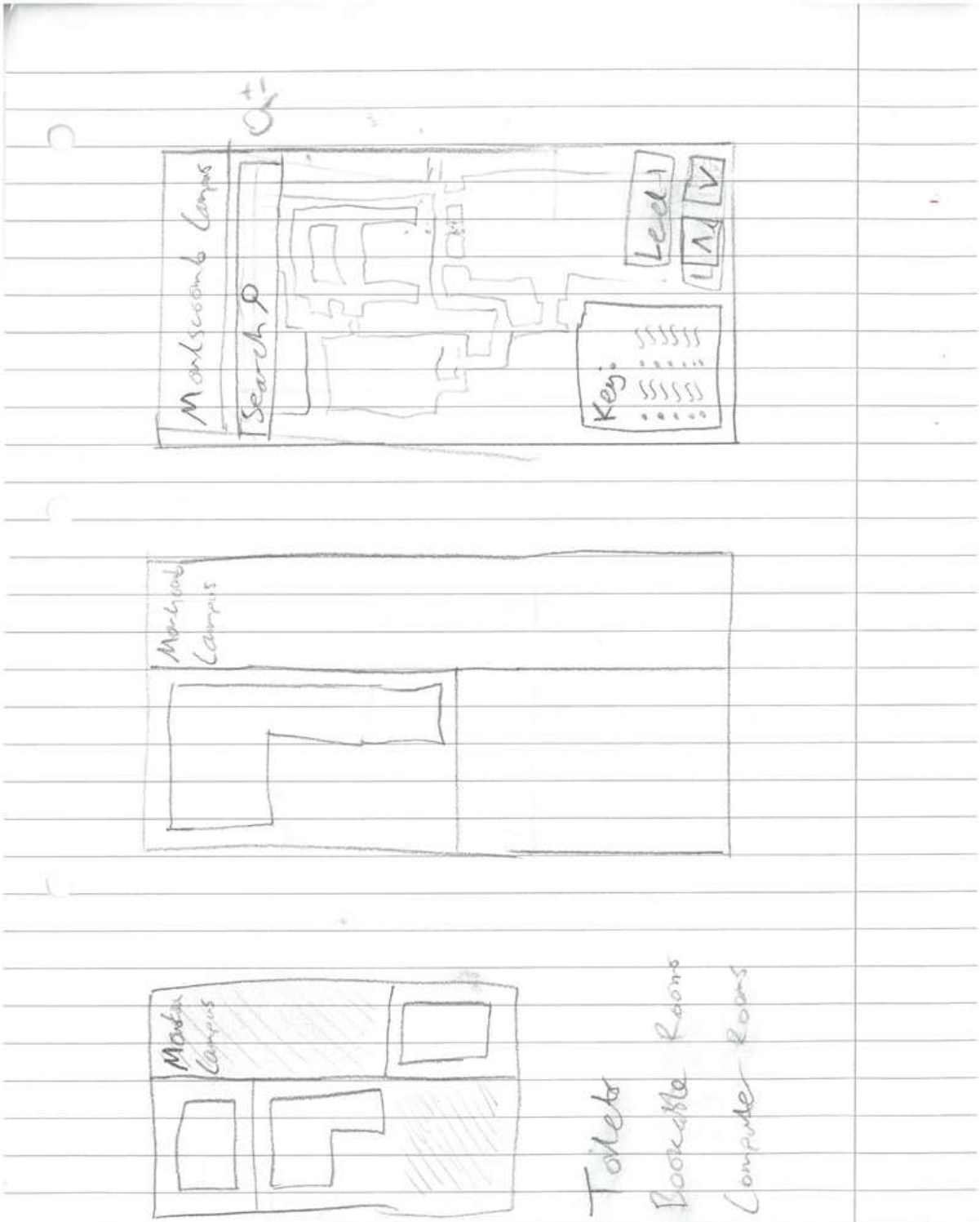
Key Idea #1

Key Idea #2

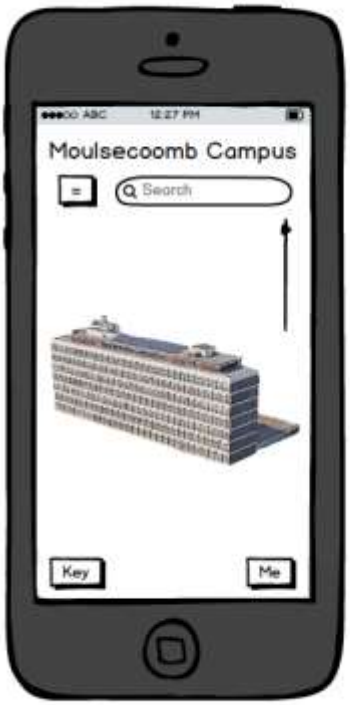
Key Idea #3

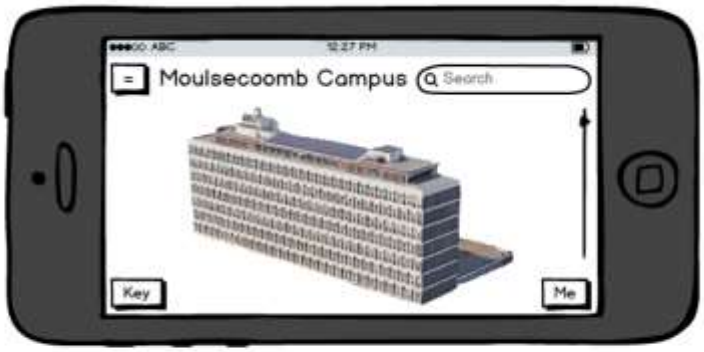


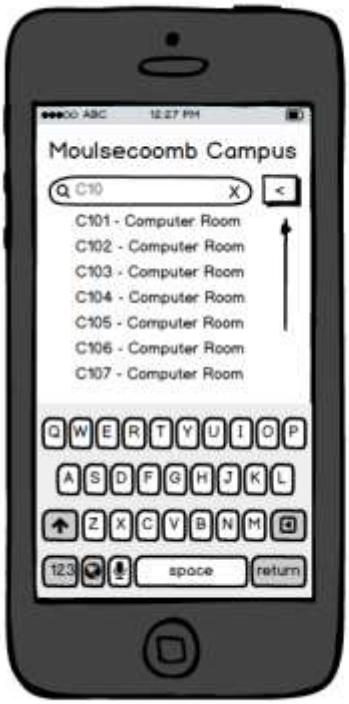
- Log-in
- Map Icons (room type Icons, Shop Icons, Lifts, Water fountains)
- Location information side-panel (shows minimal room info)
- Location information full (shows full room info)


Markscomb Campus
Search
Level 1
Key:
Markscomb Campus
Marksc Campus
Toilets
Bookable Rooms
Computer Rooms

Appendix D: High Fidelity Designs on Balsamiq Wireframes

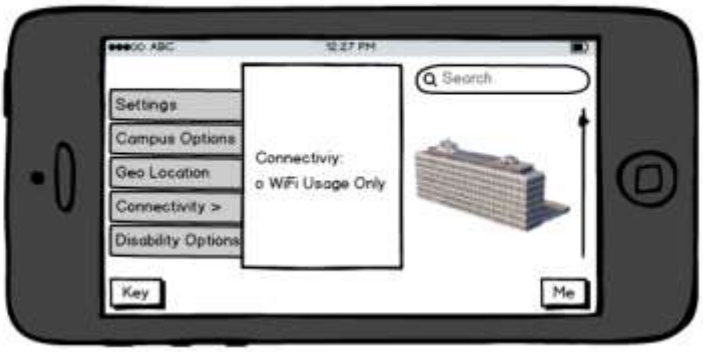




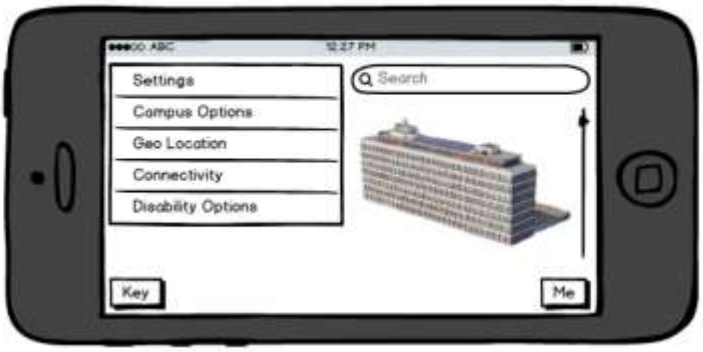

Moulsecoomb Campus
C10
C101 - Computer Room
C102 - Computer Room
C103 - Computer Room
C104 - Computer Room
C105 - Computer Room
C106 - Computer Room
C107 - Computer Room
space
return


C10
C101 - Computer Room
C102 - Computer Room
C103 - Computer Room
space
return

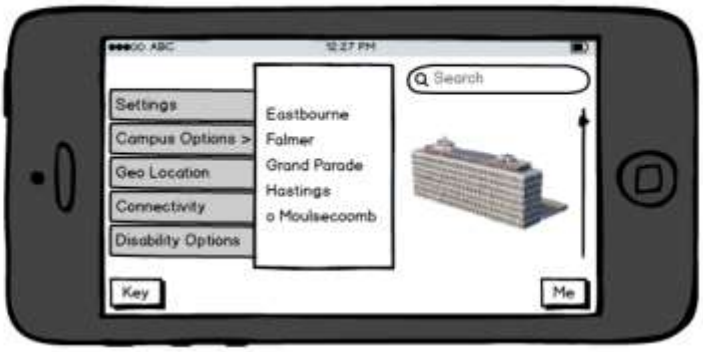

Moulsecoomb Campus
Settings
Campus Options
Geo Location
Connectivity
WiFi Usage Only
Disability Options
Key
Me


Settings
Campus Options
Geo Location
Connectivity >
Disability Options
Connectivity:
o WiFi Usage Only
Search
Key
Me

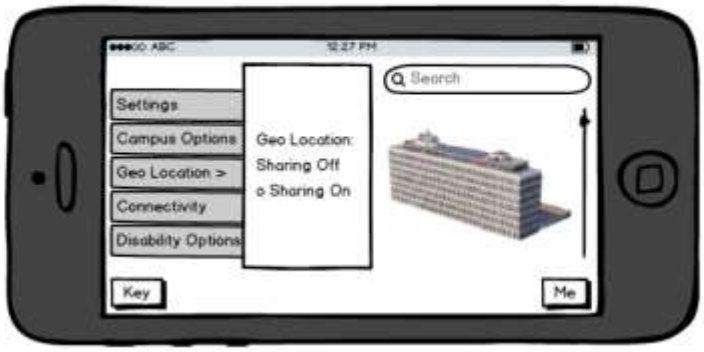

Moulsecoomb Campus
Settings
Campus Options
Geo Location
Connectivity
Disability Options
Key
Me


Settings
Campus Options
Geo Location
Connectivity
Disability Options
Search
Key
Me

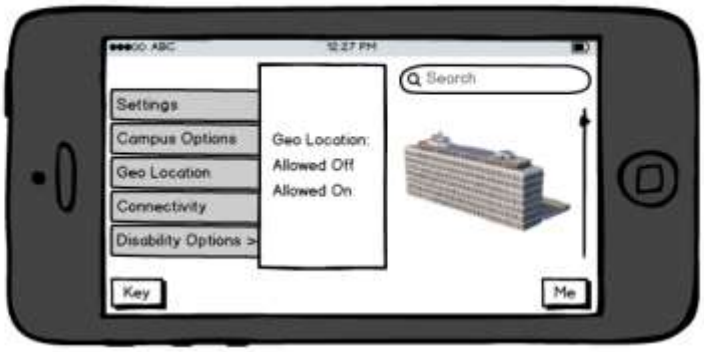

Moulsecoomb Campus
Settings
Campus Options
Eastbourne
Falmer
Grand Parade
Hastings
Moulsecoomb
Geo Location
Connectivity
Disability Options
Key
Me


Settings
Campus Options >
Geo Location
Connectivity
Disability Options
Eastbourne
Falmer
Grand Parade
Hastings
Moulsecoomb
Search
Key
Me

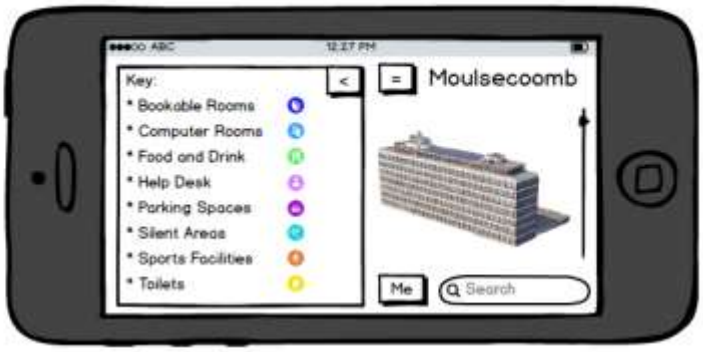

Moulsecoomb Campus
Settings
Campus Options
Geo Location
Sharing Off
Sharing On
Connectivity
Disability Options
Key
Me


Settings
Campus Options
Geo Location >
Connectivity
Disability Options
Geo Location:
Sharing Off
o Sharing On
Search
Key
Me

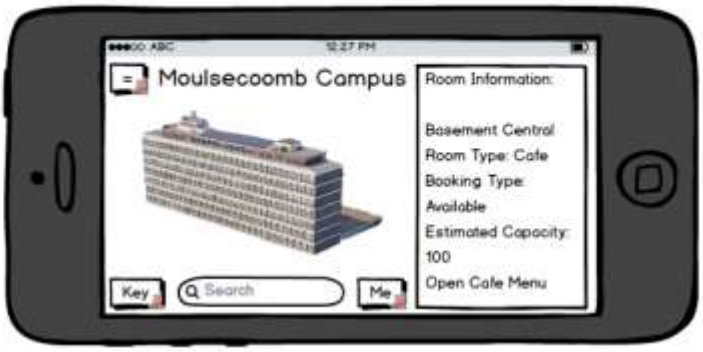

Moulsecoomb Campus
Settings
Campus Options
Geo Location
Connectivity
Disability Options
Colourblind Mode
Wheelchair Access
Key
Me


Settings
Campus Options
Geo Location
Connectivity
Disability Options >
Geo Location:
Allowed Off
Allowed On
Search
Key
Me


Moulsecoomb Campus
Search
Me
Key:
* Bookable Rooms
* Computer Rooms
* Food and Drink
* Help Desk
* Parking Spaces
* Silent Areas
* Sports Facilities
* Toilets


Key:
* Bookable Rooms
* Computer Rooms
* Food and Drink
* Help Desk
* Parking Spaces
* Silent Areas
* Sports Facilities
* Toilets
Moulsecoomb
Me
Search


Moulsecoomb Campus
Search
Key
Me
Room Information:
Basement Central
Room Type: Cafe
Booking Type: Available
Estimated Capacity: 100


Moulsecoomb Campus
Room Information:
Basement Central
Room Type: Cafe
Booking Type:
Available
Estimated Capacity:
100
Open Cafe Menu
Key
Search
Me

1.6 Project Management

For a full team project documented from conception to completion that I managed, please see the following **Agile Management Report** (also available on my portfolio if the PDF link is unavailable).

A similar, solo task that demonstrates project management is also showcased below. This project reports how technology is becoming increasingly more integrated within our daily lives, forcing us to confront issues we have never encountered before. The rate of which technology is evolving faster than ever before, resulting in some people becoming increasing wary of potential technological advances due to the many unknown outcomes that could occur. One such example is the innovation of self-driving cars; many people feel uncomfortable with the thought of a computer dealing with life-threatening scenarios. Relevant skills such as researching, agile planning and coding have been used in order to produce a program that highlights these issues in an easier to digest way, in order to showcase users how technology should not be ignored but embraced.

1.6.1 Introduction

The main aim of this project is to showcase how technology is ultimately designed by humans, and as such, show to the user why we must embrace the positives that are provided. To do so, the project highlights how shying away from the negative implications could cause issues to arise further down the line, due to current present-day decision thought process. One such example is the innovation of self-driving cars; the underlying mechanics of which are one example where ethics and technology are already becoming increasingly interconnected. Once thought of as a distant dream, these autonomous vehicles are slowly becoming a reality. Despite this, many people still feel uncomfortable with the thought of a computer dealing with life-threatening scenarios. Despite the fact computers are capable of processing information much faster than humans are, and thereby reducing the overall risk of potential casualties, some people are still hesitant to the idea.

The example I decided to build upon for my project is a commonly known thought problem in the field of ethics known as the trolley problem. This thought problem is one that is designed to have no correct answer, but instead aims users to think how they would response when presented with only bad case scenarios of a runaway trolley. In the original well-known scenario, the problem states that there are 5 people in front of the trolley that will die if hit, but a spectator has the option to divert the trolley and hit 1 other person nearby instead. In doing so, less people will be killed, but at the cost of deciding to kill another human being. This thought problem is the perfect example to apply to the technology of self-driving cars; although the data suggests that less accidents will occur overall, there will be cases where difficult decisions must be coded and put into practice in the real world.

The program I have created takes this original concept and layers up multiple scenarios that could occur. After each scenario, the data is then temporally stored (and subsequently deleted when the program is terminated) to build up a report at the end of the evaluation. It is only until the end when the report is produced for the user when they aware how their decisions would affect real life scenarios with self-driving cars, in order to avoid influencing any previous decisions. The user can then reload the program to revaluate their answers if they so desire, but now under the knowledge of how it would affect the real world. In theory the answers should be the same both times, but some people feel differently when applying these concepts to technology, mainly due to its portrayal in movies and other media. This program aims to showcase how we are the ones who decide how technology behaves, and not the other way around.

1.6.2 Methodology

To make sure the time spent on my project was managed well, I decide to utilise the agile project management approach. This was an approach I had learned about but never truly used in a previous university module; it seemed to be a good way to manage my time efficiently, and as such I decided to go with it. I decided to set up biweekly sprints, allowing me to plan ahead effectively whilst still enabling myself flexibility for future sprints. This allowed me to adapt to challenges that I faced along the way, without feeling like I was wasting time by not sticking to a rigid timeline.

One such example can be seen in Sprint 4, where a particular deliverable of creating branching paths took longer to write than expected. The agile approach allowed me to continue and conclude the task in the next sprint, whilst still allowing me to keep on schedule. This was also helped by another methodology I applied, the MoSCoW method, where I split each deliverable up into one of three categories, must, should, and could. This enabled me to make sure I was including all the tasks that needed doing in order to progress to the next sprint, before focusing on the should and subsequently could deliverables.

Product Backlog									
ID	**Deliverable**	**Planned Sprint**	**Actual Sprint**	**Est. Hours**	**Adjustment Factor**	**Extra Hours Needed**	**Total Actual Hours**	**Done?**	**Issue Log**
10	Create story branches based on player actions	4	4	19	Story branches took longer to write than expected	2	21	Yes, but late	Ran over schedule but should be included, so continued in the next sprint
11	Create story branches (continued)	5	5	22	-	0	22	Yes	-

***1.6 - Figure 1**: An extract from **Appendix A** that shows how deliverables 10 and 11 during sprint 4 ran over and subsequently carried over into the next sprint. The extra time spent in these tasks were kept track of throughout the project, in order to aid with future time allocations in the following sprints.*

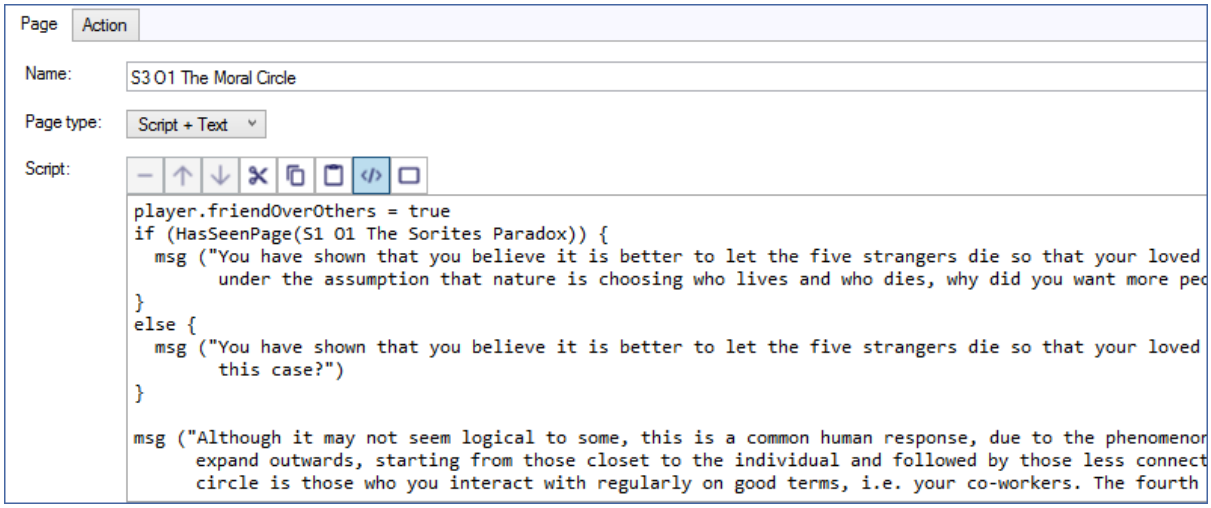
Another example where this helped can be seen in Sprint 6, where I had to adapt the project in order to meet the set requirements. Due to the coronavirus pandemic, unforeseen challenges arose across each module in my final university year. When the time came to restart work on the project, keeping track of my previous sprints dramatically helped me remodel past efforts I had already been working towards, previously a story-based version of the product, and refocus my efforts to fit the new timeframe. Without the use of agile planning, I do not believe this would have been possible.

In order to take the ideas that I had on paper and make them into a reality, I had to choose a way to represent my product. Back at the start of Sprint 4, I knew I wanted to make a program that took various inputs from a user in order to progress down a diverging path, displaying different scenes to the user depending on their choices. I decided this after researching various games and products that utilised gamification in order to hold the user's attention.

1.6.3 Production Description

To achieve my goal of producing a program showcases how data is used to inform decisions that can be applied to furthering technology, I decided to create an interactive evaluation application. I researched various different software to see what would work best for this, including Twinery,

Squiffy and TADS. These are all programs that take piece of code alongside pre-made functions and libraries in order to produce a clean and easy to use product. After testing out a variety of options, I decided to use a program called Quest (Quest, 2020) for my project.



***1.6 - Figure 2**: A screenshot of the Quest software that I used to help me meet the aims of my project.*

Quest is a text adventure creator that allows you to program multiple options and branching paths, with the aid of global variable to keep track of in-game objects that the user interacts with. Although I was not producing a text adventure game, I was producing a program that required branching paths based on a user's input. The software also allowed me to switch between a prebuilt user interface and a more detailed view of the actual code on the fly. This permitted both ease of coding in certain sections, whilst allowing me to go more in-depth with branching if statements and reading global variables to convey specific messages in others. It also provided me with prebuild functions, such as *HasSeenPage()* as can be seen in the example screenshot above. As such, Quest was the perfect software to develop my ideas in code for my project.

Before I dove straight into coding, however, I decided it was best to plan ahead first. In previous projects, I have utilised basic flow charts that make use of pseudocode to follow a basic structure. Since I was aiming to create a product with branching paths, this appeared to be a smart way to start. The processed ended up making the whole processes far easier, as I was able to keep track of the various variables that needed setting and changing depending on the input of the user. In the final evaluation at the end of the program, I pulled data from a national traffic accident survey (U.S. Department of Transportation, 2015) to showcase how over 94% of traffic accidents that occur annually are due to human error alone. This was done on purpose to push the point that self-driving cars are an inevitability rather than a novelty. In doing so, this showcases the need to evaluate the decisions that we make when it comes to developing new technologies, and how they subsequently impact and intertwine within our daily lives over time.

1.6.4 Critical Review

Overall, the game achieves the target goal of showcasing how data can be utilised to influence the way new technologies behave as they become integrated into our daily lives. This is done by using a real-life example of self-driving cars, an example of a new technology in development that one day in the near future will become common place. This was achieved through the use of an easy-to-use program hosted online that allows users to make decisions based on real-life provided scenarios, via the use of branching paths and a rundown of the user's consequences at the end of the evaluation.

In terms of planning the project, I learned how to successfully manage biweekly deadlines using sprints and the agile methodology, coupled with the use of MoSCoW priories to keep my progress on track. I also utilised flowcharts in order to plan the program ahead, to make it easier for myself when it came to coding and linking up the multiple branches the user can take. Both of these are provided as separate documents at the end of this report, which can be looked through for further evidence of learning and management of my project. I also learned how to use OBS screen recording software (OBS Studio Contributors, 2020) for the first time, which was yet another skill that I had learned and developed on during the process of producing my project.

Whilst I am happy with the final product, I did have to figure out some complications along the way that I would approach differently next time, if I were to undertake a similar project in the future. The first major one would be a clearer end goal. Although I knew from the start that I wanted to make a decision-based program, I initially planned to produce a story-based program instead. I used Jeff Vandermeer's Wonderbook (Vandermeer & Zerfoss, 2018) initially in order to create branching storylines. It would have followed the same type of progression as seen in the final product, but with more emphasis on the human side of these decisions, in the aims of leaving a more lasting impression. Ultimately, this was something that I had to pull back later on, which I was able to do effectively due to the use of my biweekly sprints, but in the future, I will know to keep a clearer mindset on how much work I would actually be able to produce over several conceding sprints.

I would also like to have included more branching paths and possibilities based on the user's inputs, as I believe this would have given users more to walk away with and think about after running through the program. This is something that I decided to keep scaled back on after Sprint 6 as previously mentioned, but if I were to approach this task again, I would have started with producing a flowchart right from the first sprint I create, as I discovered that was something that was immensely helpful in seeing the outline of my program. I would choose to use the same software as well, as it was very easy to work and replicate what I had planned to produce in my flowcharts.

Upon reflection of my work, I feel that my knowledge of highlighting an issue, managing a project effectively and producing a product to showcase said issue has definitely improved. If the program I were to be expanded upon in the future, it could also be applied to other technological advances that are currently under development. I decided to focus on self-driving cars, but the same principles could be used to discuss controversial topics such as facial recognition, digital footprints, or bionic argumentations. If I were to evolve the program to involve such topics, I would use the same methodologies to aid me in this. Due to the previously mentioned methodologies that I utilised, I was able to create a product that showcases how we are the ones who decide how technology behaves, and not the other way around.

1.6.5 Presentation

A presentation of this project can be seen at the following link: ***https://youtu.be/m26xmvnXFw4***

1.6.6 Appendix

Appendix A – Agile Sprint Planning

Product Backlog									
ID	Deliverable	Planned Sprint	Actual Sprint	Estimated Hours	Adjustment Factor	Extra Hours Needed	Total Actual Hours	Complete?	Issue Log
1	Come up with an interesting project proposal	1	1	12	-	0	12	Yes	-
2	Submit the required project proposal forms	1	1	2	-	0	2	Yes	-
3	Create an agile project planning spreadsheet	1	1	9	Extra time spent creating the spreadsheet	2	11	Yes	-
4	Select and meet up with a project supervisor	2	2	3	-	0	3	Yes	-
5	Create a document that outlines the project	2	2	19	Documentation took less time than expected	-1	18	Yes	-
6	Research how to write developed characters	3	3	7	-	0	7	Yes	-
7	Outline traits for each appearing character	3	3	6	-	0	6	Yes	-
8	Provide characters with ethical conundrums	3	3	9	-	0	9	Yes	-
9	Write up the interim investigation report	4	4	3	-	0	3	Yes	-
10	Create story branches based on player actions	4	4	19	Story branches took longer to write than expected	2	21	Yes, but late	Ran over schedule but should be included, so continued in the next sprint
11	Create story branches (deliverable continued)	5	5	22	-	0	22	Yes	-
12	Work on viva feedback from the supervisor	6	6	5	-	0	5	Yes	-
13	Solidify a choice of software to make the project in	6	6	17	-	0	17	Yes	-
14	Rescale the project scope to a manageable size	7	7	13	-	0	13	Yes	-
15	Implement new scope into a readable manner	7	7	9	-	0	9	Yes	-
16	Write the code for the project in Quest	8	8	8	Fixing game bugs took much longer than expected	8	16	Yes	-
17	Write the report for the project	8	8	7	-	0	7	Yes	-
18	Produce a project presentation	8	8	7	Project finalisation moved to next sprint for time	-3	4	Yes, but late	Ran over schedule but should be included, so continued in the next sprint
19	Finalise the project presentation	9	9	7	-	0	7	Yes	-

Sprints for the Individual Project

Sprint Information: *This sheet is an overall summary of each sprint for the individual project. Select one of the following sheets to see how each deliverable is broken down into smaller and more manageable tasks, with varing priorites.*		30/9/19 – 7/10/19	14/10/19 – 21/10/19	28/10/19 – 4/11/19	11/11/19 – 18/11/19	25/11/19 – 2/12/19	9/12/19 – 16/12/19	20/7/20 – 27/7/20	3/8/20 – 10/8/20	17/20/20 (5 Days)
		Sprint 1	*Sprint 2*	*Sprint 3*	*Sprint 4*	*Sprint 5*	*Sprint 6*	*Sprint 7*	*Sprint 8*	*Sprint 9*
Weekly Hours	Planned Effort	23	23	23	23	23	23	23	23	8
	Actual Effort	25	22	23	25	23	23	23	28	8
Overall Hours	Planned Total	23	46	69	92	115	138	161	184	192
	Actual Total	25	47	70	95	118	141	164	192	200

Sprint 1 - 30/9/2019 to 13/10/2019

Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Come up with an interesting project proposal	Must	1.1	Read the assignment brief to identify the requirements	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
			1.2	List a variety of project proposals that could be worked on	Must	Done	3	-	4	-	-	-	-	-	-	-	-	-	-	-	-	4	1
			1.3	Eliminate any ideas too short or too long for the time allocated	Should	Done	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	1	0
			1.4	Further develop the remaining proposals that could be chosen	Should	Done	6	-	-	-	-	5	-	-	-	-	-	-	-	-	-	5	-1
			1.5	Select the individual project proposal that will be worked on	Must	Done	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	1	0
2	Submit the required project proposal forms	Must	2.1	Download, fill out and submit "Appendix A - Project Proposal"	Must	Done	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	1	0
			2.2	Download, fill out and submit "Appendix B - Ethics Checklist"	Must	Done	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	1	0
3	Create an agile project planning spreadsheet	Must	3.1	Download the sprint planner provided by the university	Must	Done	1	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1	0
			3.2	Redesign the layout of the sprint for easier access to project data	Could	Done	8	-	-	-	-	-	-	-	-	-	-	2	5	-	3	10	2
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint lasted 2 hours longer than predicted.	
Actual Effort Hours								23	18	18	18	11	11	11	11	9	9	6	1	1	-2		

Sprint 3 - 28/10/2019 to 10/11/2019

Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Research how to write developed characters	Must	2.1	Research various methods to create well developed characters	Must	Done	3	-	3	-	-	-	-	-	-	-	-	-	-	-	-	3	0
			2.2	Create a list of all possible characters that could be used	Should	Done	2	-	-	2	-	-	-	-	-	-	-	-	-	-	-	2	0
			2.3	Expand on detailed secondary characters for main characters	Could	Done	2	-	-	-	-	-	-	-	-	-	-	-	2	-	-	2	0
3	Outline traits for each appearing character	Must	3.1	Create a list of all possible traits that characters could have	Must	Done	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	1	0
			3.2	Develop selected characters with different backgrounds and traits	Should	Done	3	-	-	-	-	3	-	-	-	-	-	-	-	-	-	3	0
			3.3	Create detailed traits to possible background characters	Could	Done	2	-	-	-	-	-	-	-	-	-	-	-	2	-	-	2	0
4	Provide characters with ethical conundrums	Must	4.1	Research different popular ethical dilemmas in philosophy	Must	Done	4	-	-	-	-	-	-	-	3	-	2	-	-	-	-	5	1
			4.2	Research various problems people face throughout life	Should	Done	4	-	-	-	-	-	-	-	-	-	3	-	-	-	-	3	-1
			4.3	Apply appropriate problems and dilemmas to each character	Should	Done	1	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1	0
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint lasted exactly as long as predicted.	
Actual Effort Hours								23	19	17	17	13	13	13	10	10	4	4	0	0	0		

Sprint 2 - 14/10/2019 to 27/10/2019

Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Select and meet up with a project supervisor	Must	2.1	Finalize a supervisor to oversee the development of the project	Must	Done	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-	1	0
			2.2	Organise a time to meet up with the project supervisor	Must	Done	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-	1	0
			2.3	Meet up with the supervisor to discuss possible ideas	Must	Done	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	1	0
3	Create a document that outlines the project	Must	3.1	Set-up a Google Docs file that can be worked on from anywhere	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
			3.2	Outline the direction that the project may head towards	Must	Done	5	-	3	-	-	-	-	-	-	-	2	-	-	-	-	5	0
			3.3	Eliminate any ideas unsupported by the project supervisor	Should	Done	1	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1	0
			3.4	Develop possible avenues in order to see which would best	Should	Done	11	-	-	-	-	-	-	-	-	-	2	3	3	-	2	10	-1
			3.5	Decide on the best avenue to take the individual project	Must	Done	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	0
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint ended 1 hour faster than predicted.	
Actual Effort Hours								23	18	16	16	16	16	16	16	15	10	7	4	4	1		

Sprint 4 - 11/11/2019 to 24/11/2019

Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Write up the interim investigation report	Must	2.1	Download, write up and submit the interim investigation report	Must	Done	3	-	3	-	-	-	-	-	-	-	-	-	-	-	-	3	0
3	Create story branches based on player actions	Must	3.1	Write a prototype master branch (if no player actions occurred)	Must	Done	5	-	-	3	4	-	2	-	-	-	-	-	-	-	-	9	4
			3.2	Highlight possible areas that the player could choose the outcome	Must	Done	1	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1	0
			3.3	Design the structure to make it easy to backtrack old choices	Must	Done	1	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	0
			3.4	Create branches based off each possible player choice for Act I	Should	Done	6	-	-	-	-	-	-	-	4	-	-	-	6	-	-	10	4
			3.5	Create branches based off each possible player choice for Act II	Should	Not Started	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-6
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint lasted 2 hours longer than predicted.	
Actual Effort Hours								23	19	16	12	12	9	9	4	4	4	4	-2	-2	-2		

Sprint 5 - 25/11/2019 to 8/12/2019

Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Create story branches (deliverable continued)	Must	2.1	Create branches based off each possible player choice for Act II	Should	Done	10	-	4	3	-	3	-	-	-	-	-	-	-	-	-	10	0
			2.2	Create branches based off each possible player choice for Act III	Should	Done	10	-	-	-	-	-	-	-	6	-	4	-	-	-	-	10	0
			2.3	Format the look and layout of the branches to be easy to follow	Could	Done	2	-	-	-	-	-	-	-	-	-	-	-	-	2	-	2	0
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint lasted exactly as long as predicted.	
Actual Effort Hours								23	18	15	15	12	12	12	6	6	2	2	2	0	0		

Sprint 6 - 9/12/2019 to 22/12/2019

Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Work on wva feedback from the supervisor	Must	2.1	Talk to the supervisor about current progress and ideas	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
			2.2	Take away notes to analyse what could be changed or improved	Must	Done	4	-	2	2	-	-	-	-	-	-	-	-	-	-	-	4	0
3	Solidify a choice of software to make the project in	Must	3.1	Test out the Twinery website to see if it would be the best fit	Should	Done	4	-	-	-	-	2	-	-	2	-	-	-	-	-	-	4	0
			3.2	Test out Squiffy Software Squiffy to see if it would be the best fit	Should	Done	4	-	-	-	-	-	-	-	2	2	-	-	-	-	-	4	0
			3.3	Test out TADS Software to see if it would be the best fit	Should	Done	4	-	-	-	-	-	-	-	-	-	4	-	-	-	-	4	0
			3.4	Test out Quest Software to see if it would be the best fit	Should	Done	4	-	-	-	-	-	-	-	-	-	-	2	2	-	-	4	0
			3.5	Make a firm choice on the piece of software to use for the project	Must	Done	1	-	-	-	-	-	-	-	-	-	-	-	1	-	-	1	0
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint ended 1 hour faster than predicted.	
Actual Effort Hours								23	19	17	17	15	15	15	11	9	5	3	1	1	1		

Sprint 7 - 20/7/2020 to 2/8/2020

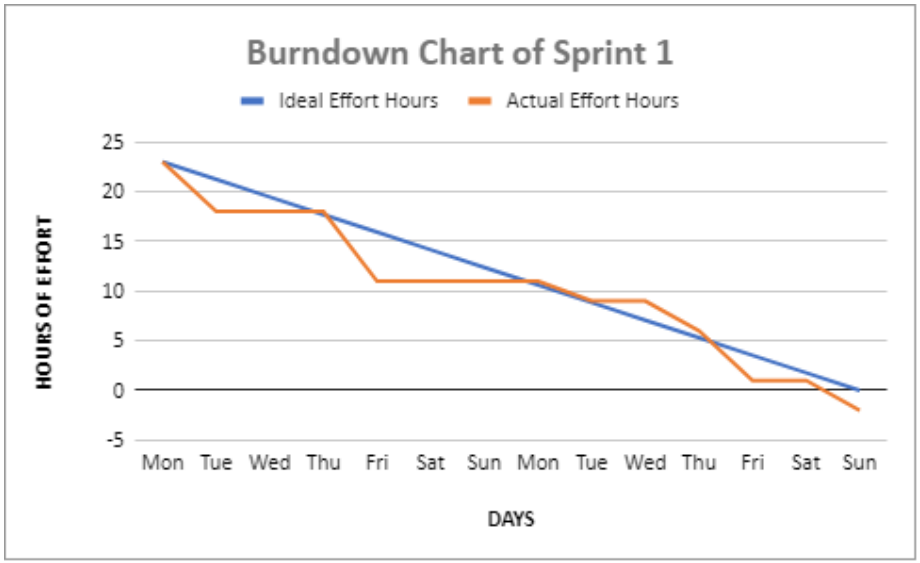
Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Rescale the project scope to a manageable size	Must	2.1	Reevaulate what parts of the project remain in place	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
			2.2	Shift focus from character based storytelling to ethics based	Should	Done	1	-	1													1	0
			2.3	Take original Act I idea (Trolley Problem) and expand upon it	Should	Done	1	-	1													1	0
			2.3	Research various other related ethical driving thought problems	Should	Done	10	-	-	-	3	2	-	3	2	-	-	-	-	-	-	10	0
3	Implement new scope into a readable manner	Must	3.1	Create a flowchart of all the paths the player's possible choices	Should	Done	8	-	-	-	-	-	-	-	-	3	3	-	2	-	-	8	0
			3.2	Download Quest to implement the game logic as scripted code	Must	Done	1	-	-	-	-	-	-	-	-	-	-	-	1	-	-	1	0
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint lasted exactly as long as predicted.	
Actual Effort Hours								23	19	19	16	14	14	11	9	6	3	3	0	0	0		

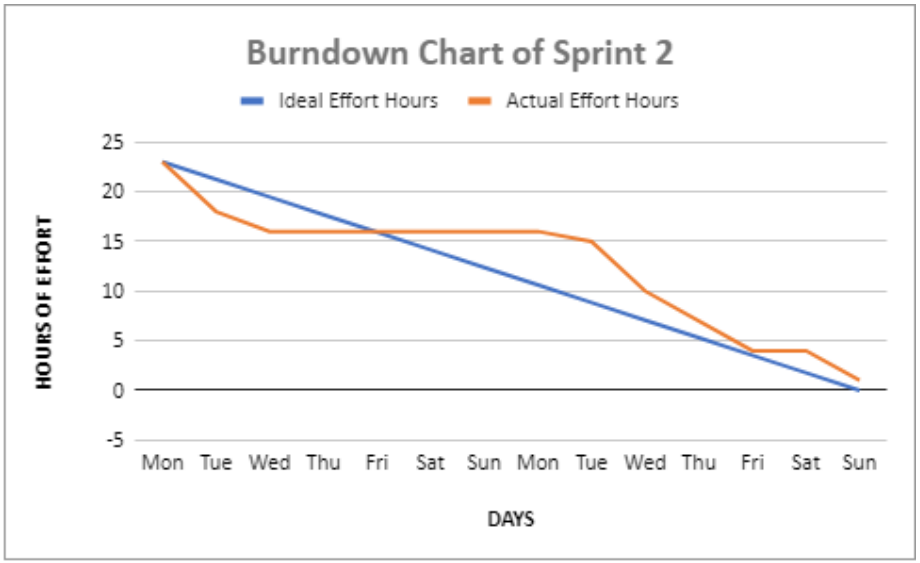
Sprint 8 - 3/8/2020 to 16/8/2020

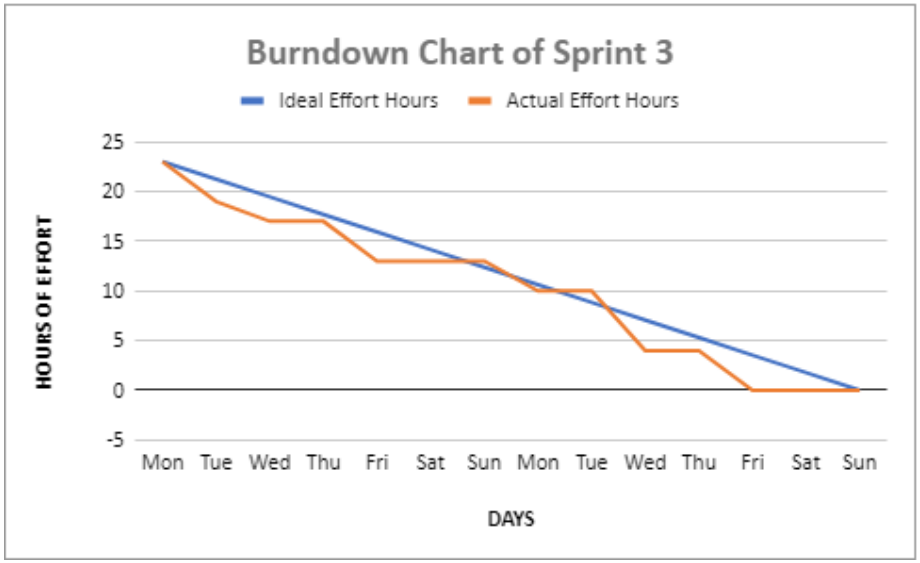
Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	0
2	Write the code for the project in Quest	Must	2.1	Implement the program logic flowchart as code in Quest	Must	Done	6	-	3	-	4	1	-	-	-	-	-	-	-	-	-	8	2
			2.2	Check and fix any possible bugs that may be hidden in the code	Should	Done	2	-	-	-	-	3	2	-	3	-	-	-	-	-	-	8	6
3	Write the report for the project	Must	3.1	Continue and finish writing up the report for the project	Must	Done	6	-	-	-	-	-	-	-	2	3	1	-	-	-	-	6	0
			3.2	Check for any grammar issues that may need fixing	Should	Done	1	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1	0
4	Produce a project presentation	Must	4.1	Plan out the project presentation	Must	Done	1	-	-	-	-	-	-	-	-	-	-	-	1	-	-	1	0
			4.2	Work on the project presentation	Must	Done	3	-	-	-	-	-	-	-	-	-	-	-	3	-	-	3	0
			4.3	Fix any last minute tweaks or adjustments that need doing	Should	Not Started	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-3
Ideal Effort Hours								23	21	19	18	16	14	12	11	9	7	5	4	2	0	This sprint lasted 5 hours longer than predicted.	
Actual Effort Hours								23	19	19	15	11	9	9	4	1	-1	-1	-5	-5	-5		

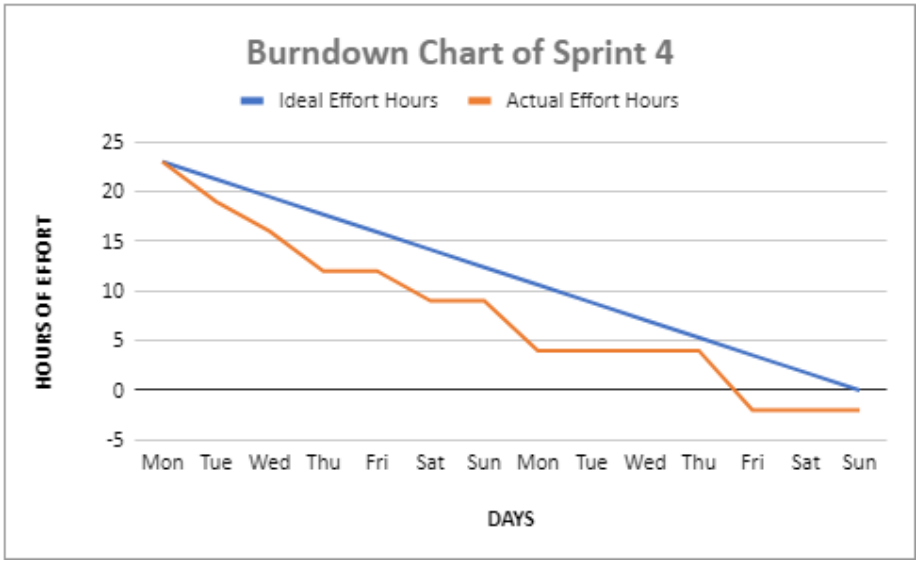
Sprint 9 - 17/8/2020 to 21/8/2020

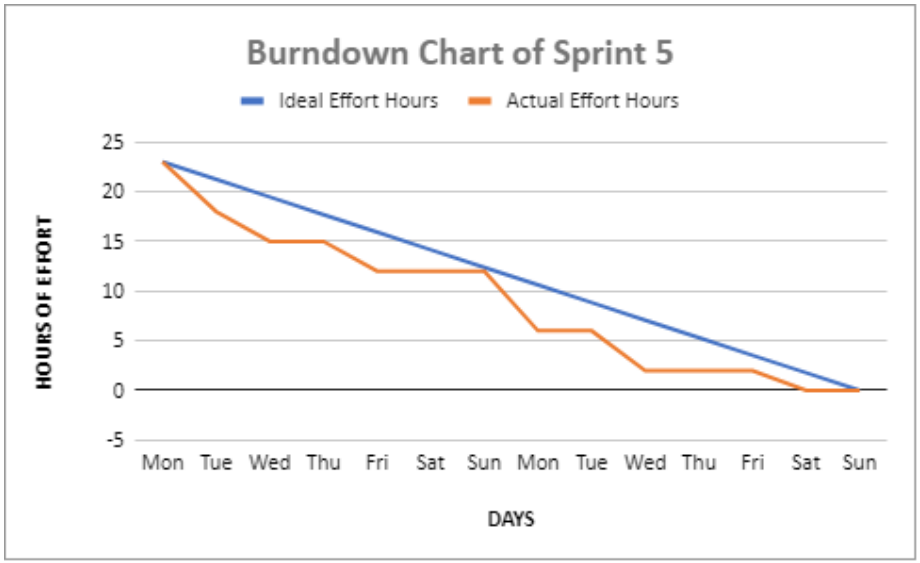
Breakdown of Deliverables								Week 1							Week 2							Hours	
Deliverable		Overall Priority	Individual Tasks		Individual Priorities	Progress	Estimated Hours	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Actual Hours	Difference in Hours
1	Iteration planning	Must	1.1	Detailed sprint planning	Must	Done	1	-	1	-	-	-										1	0
2	Finalise the project presentation	Must	2.1	Fix any last minute tweaks or adjustments that need doing	Should	Done	3	-	3	-	-	-										3	0
			2.2	Make sure everything is finalised and ready to be submitted	Should	Done	3	-	-	2	1	-										3	0
			2.3	Hand in the project	Must	Done	1	-	-	-	-	1										1	0
Ideal Effort Hours								8	6	4	2	0										This sprint lasted exactly as long as predicted.	
Actual Effort Hours								8	4	4	4	4											

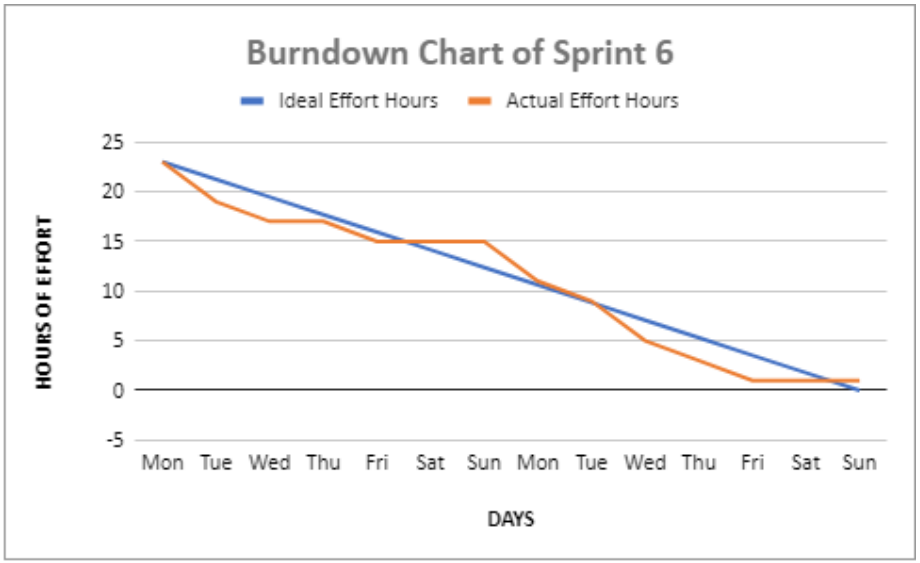

Burndown Chart of Sprint 1
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

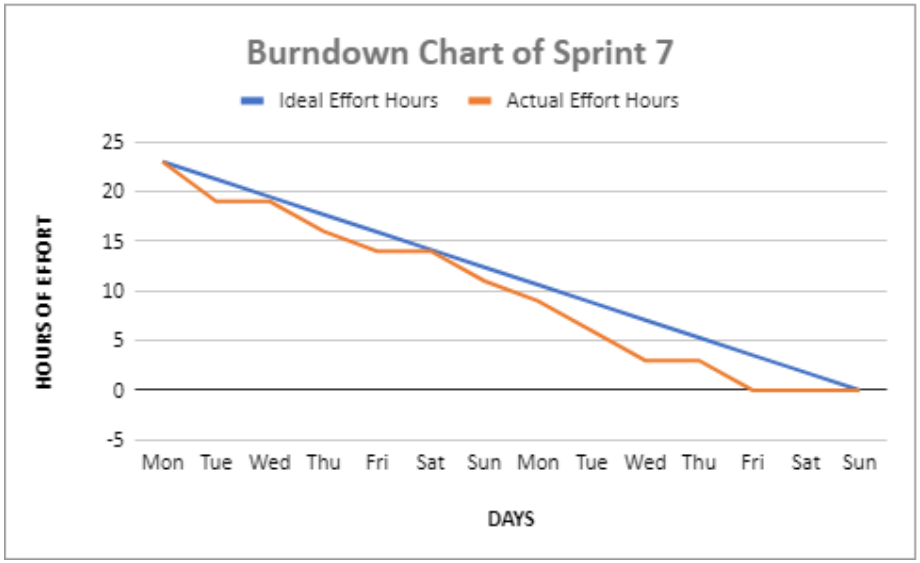

Burndown Chart of Sprint 2
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

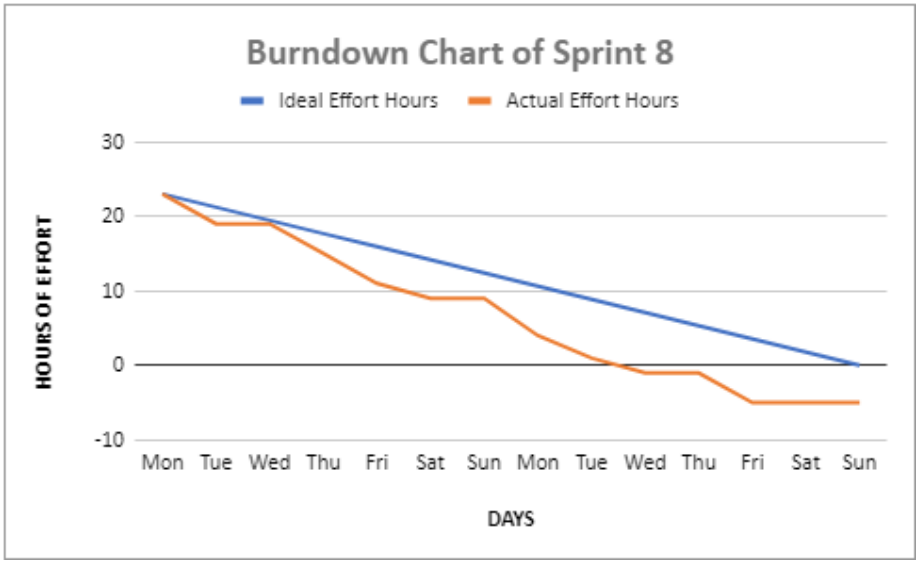

Burndown Chart of Sprint 3
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS


Burndown Chart of Sprint 4
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

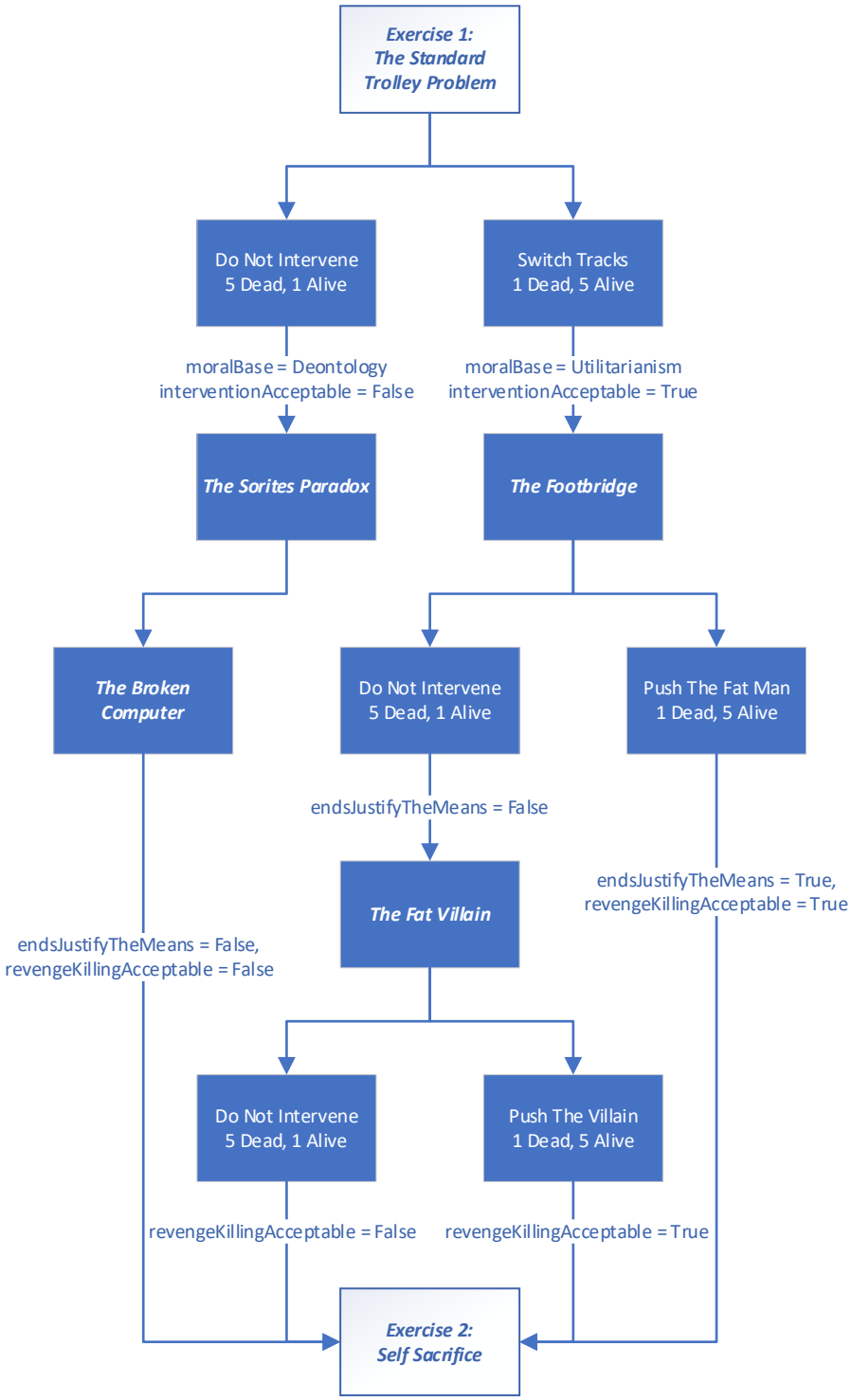

Burndown Chart of Sprint 5
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

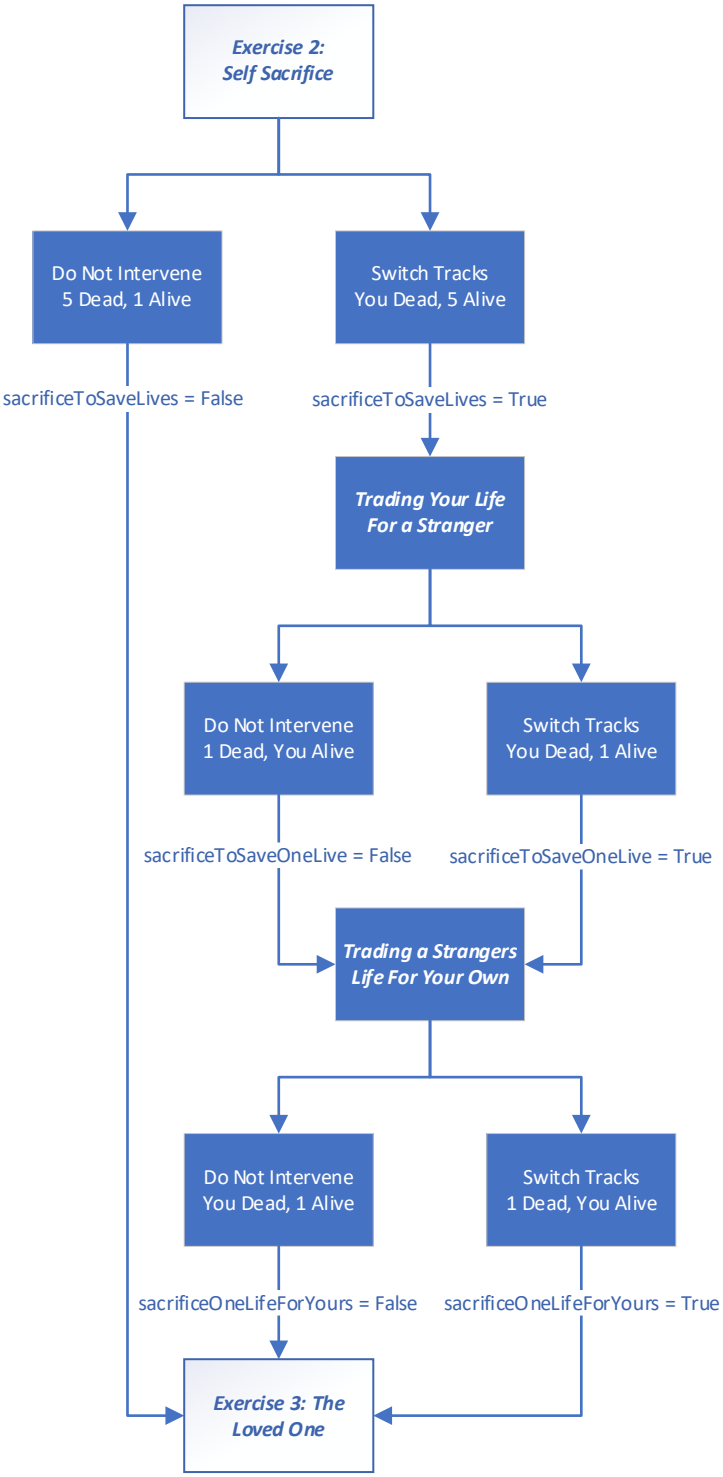

Burndown Chart of Sprint 6
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

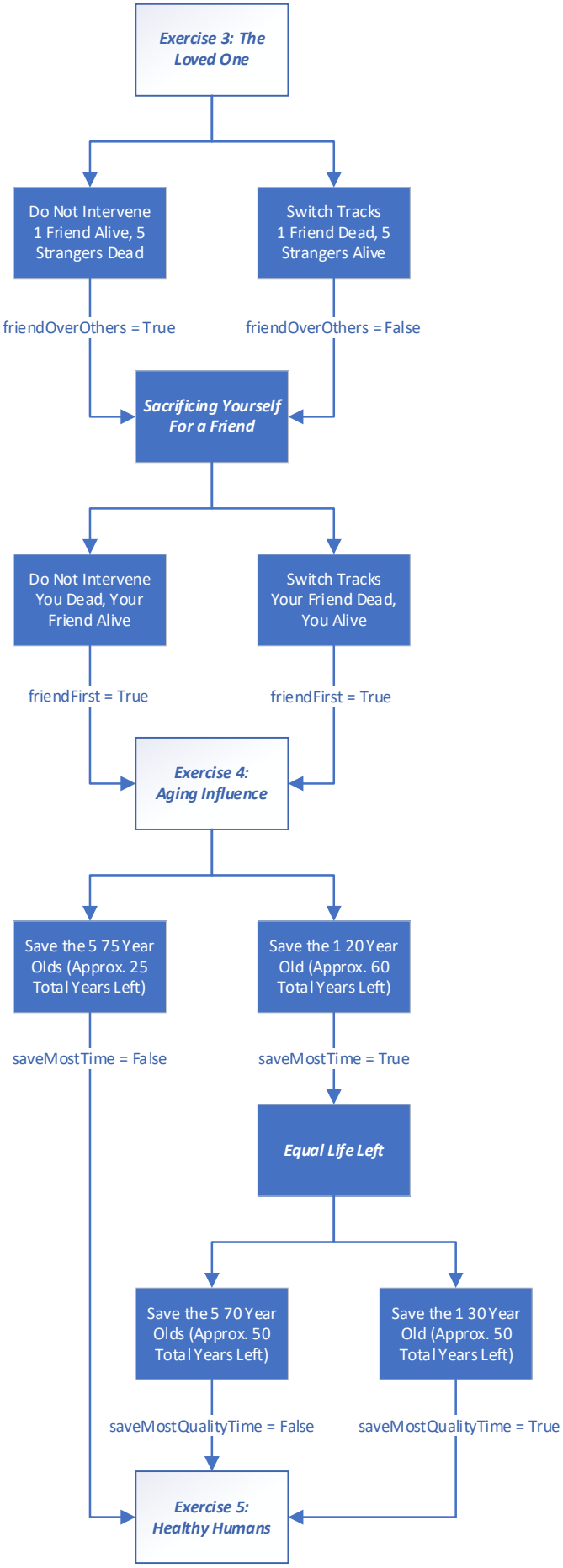

Burndown Chart of Sprint 7
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
25
20
15
10
5
0
-5
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

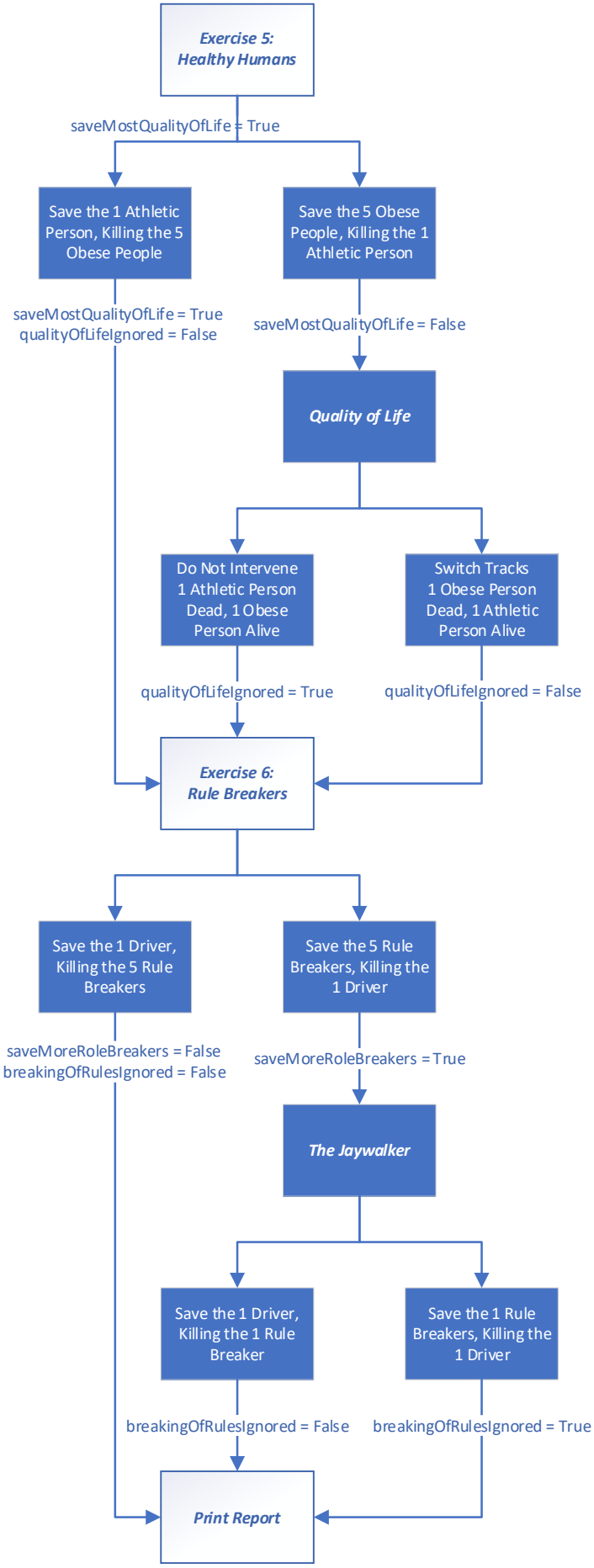

Burndown Chart of Sprint 8
Ideal Effort Hours
Actual Effort Hours
HOURS OF EFFORT
30
20
10
0
-10
Mon Tue Wed Thu Fri Sat Sun Mon Tue Wed Thu Fri Sat Sun
DAYS

Appendix B – Agile Sprint Planning




Exercise 2:
Self Sacrifice
Do Not Intervene
5 Dead, 1 Alive
Switch Tracks
You Dead, 5 Alive
sacrificeToSaveLives = False
sacrificeToSaveLives = True
Trading Your Life
For a Stranger
Do Not Intervene
1 Dead, You Alive
Switch Tracks
You Dead, 1 Alive
sacrificeToSaveOneLive = False
sacrificeToSaveOneLive = True
Trading a Strangers
Life For Your Own
Do Not Intervene
You Dead, 1 Alive
Switch Tracks
1 Dead, You Alive
sacrificeOneLifeForYours = False
sacrificeOneLifeForYours = True
Exercise 3: The
Loved One


Exercise 3: The Loved One
Do Not Intervene 1 Friend Alive, 5 Strangers Dead
Switch Tracks 1 Friend Dead, 5 Strangers Alive
friendOverOthers = True
friendOverOthers = False
Sacrificing Yourself For a Friend
Do Not Intervene You Dead, Your Friend Alive
Switch Tracks Your Friend Dead, You Alive
friendFirst = True
friendFirst = True
Exercise 4: Aging Influence
Save the 5 75 Year Olds (Approx. 25 Total Years Left)
Save the 1 20 Year Old (Approx. 60 Total Years Left)
saveMostTime = False
saveMostTime = True
Equal Life Left
Save the 5 70 Year Olds (Approx. 50 Total Years Left)
Save the 1 30 Year Old (Approx. 50 Total Years Left)
saveMostQualityTime = False
saveMostQualityTime = True
Exercise 5: Healthy Humans


Exercise 5: Healthy Humans
saveMostQualityOfLife = True
Save the 1 Athletic Person, Killing the 5 Obese People
Save the 5 Obese People, Killing the 1 Athletic Person
saveMostQualityOfLife = True
qualityOfLifeIgnored = False
saveMostQualityOfLife = False
Quality of Life
Do Not Intervene 1 Athletic Person Dead, 1 Obese Person Alive
Switch Tracks 1 Obese Person Dead, 1 Athletic Person Alive
qualityOfLifeIgnored = True
qualityOfLifeIgnored = False
Exercise 6: Rule Breakers
Save the 1 Driver, Killing the 5 Rule Breakers
Save the 5 Rule Breakers, Killing the 1 Driver
saveMoreRoleBreakers = False
breakingOfRulesIgnored = False
saveMoreRoleBreakers = True
The Jaywalker
Save the 1 Driver, Killing the 1 Rule Breaker
Save the 1 Rule Breakers, Killing the 1 Driver
breakingOfRulesIgnored = False
breakingOfRulesIgnored = True
Print Report

2 Java Code

The following Java tasks were created to showcase coding features such as functions, classes, and inheritance in an object-oriented programming structure to produce a program.

2.1 Validating Inputs

2.1.1 Main Class

```
public class Main {

    public static void main(String[] args) {

        int userInput = 0;
        boolean mainLoop = true;
        boolean validationCheck = false;
        while (mainLoop == true) {

            while (validationCheck == false) {

                System.out.print("Please enter an integer between 1 and 25: ");
                userInput = BIO.getInt();
                if (userInput >= 1 && userInput <= 25) {
                    System.out.println();
                    validationCheck = true;
                } else {
                    System.out.println("Invalid input detected.\n");
                }
            }

            for (int x = 1; x <= 12; x++) {
                System.out.println(x + " * " + userInput + " = " + (x * userInput));
            }

            validationCheck = false;
            System.out.println();
        }
    }
}
```

2.1.2 Test Outputs

```
Please enter an integer between 1 and 25: 1

1 * 1 = 1
2 * 1 = 2
3 * 1 = 3
4 * 1 = 4
5 * 1 = 5
6 * 1 = 6
7 * 1 = 7
8 * 1 = 8
9 * 1 = 9
10 * 1 = 10
11 * 1 = 11
12 * 1 = 12
```

```
Please enter an integer between 1 and 25: 1.1
Invalid input detected.

Please enter an integer between 1 and 25: 2

1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
6 * 2 = 12
7 * 2 = 14
8 * 2 = 16
9 * 2 = 18
10 * 2 = 20
11 * 2 = 22
12 * 2 = 24

Please enter an integer between 1 and 25: 26
Invalid input detected.

Please enter an integer between 1 and 25: 0
Invalid input detected.

Please enter an integer between 1 and 25: A
Invalid input detected.

Please enter an integer between 1 and 25:
```

2.2 Creating Bank Accounts

2.2.1 Account Class

```
package dataPackage;

class Account {

    private double theBalance = 0.00;
    private double theOverdraft = 0.00; //Overdraft allowed

    public double getBalance() {
        return theBalance;
    }

    public double withdraw(final double money) {
        assert money >= 0.00; //Cause error if money -ve
        if (theBalance - money >= theOverdraft) {
            theBalance = theBalance - money;
            return money;
        } else {
            return 0.00;
        }
    }
```

```
    public void deposit(final double money) {
        assert money >= 0.00; //Cause error if money -ve
        theBalance = theBalance + money;
    }

    public void setOverdraftLimit(final double money) {
        theOverdraft = money;
    }

    public double getOverdraftLimit() {
        return theOverdraft;
    }

}
```

2.2.2 AccountBetter1 Class

```
package dataPackage;

class AccountBetter1 extends Account implements Transfer {

    @Override
    public boolean transferFrom(Account from, double amount) {

        if ((from.getBalance() - amount >= 0) && (amount >= 0)) {
            from.withdraw(amount);
            deposit(amount);
            return true;
        } else {
            return false;
        }
    }

    @Override
    public boolean transferTo(Account to, double amount) {

        if ((getBalance() - amount >= 0) && (amount >= 0)) {
            withdraw(amount);
            to.deposit(amount);
            return true;
        } else {
            return false;
        }
    }

}
```

2.2.3 AccountBetter2 Class

```
package dataPackage;

class AccountBetter2 extends AccountBetter1 implements Interest {

    @Override
    public boolean inCredit() {

        if (getBalance() >= 0) {

            return true;

        } else {

            return false;
        }
    }

    @Override
    public void creditCharge() {

        if (getBalance() < 0) {

            if (getOverdraftLimit() > (getBalance() + (getBalance() * 0.00026116))) {

                setOverdraftLimit((getBalance() + (getBalance() * 0.00026116)));
            }
            withdraw(-(getBalance() * 0.00026116));
        }
        return;
    }

}
```

2.2.4 AccountStudent Class

```
package dataPackage;

class AccountStudent extends AccountBetter2 implements Interest {

    @Override
    public void creditCharge() {

        if (getBalance() < -5000) {

            if (getOverdraftLimit() > (getBalance() + (getBalance() * 0.00026116))) {

                setOverdraftLimit((getBalance() + (getBalance() * 0.00026116)));
            }
            withdraw(-(getBalance() * 0.00026116));
        }
        return;
    }

}
```

2.2.5 Main Class

```
// 1.3 Testing ------------------------------ Expected Results

System.out.println("\n1.3 Testing:");

AccountBetter1 ab = new AccountBetter1();
Account a = new Account();

ab.deposit(100.00);
System.out.println();
System.out.println("Ab = " + ab.getBalance());  // 100.00
System.out.println("A = " + a.getBalance());    // 0.00

a.deposit(50.00);
System.out.println();
System.out.println("Ab = " + ab.getBalance());  // 100.00
System.out.println("A = " + a.getBalance());    // 50.00

System.out.println();
System.out.println(ab.transferTo(a, 50.00));    // Returns True
System.out.println("Ab = " + ab.getBalance());  // 50.00
System.out.println("A = " + a.getBalance());    // 100.00

System.out.println();
System.out.println(ab.transferTo(a, 40.00));    // Returns True
System.out.println("Ab = " + ab.getBalance());  // 10.00
System.out.println("A = " + a.getBalance());    // 140.00

System.out.println();
System.out.println(ab.transferTo(a, -1.00));    // Returns False
System.out.println("Ab = " + ab.getBalance());  // 10.00
System.out.println("A = " + a.getBalance());    // 140.00

System.out.println();
System.out.println(ab.transferTo(a, 10.00));    // Returns True
System.out.println("Ab = " + ab.getBalance());  // 0.00
System.out.println("A = " + a.getBalance());    // 150.00

System.out.println();
System.out.println(ab.transferTo(a, 1.00));     // Returns False
System.out.println("Ab = " + ab.getBalance());  // 0.00
System.out.println("A = " + a.getBalance());    // 150.00
```

```
System.out.println();
System.out.println(ab.transferTo(a, -0.01));    // Returns False
System.out.println("Ab = " + ab.getBalance());  // 0.00
System.out.println("A = " + a.getBalance());    // 150.00

System.out.println();
System.out.println(ab.transferFrom(a, 50.00));  // Returns True
System.out.println("Ab = " + ab.getBalance());  // 50.00
System.out.println("A = " + a.getBalance());    // 100.00

System.out.println();
System.out.println(ab.transferFrom(a, 50.00));  // Returns True
System.out.println("Ab = " + ab.getBalance());  // 100.00
System.out.println("A = " + a.getBalance());    // 50.00

System.out.println();
System.out.println(ab.transferFrom(a, 40.00));  // Returns True
System.out.println("Ab = " + ab.getBalance());  // 140.00
System.out.println("A = " + a.getBalance());    // 10.00

System.out.println();
System.out.println(ab.transferFrom(a, -1.00));  // Returns False
System.out.println("Ab = " + ab.getBalance());  // 140.00
System.out.println("A = " + a.getBalance());    // 10.00

System.out.println();
System.out.println(ab.transferFrom(a, 10.00));  // Returns True
System.out.println("Ab = " + ab.getBalance());  // 150.00
System.out.println("A = " + a.getBalance());    // 0.00

System.out.println();
System.out.println(ab.transferFrom(a, 1.00));   // Returns False
System.out.println("Ab = " + ab.getBalance());  // 150.00
System.out.println("A = " + a.getBalance());    // 0.00

System.out.println();
System.out.println(ab.transferFrom(a, -0.01));  // Returns False
System.out.println("Ab = " + ab.getBalance());  // 150.00
System.out.println("A = " + a.getBalance());    // 0.00
```

```
// 1.4 Testing ------------------------------------------------- Expected Results

System.out.println("\n1.4 Testing:");

AccountBetter2 bob = new AccountBetter2();

bob.deposit(100.00);                                            // 100.00
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // 100.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // 0.00

bob.creditCharge();
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // 100.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // 0.00

System.out.println();
System.out.println(bob.inCredit());                             // Returns True
System.out.println("Bob (Balance) = " + bob.getBalance());      // 100.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // 0.00

bob.withdraw(100.00);
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // 0.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // 0.00

bob.creditCharge();
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // 0.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // 0.00

System.out.println();
System.out.println(bob.inCredit());                             // Returns True
System.out.println("Bob (Balance) = " + bob.getBalance());      // 0.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // 0.00

bob.setOverdraftLimit(-100.00);
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // 0.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -100.00

bob.creditCharge();
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // 0.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -100.00

System.out.println();
System.out.println(bob.inCredit());                             // Returns True
System.out.println("Bob (Balance) = " + bob.getBalance());      // 0.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -100.00

bob.withdraw(100.00);
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // -100.00
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -100.00

bob.creditCharge();
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());      // -100.03
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -100.03
```

```
System.out.println();
System.out.println(bob.inCredit());                                    // Returns False
System.out.println("Bob (Balance) = " + bob.getBalance());             // -100.03
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -100.03

bob.setOverdraftLimit(-1000.00);
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());             // -100.03
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -1000.00

System.out.println();
System.out.println(bob.inCredit());                                    // Returns False
System.out.println("Bob (Balance) = " + bob.getBalance());             // -100.03
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -1000.00

bob.creditCharge();
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());             // 100.05
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -1000.00

bob.withdraw(899.00);
System.out.println();
System.out.println("Bob (Balance) = " + bob.getBalance());             // -999.05
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -1000.00

System.out.println();
System.out.println(bob.inCredit());                                    // Returns False
System.out.println("Bob (Balance) = " + bob.getBalance());             // -999.05
System.out.println("Bob (Overdraft) = " + bob.getOverdraftLimit()); // -1000.00
```

```java
// 1.5 Testing ------------------------------------------------------------ Expected Results

System.out.println("\n1.5 Testing:");

AccountStudent calvin = new AccountStudent();

calvin.deposit(100.00);                                                    // 100.00
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // 0.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // 0.00

System.out.println();
System.out.println(calvin.inCredit());                                     // Returns True
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // 0.00

calvin.withdraw(100.00);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 0.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // 0.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 0.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // 0.00

System.out.println();
System.out.println(calvin.inCredit());                                     // Returns True
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 0.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // 0.00

calvin.setOverdraftLimit(-100.00);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 0.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // -100.00

System.out.println();
System.out.println(calvin.inCredit());                                     // Returns True
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // 0.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // -100.00

calvin.withdraw(100.00);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // -100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // -100.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // -100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // -100.00

System.out.println();
System.out.println(calvin.inCredit());                                     // Returns False
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // -100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // -100.00

calvin.setOverdraftLimit(-5010.00);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());           // -100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());  // -5010.00
```

```
calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -100.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.withdraw(4899.99);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -4999.99
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

System.out.println();
System.out.println(calvin.inCredit());                                      // Returns False
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -4999.99
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.withdraw(0.01);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5000.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5000.00
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.withdraw(0.01);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5000.01
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5001.32
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.withdraw(0.01);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5001.33
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

System.out.println();
System.out.println(calvin.inCredit());                                      // Returns False
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5001.33
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -5002.63
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.deposit(2.64);
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -4999.99
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

System.out.println();
System.out.println(calvin.inCredit());                                      // Returns False
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -4999.99
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -4999.99
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00

calvin.creditCharge();
System.out.println();
System.out.println("Calvin (Balance) = " + calvin.getBalance());            // -4999.99
System.out.println("Calvin (Overdraft) = " + calvin.getOverdraftLimit());   // -5010.00
```

2.2.6 Interest Interface

```
package dataPackage;

interface Interest {

    boolean inCredit(); //Is account in credit
    void creditCharge(); //Add credit charge (Daily)
}
```

2.2.7 Transfer Interface

```
package dataPackage;

interface Transfer {

    public boolean transferFrom(Account from, double amount);
    public boolean transferTo(Account to, double amount);
}
```

2.2.8 Testing Outputs

```
Mike = 100.00
Miri = 90.00
Cori = 110.00

1.3 Testing:

Ab = 100.00
A = 0.00

Ab = 100.00
A = 50.00

true
Ab = 50.00
A = 100.00

true
Ab = 10.00
A = 140.00

false
Ab = 10.00
A = 140.00

true
Ab = 0.00
A = 150.00

false
Ab = 0.00
A = 150.00

false
Ab = 0.00
A = 150.00

true
Ab = 50.00
A = 100.00

true
Ab = 100.00
A = 50.00

true
Ab = 140.00
A = 10.00

false
Ab = 140.00
A = 10.00

true
Ab = 150.00
A = 0.00

false
Ab = 150.00
A = 0.00

false
Ab = 150.00
A = 0.00
```

```
1.4 Testing:

Bob (Balance) = 100.00
Bob (Overdraft) = 0.00

Bob (Balance) = 100.00
Bob (Overdraft) = 0.00

true
Bob (Balance) = 100.00
Bob (Overdraft) = 0.00

Bob (Balance) = 0.00
Bob (Overdraft) = 0.00

Bob (Balance) = 0.00
Bob (Overdraft) = 0.00

true
Bob (Balance) = 0.00
Bob (Overdraft) = 0.00

Bob (Balance) = 0.00
Bob (Overdraft) = -100.00

Bob (Balance) = 0.00
Bob (Overdraft) = -100.00

true
Bob (Balance) = 0.00
Bob (Overdraft) = -100.00

Bob (Balance) = -100.00
Bob (Overdraft) = -100.00

Bob (Balance) = -100.03
Bob (Overdraft) = -100.03

false
Bob (Balance) = -100.03
Bob (Overdraft) = -100.03

Bob (Balance) = -100.03
Bob (Overdraft) = -100.03

false
Bob (Balance) = -100.03
Bob (Overdraft) = -1000.00

Bob (Balance) = -100.05
Bob (Overdraft) = -1000.00

Bob (Balance) = -999.05
Bob (Overdraft) = -1000.00

false
Bob (Balance) = -999.05
Bob (Overdraft) = -1000.00
```

```
1.5 Testing:

Calvin (Balance) = 100.00
Calvin (Overdraft) = 0.00

Calvin (Balance) = 100.00
Calvin (Overdraft) = 0.00

true
Calvin (Balance) = 100.00
Calvin (Overdraft) = 0.00

Calvin (Balance) = 0.00
Calvin (Overdraft) = 0.00

Calvin (Balance) = 0.00
Calvin (Overdraft) = 0.00

true
Calvin (Balance) = 0.00
Calvin (Overdraft) = 0.00

Calvin (Balance) = 0.00
Calvin (Overdraft) = -100.00

Calvin (Balance) = 0.00
Calvin (Overdraft) = -100.00

true
Calvin (Balance) = 0.00
Calvin (Overdraft) = -100.00

Calvin (Balance) = -100.00
Calvin (Overdraft) = -100.00

Calvin (Balance) = -100.00
Calvin (Overdraft) = -100.00

false
Calvin (Balance) = -100.00
Calvin (Overdraft) = -100.00

Calvin (Balance) = -100.00
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -100.00
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -4999.99
Calvin (Overdraft) = -5010.00

false
Calvin (Balance) = -4999.99
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -5000.00
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -5000.00
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -5000.01
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -5001.32
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -5001.33
Calvin (Overdraft) = -5010.00

false
Calvin (Balance) = -5001.33
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -5002.63
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -4999.99
Calvin (Overdraft) = -5010.00

false
Calvin (Balance) = -4999.99
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -4999.99
Calvin (Overdraft) = -5010.00

Calvin (Balance) = -4999.99
Calvin (Overdraft) = -5010.00
```

2.3 Eight Queens Solution

The first action I took when completing this task was creating the guidelines that I needed to follow in order to adhere to the specification. This included having an input for the user to choose the first queen location, a good physical representation of each queen's location and the ability to remove previous queens if needed by backtracking. I specifically choose to use the backtracking algorithm as I wanted my code to be as fast and efficient as possible, and after developing various ways of making this happen, I believed this to be the best way to achieve this goal. In order to successfully deduce which was the next available space to place a queen, I created three rules for my program to follow:

- Queens cannot be located in the same column.
- Queens cannot be located in the same row.
- Queens cannot be located in the same diagonal.

I stored the column and row values of each queen, so that I could iterate through each column until possible locations are found. If at any point a location being searched is invalid, the program breaks out of that section of code and skips straight to searching the next space. Once no more queens can be placed, the previous queen is removed. The program then searches the next available space until all queens can be placed. By keeping a record of all previous queens and only backtracking by one instead of starting from scratch, the time it takes to produce a solution is significantly reduced.

To help myself and anyone else reading my code, I added comments for all the major function in the program. This allows me to showcase what each step of the program does, making it easier for me to see which sections of code I needed to focus on when trying to debug my program. One main issue I had when designing my code when approaching this task was looping the search back to the start of the board when the program had reached the final column. I originally wrote the program to only search until the final column, however I did not account for the fact users may decide not to place the queen in the first column; detecting queens to the left diagonally took some extra thinking.

I decided the best way to target the problem was to split the original third rule into four separates rules, by search the upper left, lower left, lower right, and upper right diagonals instead. One simple way I could utilise this in the future to further the efficiency of my program could be to detect which corner the current location is closed to and starting with the corresponding diagonal. For example, if a location near the top left-hand corner is being searched, there is a more likely chance of a queen being in the bottom right diagonal due to the increased number of spaces compared to the top left.

There are a few additional design choices I could add if I were to attempt this task again in the future. For example, to allow for every possibility once the user has inputted the first queen's location, you could continue to backtrack until no more possible locations are left. This would be done by printing out each solution once the program has placed all eight queens, but then continue to backtrack the final queen each time a solution is found. Once every solution is discovered, the program would then terminate after printing out how many solutions had been found.

Another change I could make to the program could be to all the user to choose how many queens the user would want to place or make the size of the board a variable that the user could change. Ultimately, I decided to focus all my efforts on following the guidelines set out for me in order to achieve a comprehensive solution in the allotted time. Overall, I am very satisfied with the result, due to the efficiency based on the algorithm I chose to use and the overall design of the program.

2.3.1 Main Class

This program is designed to solve the eight queen's problem using backtracking. The program begins with the user inputting the column and row of the first queen. The program then calculates all of the resulting board positions for the remaining queens. If it reaches the end without finding all 8 queens, the previous queen is removed. A new location is then considered, and the program moves forward again. For each queen, the side is checked, followed by an upper left diagonal check; The remaining queens are then completed in an anti-clockwise fashion until all 8 queens have been placed.

```
package dataPackage;

import java.util.Scanner;

public class Main {

    // Prints a visual representation of each queen location to the console.

    void printSolution(int board[][]) {

        System.out.println("\n/ A B C D E F G H");

        for (int i = 0; i < 8; i++) {
            System.out.print(8 - i + " ");

            for (int j = 0; j < 8; j++) {
                System.out.print(board[i][j] + " ");
            }

        System.out.println();
    }
}

// Checks for the next possible queen location, based on the previous ones placed.

boolean validLocation(int board[][], int row, int column) {

    int i, j, counter;

    // Checks to see if a queen is located on the same row

    for (i = column, counter = 0; counter < 8; i++, counter++) {

        if (i >= 8) {
            i -= 8;
        }

        if (board[row][i] == 1) {

            System.out.println("Column = " + (column+1) + ", Row = " + (row+1) + "
            failed, already a queen to the side.");
            return false;
        }
    }

    // Checks to see if a queen is located on the upper left diagonal.

    for (i = row, j = column; i >= 0 && j >= 0; i--, j--) {

        if (board[i][j] == 1) {
```

```
            System.out.println("Column = " + (column+1) + ", Row = " + (row+1) + "
            failed, already a queen in the upper left diagonal.");
            return false;
        }
    }

    // Checks to see if a queen is located on the lower left diagonal

    for (i = row, j = column; j < 8 && i >= 0; i--, j++) {

        if (board[i][j] == 1) {

            System.out.println("Column = " + (column+1) + ", Row = " + (row+1) + "
    failed, already a queen in the lower left diagonal.");
            return false;
        }
    }

    // Checks to see if a queen is located on the lower right diagonal

    for (i = row, j = column; i < 8 && j < 8; i++, j++) {

        if (board[i][j] == 1) {

            System.out.println("Column = " + (column+1) + ", Row = " + (row+1) + "
            failed, already a queen in the lower right diagonal.");
            return false;
        }
    }

    // Checks to see if a queen is located on the upper right diagonal

    for (i = row, j = column; j >= 0 && i < 8; i++, j--) {

        if (board[i][j] == 1) {

            System.out.println("Column = " + (column+1) + ", Row = " + (row+1) + "
            failed, already a queen in the upper right diagonal.");
            return false;
        }
    }

    System.out.println("Column = " + (column + 1) + ", Row = " + (row + 1) + "
    placed.");
    return true;
}

boolean backtrackFunction(int board[][], int column, int userQueenRow, int placed) {

    // This stops the continuous loop if all 8 queens have been placed on the board.

    if (placed >= 8) {
        System.out.println("Total Queens Placed: " + placed);
        return true;
    }

    int y = column;
    if (column >= 8) {
        y = column - 8;
    }
```

```
    System.out.println("Total Queens Placed: " + placed);

    // Tries placing a queen in this column by searching all the rows for locations.

    for (int i = userQueenRow; i < userQueenRow + 8; i++) {

    // If i goes over 7, this allows the program to loop back to the beginning.

        int x = i;

        if (i >= 8) {
            x = i - 8;
        }

        int xPrinted = x + 1;
        int yPrinted = y + 1;

        // If a location is valid, this places a queen down in that spot.

        if (validLocation(board, x, y)) {

            board[x][y] = 1;
            placed += 1;

            // This allows the rest of the queens to be placed.

            if (backtrackFunction(board, yPrinted, xPrinted, placed) == true) {
                return true;
            }

            // If it reaches the end without all queens, backtrack to remove the last one.

            System.out.println("Column = " + (yPrinted) + ", Row = " + (xPrinted) + "
            removed.");
            board[x][y] = 0;
            placed -= 1;
        }
    }
    return false;
}

boolean solveNQ(int userQueenColumn, int userQueenRow, int placed) {

    int board[][] = {{0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0}};

    if (backtrackFunction(board, userQueenColumn, userQueenRow, placed) == false) {

        System.out.print("Error, no solution detected.");
        return false;
    }

    printSolution(board);
    return true;
}
```

```
public static void main(String args[]) {

    // This allows the user to input the first location of the queen.

    Scanner reader = new Scanner(System.in);
    System.out.print("Enter the column number of your queen: ");
    int userQueenColumn = reader.nextInt() - 1;
    System.out.print("Enter the row number of your queen: ");
    int userQueenRow = reader.nextInt() - 1;
    reader.close();
    int placed = 0;

    // This starts a timer to see how long it takes to produce a viable result.

    long tStart = System.currentTimeMillis();

    // Runs the program to produce a solution based on the first queen location.

    Main NQueensProgram = new Main();
    NQueensProgram.solveNQ(userQueenColumn, userQueenRow, placed);

    // This ends the previously mentioned timer.

    long tEnd = System.currentTimeMillis();
    long tDelta = tEnd - tStart;

    // This works out how long it has taken to produce a viable result.

    System.out.println("\nTime taken to produce a solution: " + tDelta + "
    milliseconds");
    }
}
```

2.4 Priority Ticketing System

For this task I have decided to use linked lists, a dynamic data structure that can grow and shrink during a program's runtime, which is done by allocating and deallocating memory. There is no need to give an initial size to the linked list, so there is no memory wastage as the list size can change as memory is only allocated when it is required. One potential downside to this is that more memory is required to store the elements themselves due to each node containing a pointer. If I were to use an array, however, then there could be a lot of memory wastage if the array is too large for the number of elements stored instead it. Since an IT ticketing system will not have a set number of tickets at any given time, the advantage from the linked list is greater than the downside of using more memory for the pointers, as less memory is usually required overall, provided the code is written efficiently.

Inserting and deleting nodes using linked lists was very easy to do, as unlike an array I did not need to shift the elements after the insertion or deletion of an element. All I needed to do with the linked list was update the address that is present in the next pointer of a node. This makes developing data structures such as stack and queues easily to implement using linked list. Traversing back through a linked list can be difficult to implement, however, without the use of a doubly linked list. If I were to implement a doubly linked list, then a great deal of extra memory would be required, resulting in a waste of memory. While a doubly linked list might have been more useful for other applications, the benefits did not outweigh the extra memory required, so I decided to stick with a singly linked list.

2.4.1 Main Class

This class simulates a ticket queueing system that sorts out submitted tickets by priority. Each ticket contains a unique ID, a description of the issue, a creator, an owner (the person handling the ticket), and a priority. This ranges from 1 (most important) to 4 (least important), with higher priority tickets being dealt with before less important issues. Once a ticket is resolved, it is then removed from the ticket system, where the next more important ticket is pushed to the top.

```
package dataPackage;

public class Main {

    public static void main(String[] args) {

        PriorityQueue ticketQueue = new PriorityQueue();

        Ticket ticketJay = new Ticket(1, "I think I have a virus, my PC's so slow since
downloading a file.", "Jay Massey", "Adam Tyler", 1);
        Ticket ticketBill = new Ticket(2, "A network issue is not letting me upload any of
my work.", "Bill Woods", "Reece Tennant", 2);
        Ticket ticketHenry = new Ticket(3, "This software isn't working since the new
update.", "Henry Adams", "Adam Tyler", 3);
        Ticket ticketSteve = new Ticket(4, "My new computer has arrived and needs setting
up.", "Steve Banks", "Reece Tennant", 4);
        Ticket ticketJoe = new Ticket(5, "The new maintenance guy doesn't know what he's
doing, can you help?", "Joe Mayes", "Adam Tyler", 4);
        Ticket ticketDave = new Ticket(6, "My new mouse isn't working, can't I just get a
new one?", "Dave Turner", "Reece Tennant", 4);
        Ticket ticketAmy = new Ticket(7, "I think someone is logging onto my computer
remotely.", "Amy Hills", "Reece Tennant", 1);
```

```
        // Inserting example tickets into the system.

        ticketQueue.insert(ticketJay);      //
        ticketQueue.insert(ticketBill);     // These are inserted into the ticket queue,
        ticketQueue.insert(ticketHenry);    // where they are sorted based on the selected
        ticketQueue.insert(ticketSteve);    // priority level, with the higher priority
        ticketQueue.insert(ticketJoe);      // tickets placed right at the top.
        ticketQueue.insert(ticketDave);     //

        // Displaying all the tickets in the system.

        ticketQueue.displayAll(); // Displays the unsolved tickets currently queued.

        // Displaying and removing the most important ticket in the queue.

        ticketQueue.displayTop(); // Displays the top ticket in the queue.
        ticketQueue.removeTop(); // Removes the top ticket from the queue.

        // Removing a ticket from the queue, using the unique ID that corresponds to it.

        ticketQueue.removeTicket(4); // Remove the ticket with the corresponding ID.

        // Search for a ticket in the queue, using the unique ID that corresponds to it.

        ticketQueue.searchTicket(3); // Searches for the ticket with the corresponding ID.

        // Displaying and removing the most important ticket in the queue.

        ticketQueue.displayTop();
        ticketQueue.removeTop();

        // Changing the priority of an already existing ticket.

        ticketQueue.displayTop();
        ticketQueue.displayAll();
        ticketQueue.changePriority(5, "That new guy was awful, he's made my PC situation
worse by somehow disabling my network.", 2);
        ticketQueue.displayTop();
        ticketQueue.displayAll();

        // Inserting a new ticket into the system, placed then based on its priority.
        ticketQueue.insert(ticketAmy);
        ticketQueue.displayTop();
        ticketQueue.displayAll();
    }
}
```

2.4.2 Priority Class

This class merges the 4 individual queues to create a priority queue.

```
package dataPackage;

public class PriorityQueue {

    private Queue[] queues;  // This makes the variable accessible only inside this class.

    public PriorityQueue() {

        queues = new Queue[4];       // Creates 4 queues with the class Queue.
        queues[0] = new Queue();     // Creates a queue for priority 1 tickets.
        queues[1] = new Queue();     // Creates a queue for priority 2 tickets.
        queues[2] = new Queue();     // Creates a queue for priority 3 tickets.
        queues[3] = new Queue();     // Creates a queue for priority 4 tickets.
    }

    public void changePriority(int id, String newDesc, int newPrior) {

        for(int q = 0; q < 4; q++) {

            if(queues[q].ticketPresent()) { // If the current queue is not empty.

                Ticket change = queues[q].changePriority(id, newDesc, newPrior);
                if (change != null) {

                    Ticket newTicket = new Ticket(change.getID(), change.getDescription(),
change.getCreator(), change.getHandler(), change.getPriority()); // This is done so only
the first ticket in "change" is inserted back in the queue.
                    this.insert(newTicket);
                    break;
                }
            }
        }
    }

    public void displayAll() {

        System.out.println("\nAll Remaining Tickets: (" + queueLength() + ")\n");

        for(int q = 0; q < 4; q++) {
            if(queues[q].ticketPresent()) {
                queues[q].displayAll();     // This displays all the tickets in the queue.
            }
        }
    }

    public void displayTop() {

    System.out.println("\nNext Ticket:");

    for(int q = 0; q < 4; q++) {
        if(queues[q].ticketPresent()) {
            queues[q].displayTop();         // Causes top queued ticket to display.
            break; // Stops more tickets being printed.
        }
    }
}
```

```
public void insert(Ticket ticket) {

    switch(ticket.getPriority()) {          // Finds priority to determine queue.

        case 1:
            queues[0].insert(ticket);       // Inserts a new priority 1 ticket.
            break;
        case 2:
            queues[1].insert(ticket);       // Inserts a new priority 2 ticket.
            break;
        case 3:
            queues[2].insert(ticket);       // Inserts a new priority 3 ticket.
            break;
        case 4:
            queues[3].insert(ticket);       // Inserts a new priority 4 ticket.
            break;
        }
    }

    public int queueLength() {              // Finds the number of tickets in ticket queue.

        int length = 0;

        for(int q = 0; q < 4; q++) {
            if(queues[q].ticketPresent()) {
                length += queues[q].ticketCycle();        // Adds 1 for each queued ticket.
            }
        }
        return length;       // Returns the total number of tickets in the entire system.
    }

    public void removeTicket(int removeID) {

        for(int q = 0; q < 4; q++) {
            if(queues[q].ticketPresent()) {               // The 0 in this function tells
                queues[q].removeTicket(removeID, 0);      // the system to print that the
            }                                             // ticket has been removed.
        }
    }

    public void removeTop() {

        for(int q = 0; q < 4; q++) {
            if(queues[q].ticketPresent()) {
                queues[q].removeTop();
                break;
            }
        }
    }

    public void searchTicket(int searchID) {

        System.out.println("\nSearching for Ticket " + searchID + ":");

        for(int q = 0; q < 4; q++) {
            if(queues[q].ticketPresent()) {
                queues[q].searchTicket(searchID);  // Checks for the user requested ticket.
            }
        }
    }
}
```

2.4.3 Queue Class

This class constructs a queueing system, adding values to the end and removes items from the front.

```
package dataPackage;

public class Queue {

    private Ticket head;
    private Ticket tail;

    public Queue() {

        head = null;
        tail = null;
    }

    public Ticket changePriority(int inputedID, String newDesc, int newPrior) {

        Ticket current = head;

        while (current != null) {    // Loops until the end of the list.

            if (inputedID == current.getID()) {

                current.changePriority(newDesc, newPrior);        // Updates the ticket.
                System.out.println("\nTicket " + inputedID + " has been updated to
priority " + newPrior + ".");
                removeTicket(current.getID(), 1);
                return current;

            } else {

                current = current.getNext();       // Updates ticket to next in the list.
            }
        }

        return null;
    }

    public void displayAll() {

        Ticket current = head;

        while (current != null) {           // Loops until the end of the list.

            current.displayTicket();        // Displays current ticket in the list.
            current = current.getNext();
        }
    }

    public void displayTop() {

        head.displayTicket();
    }
```

```
public void insert(Ticket ticket) {

    if(head == null && tail == null) {         // If queue is empty, execute this code.

        head = ticket;                         // Sets the head to the inputted ticket.
        tail = ticket;                         // Sets the tail to the inputted ticket.

    } else {

        tail.setNext(ticket);                  // Sets the next ticket in the tail.
        tail = ticket;                         // Sets the tail to the inputted ticket.
    }
}

public boolean isEmpty() {

    return head == null;                       // Checks if the ticket queue is empty.
}

public Ticket removeTicket(int removeID, int removeType) {

    Ticket current = head;

    if (current.getID() == removeID) {

        if (current.getNext() == null) {

            head = null;
            tail = null;
            return current;

        } else {

            if (removeType == 0) {             // that the ticket has been removed.

                System.out.println("\nTicket " + current.getID() + " removed.");
            }

            head = current.getNext();
            return current;
        }
    }

    while (current.getNext() != null) {        // Loops until the end of the list.

        int checkID = current.getNext().getID();

        if (checkID == removeID) {

            current.setNext(current.getNext().getNext());
            System.out.println("\nTicket " + checkID + " removed.");
            return current;

        } else {

            current = current.getNext();
        }
    }

    return current;
}
```

```
    public Ticket removeTop() {

        Ticket ticket = head;
        System.out.println("\nTicket " + ticket.getID() + " completed." );

        if(head == tail) {          // If one ticket only, set both head and tail to null.

            head = null;
            tail = null;

        } else {

            head = head.getNext();
        }

        return ticket;
    }

    public Ticket searchTicket(int inputedID) {

        Ticket current = head;

        while (current != null) {   // Loops until the end of the list.

            if (inputedID == current.getID()) {

                current.displayTicket();
                return current;

            } else {

                current = current.getNext();
            }
        }

        return null;
    }

    public int ticketCycle() {

        int cycleLength = 0;
        Ticket current = head;

        while (current != null) {   // Loops until the end of the list.

            current = current.getNext();
            cycleLength += 1;
        }

        return cycleLength;         // Returns how many tickets are in a queue.
    }

    public boolean ticketPresent() {

        return head != null;        // Checks if there is a ticket in the queue.
    }
}
```

2.4.4 Ticket Class

This class is used to fetch data relating to each ticket processed in the system.

```
package dataPackage;

public class Ticket {

    private int ID; // A unique ID for the ticket.
    private String description; // A description of the problem.
    private String creator; // The creator of the problem.
    private String handler; // The handler of the problem.
    private int priority; // The priority of the ticket.
    private Ticket next; // The next ticket pointer.

    public Ticket(int id, String desc, String create, String handle, int rank) {
        ID = id;
        description = desc;
        creator = create;
        handler = handle;
        priority = rank;
    }

    public void changePriority(String newDesc, int newPrior) { // Changes ticket priority.
        description = newDesc; priority = newPrior;
    }

    public void displayTicket() { // This is the displayed structure for each ticket.
        System.out.printf("Ticket ID: " + ID + ", Description: '" + description + "',
Creator: " + creator + ", Handler: " + handler + ", Priority: " + priority + "\n");
    }

    public String getCreator() {
        return creator; // Returns the name of the person submitting the ticket.
    }

    public String getDescription() {
        return description; // Returns the ticket description.
    }

    public String getHandler() {
        return handler; // Returns the name of the person dealing with the ticket.
    }

    public int getID() {
        return ID; // Returns the ticket ID.
    }

    public Ticket getNext() {
        return next; // Returns the next ticket pointer.
    }

    public int getPriority() {
        return priority; // Returns the priority of the ticket.
    }

    public void setNext(Ticket ticket) {
        next = ticket; // Sets the next ticket pointer.
    }
}
```

2.5 Designing a Process Scheduler

2.5.1 Research

In order to create an effective process scheduler, I initially decided to work out the most efficient algorithm for the task. The only requirement was that the scheduler needed the ability to prioritise tasks to be one of three levels on initialisation, however these priorities could later be changed to avoid task stagnation if needed. I started to familiarise myself with many algorithms such as First Come First Serve, Shortest Job First, and Priority Scheduling, in order to determine what the most efficient algorithm for the task was. While many of the algorithms I researched worked well in certain scenarios, I felt like they did not achieve the efficiency I was looking for. After trying to weigh the varying pros and cons of each of the algorithms against one another, I decided a better approach would be to list the variables I needed to consider in order to make the most efficient scheduler possible, each of which are listed below:

- Completion Time - The time that a particular process is completed.
- Turnaround Time - The completion time subtracted by the arrival time.
- Waiting Time - The amount of time a process is waiting in order to be processed.
- Response Time - The time it takes from a process arriving until the first response.

These variables needed to be as low as possible, in order for the following to be as high as possible:

- Throughput - The number of processes completed per time unit (seconds, minutes, etc).

These are the factors I kept in mind when designing my process scheduler. I used Priority Scheduling as a base algorithm but created an aging system that added points to all processes that have not yet been completed. When a process initially arrives, its age is tracked until it has been fully processed. When a process has an age greater than a set limit, the number of points attributed to it means that it will have reached the top of the priority queue. This limit was not set in stone; rather it was calculated using the average burst time of the completed processes. This meant that whilst I was not able to use actual bursts in the calculations, I was able to use an estimate based on the real data that was being passed through the scheduler.

The more processes that arrived in the system, the more data it had to work out what the average burst time is, and therefore the more efficient the scheduler became. This was particularly helpful in working out the order of processes when longer processes entered the system. If a process was taking longer than the average burst time, the system knew that it must be very close to competition and therefore it continued to be processed for the final few bursts, rather than switching to a new process. This prevented long process starvation, as slightly longer processes were allowed to finish being processed when previously they were not. This would not have been the case if I had simply used default priority scheduling, which would have reduced the efficiency.

These solutions decrease the average waiting and turnaround time, resulting in processes being completed quicker than they were before. Whilst the brief only required hard coding 5 processes into the system as it was mainly testing my algorithm application, the way it works would allow it to be expanded to an infinite number of processes. If I were to reattempt this task in the future, I would change the processes to be dynamic rather than only allowing 5 processes in the final code.

2.5.2 Planning with Excel

Showcased in **Figure 1** is an Excel spreadsheet that I created for myself whilst I was trying to write some pseudocode, in order to better visualise the algorithm that I was envisioning before coding it.

- Dark Blue - Initial Random Data.
- Yellow - Manual Input Override (Can be left blank).
- Brown - Data Used (Random data unless manual inputs are pre-set).
- Light Blue - Cells Always Equal to 0.
- Green - Calculation in Progress.
- Orange - Calculation Completed.

Inputs	Initial Random Data				Manual Override Input				Final Input Data			
ID	Arrival	Burst		Priority	Arrival	Burst		Priority	Arrival	Burst		Priority
		Predicted	*Actual*			*Predicted*	*Actual*			*Predicted*	*Actual*	
P1	0	14	11	3	0				0	14	11	3
P2	19	10	12	2					19	10	12	2
P3	30	33	17	1					30	33	17	1
P4	34	37	48	2					34	37	48	2
P5	51	28	40	3					51	28	40	3

Algorithm Efficiency						
Efficiency	Completion Time	CPU Utilization	Turnaround Time	Waiting Time	Response Time	Throughput
Aim	Minimum Time Spent	Maximum Efficiency	Minimum Time Spent	Minimum Time Spent	Minimum Time Spent	Maximum Amount
Desc.	The time that a particular process is completed.	Total CPU Time - Time Switching Processes / Total CPU Time	The completion time subtracted by the arrival time.	The amount of time a process is waiting in order to be processed.	The time it takes from a process arriving until the first response.	The number of processes completed per time unit.
P1	11	This cannot accurately be tested here as the time spent switching processes varies based on the CPU speed.	11	0	0	27.2
P2	31		12	0	19	
P3	48		18	1	31	
P4	136		102	54	48	
P5	91		40	0	96	
Max	136	Aim:	102	54	96	
Total		100.00%		55	194	

***2.5 - Figure 1**: An example of the algorithm using random inputs, breaking it down into smaller steps.*

Algorithm Simulation (Step by Step Process of the Java Program)																	
Time	Length	ID	Arrival	Burst Time		Priority	Points	Order	Actual Burst			Time Left	Predicted Burst		Age	Progress	Time Done
				Predicted	*Actual*				*Total*	*Done Pre.*	*Done*		*Total*	*Left*			
								Process 1 Arrives									
2	2	P1	0	14	11	3	0	P1	11	0	2	0	11	11	2	9%	N/A
								Process 2 Arrives									
17	15	P1	0	14	11	3	1	P1	11	2	17	0	11	6	17	74%	N/A
		P2	2	10	12	2	0	P2	12	0	0	0	10	10	15	0%	N/A
								Process 3 Arrives									
38	9	P1	0	14	11	3	2	P1	11	17	13	3	11	0	0	100%	23
		P2	2	10	12	2	1	P3	18	0	1	0	10	27	33	17%	N/A
		P3	17	33	58	1	0	P3	58	0	0	0	33	33	3	0%	N/A
								Process 4 Arrives									
43	17	P1	0	14	11	3	1	P4	16	0	16	1	11	3	0	100%	42
		P2	2	10	12	2	1	P1	11	11	11	1	11	0	0	100%	23
		P3	17	33	58	1	0	P2	12	1	4	0	10	26	44	22%	N/A
		P4	26	11	16	2	2	P3	58	0	0	0	33	33	80	0%	N/A
								Process 5 Arrives									
134	91	P1	0	14	11	3	3	P1	16	0	16	33	33	-15	0	100%	78
		P2	2	10	12	2	2	P4	16	16	16	33	11	5	0	100%	42
		P3	17	33	58	1	10	P1	23	23	23	33	23	0	0	100%	23
		P4	26	11	16	2	4	P2	18	4	18	19	10	12	0	100%	115
		P5	43	30	19	3	1	P5	19	0	19	0	30	11	0	100%	134

***2.5 - Figure 2**: The resulting step-by-step process of the scheduler, using all the data from **Figure 1**.*

A final breakdown of **Figure 2** can be seen below, which allowed me to finish coding the project.

Description of Algorithm Variables		
Time		The time *(in ms)* that the next process arrives. This is when the algorithm determines if the process should continue based on the factors below.
Length		This is how long *(in ms)* until the next process arrives *(or if all processes have arrived, until they have all been completed)*.
Priority		An initial priority level of 1 is high, 2 is medium and 3 is low.
Points		The higher the priority, the more points allocated. Processes that have been around longer than the average burst time are given more points
Order		The process with the highest number of points is calculated to be the most efficient process to work on next.
Actual Burst	**Total**	The time it will take to complete a process *(in ms, this number is only used if it is known, otherwise it is compared to the average burst time).*
	Done Pre.	The amount of time *(in ms)* spent on the process so far.
	Done	The amount of time *(in ms)* spent on the process at the time the next process arrives.
Time Left		The time left to complete a process *(in ms, not used in any calculations as it would be unknown, this is simply to represent the algorithm above)*.
Predicted Burst	**Total**	The amount of time *(in ms)* that the process is predicted to take.
	Left	The amount of time *(in ms)* left before the process is completed.
Age		This is how much time *(in ms)* the process has spent waiting to be completed. Finished processes have their age set back to 0.
Progress		The current status of each process when the next one arrives *(this would also be unknown and is just used to showcase the algorithm efficiency)*.
Time Done		The time *(in ms)* that the current process finished. If it is still ongoing, a value of "N/A" is displayed instead.

3 Python Code

The following Python tasks were created to showcase various features that have not yet been highlighted in previous Java tasks, including list amending and data writing.

3.1 Basic Mental Arithmetic Test

3.1.1 Calculating Random Questions

```
import random # Imports the module "random", so the program can randomly generate numbers.
total = 0
name = input("Please input your full name. ")
print("Hello", name + ", you must now answer the 10 basic questions.\n")

for n in range(10):

    number1 = random.randrange(1, 51) # The variable "number1" is set to a random number
between 1 and 50.
    number2 = random.randrange(1, 51) # The variable "number2" is set to a random number
between 1 and 50.
    operation = random.randrange(1, 4) # The variable "operation" is set to a random number
between 1 and 3.

    if operation == 1: # If "operation" is 1, the program asks the user to attract the two
numbers.

        guess = int(input("Question " + str(n + 1) + ": What is " + str(number1) + " add "
+ str(number2) + "? "))
        answer = number1 + number2

    elif operation == 2: # If "operation" is 2, the program asks the user to subtract the
two numbers.

        guess = int(input("Question " + str(n + 1) + ": What is " + str(number1) + "
subtract " + str(number2) + "? "))
        answer = number1 - number2

    elif operation == 3: # If "operation" is 3, the program asks the user to multiply two
numbers together.

        guess = int(input("Question " + str(n + 1) + ": What is " + str(int(number1 / 5)) +
" times " + str(int(number2 / 5)) + "? "))
        answer = int(number1 / 5) * int(number2 / 5) # Divided by 5 to make the mental
calculation easier.
        # The variables "number1" and "number2" are converted from possible floats to
integers.

    if guess == answer:

        print("Correct!")
        total += 1 # Adds 1 to the variable "total".

    else:

        print("Incorrect! The correct answer is", str(answer) + ".")

print("\nYou scored", total, "out of 10.")
```

3.1.2 Writing User Input Data to a File

```
import random # Imports the module "random", so the program can randomly generate numbers.

class1 = [] # This creates an empty list called "class1".
class2 = [] # This creates another empty list, called "class2".
class3 = [] # This creates one more empty list, called "class3".

mainLoop = True # Allows the program later on to loop infinitly until a student or teacher
terminates the program.

def doesClassExist(): # Defines a function called "doesClassExist", which is used to
determine if the class the user inputs is a registered class or not.

    global classNumber # Declares "classNumber" as a global variable, allowing it to be
accessed by any part of the program.
    global theClass # Declares "theClass" as a global variable, allowing it to be accessed
by any part of the program.
    classCheckLoop = False # Sets "classCheck" to False, to later check to see if the class
the user inputted is valid.
    while classCheckLoop == False: # While "classCheck" is set to False, this will continue
looping. This is to make sure the user inputs a class that is valid.

        try: # The program will try and run this section of code, however if the user
inputs something that is not a number, the program will stop doing this section of code,
rather than simply breaking.

            classNumber = int(input("What class are you in? (1, 2 or 3?) "))

            if classNumber < 1 or classNumber > 3:
                print("Class number not recognised.")

            else:
                classCheckLoop = True # This sets the variable "classCheck" to True,
stopping this loop from still looping.

                # Sets the following variable "theClass" to be equal to the user's class.

                if classNumber == 1:
                    theClass = class1
                elif classNumber == 2:
                    theClass = class2
                elif classNumber == 3:
                    theClass = class3

        except ValueError: # If the user has inputted something that is not a number, the
program will print the message below.

            print("I'm sorry, that was not a valid response.")

    return(classNumber) # Whenever this function is called, it returns the variable
"classNumber".

# Start of the main program

print("""Welcome to the Python 10 Questions Program, where you will be asked in total 10
basic arithmetic questions, containing addition, subtraction or multiplication.
The results will then be stored along with all the other students in your class.\n""")
```

```
while mainLoop == True: # This will loop the program forever, so long as mainLoop is equal
to True.

    total = 0
    restartLoop = True
    name = input("Please input your full name. ")
    classNumber = doesClassExist() # The function "doesClassExist" at the top of the
program will run, and returns the users class number, which is stored as the variable
"classNumber".
    print("\nHello", name + ", you must now answer the 10 basic arithmetic questions.\n")

    for question in range(10):

        number1 = random.randrange(1, 51) # The variable "number1" is set to a random
number between 1 and 50.
        number2 = random.randrange(1, 51) # The variable "number2" is set to a random
number between 1 and 50.
        operation = random.randrange(1, 4) # The variable "operation" is set to a random
number between 1 and 3.

        if operation == 1: # If "operation" is 1, the program asks the user to attract the
two numbers.
            guess = int(input("Question " + str(n + 1) + ": What is " + str(number1) + "
add " + str(number2) + "? "))
            answer = number1 + number2

        elif operation == 2: # If "operation" is 2, the program asks the user to subtract
the two numbers.
            guess = int(input("Question " + str(n + 1) + ": What is " + str(number1) + "
subtract " + str(number2) + "? "))
            answer = number1 - number2

        elif operation == 3: # If "operation" is 3, the program asks the user to multiply
the two numbers together.

            guess = int(input("Question " + str(n + 1) + ": What is " + str(int(number1 /
5)) + " times " + str(int(number2 / 5)) + "? "))
            answer = int(number1 / 5) * int(number2 / 5) # Divided by 5 to make the mental
calculation easier.
            # The variables "number1" and "number2" are converted from possible floats to
integers.

        try: # The program will try to convert the variable "guess" to a float, so that if
the user accidently inputs anything that is not a number, then the program will not break.

            if guess == answer:

                print("Correct!")
                total += 1 # Adds 1 to the variable "total".

            else:
                print("Incorrect! The correct answer is", str(answer) + ".")

        except ValueError: # If the user has inputted something that is not a number, then
the program will stop trying to convert the variable "guess" to a float and will print out
the message below.
            print("I'm sorry, that is not a valid number. The correct answer is",
str(answer) + ".")

    print("\nYou scored", total, "out of 10.\n") # This tells the user how many questions
they got right out of 10.
```

```
    if classNumber == 1:
        class1.append((name, total)) # Adds the variables "name" and "total" to the end of
list "class1".
    elif classNumber == 2:
        class2.append((name, total)) # Adds the variables "name" and "total" to the end of
list "class2".
    elif classNumber == 3:
        class3.append((name, total)) # Adds the variables "name" and "total" to the end of
list "class3".

    resultsAll = "\n" # Sets the variable "resultsAll" to a new line.

    for n in range(len(theClass)): # Loops 1 time for each student in the list "theClass".

        results = "\nName: " + str(theClass[n][0]) + "\nScore: " + str(theClass[n][1]) +
"/10\n"
        resultsAll += results

    f = open("C:/Users/Custom/Desktop/Class " + str(classNumber) + " Data.txt", "w") # This
opens a file and refers to it as "f" whenever it is used in the program.
    f.write("These are all the results for each student in class " + str(classNumber) + "
who have taken the Python 10 Questions Program." + """\nA total of """ + str(len(theClass))
+ " students in class " + str(classNumber) + " have taken the quiz so far." +
str(resultsAll)) # This writes all the data to a text file with the class name as the
title, as well as a short explanation of what the data is. The file also contains one of
three ways to sort the data, either by alphabetical order, highest score or highest average
score.
    f.close() # This closes the file "f" that the program has just written to.

    print("Your results have been saved in the text file \"Class " + str(classNumber) + "
Data\".")

    while restartLoop == True:

        restart = input("Do you want to restart the program? ")

        if restart.lower() == "yes" or restart.lower() == "y":

            print("The program will now restart.\n")
            restartLoop = False

        elif restart.lower() == "no" or restart.lower() == "n":

            print("The program will now end.")
            mainLoop = restartLoop = False

        else:
            print("I'm sorry, that was not a valid response.\n")
```

3.1.3 Extending Program Functionality

```
import random # Imports "random", so the program can randomly generate numbers.
from operator import itemgetter # Imports "itemgetter" from the module "operator", allowing
the program to sort tuples by any of the data inside, rather than just the first one.

class1 = [] # This creates an empty list called "class1".
class2 = [] # This creates another empty list, called "class2".
class3 = [] # This creates one more empty list, called "class3".

mainLoop = True
teacherMode = False
requestType = 1

def username(): # A function used several times to detemine different users name, as
multiple students will use the program.

    global name # Declares "name" as a global variable, allowing it to be accessed by any
part of the program.
    global teacherMode # Declares "teacherMode" as a global variable, allowing it to be
accessed by any part of the program.
    name = input("Please input your full name. ")
    return(name) # Whenever this function is called, it returns the variable "name".

def doesClassExist(): # A function used to determine if the class the user inputs is a
registered class or not.

    global classNumber # Declares "classNumber" a global variable, allowing it to be
accessed by any part of the program.
    classCheck = False # Sets "classCheck" to False, making it a boolean, and checks to see
if the class the user inputted is valid.
    while classCheck == False: # While "classCheck" is set to False, this will continue
looping. This is to make sure the user inputs a class that is valid.

        try: # The program will try and run this section of code, however if the user
inputs something that is not a number, the program will stop doing this section of code,
rather than simply breaking.

            classNumber = int(input("What class are you in? (1, 2 or 3?) "))

            if classNumber < 1 or classNumber > 3:

                print("Class number not recognised.")

            else:

                classCheck = True

        except ValueError: # If the user inputs something that is not a number, the program
will print out the message below.

            print("I'm sorry, that was not a valid response.")

    return(classNumber) # Whenever this function is called, it returns the variable
"classNumber".

def request(whichScore, reason, sortBy, isReverse, theClass, classNumber): # A function
used to sort out all the data into a certain order, depending on what the user chooses.
```

```
    classNames = [x[0] for x in theClass] # Creates a list of all the students in whatever
class the user selected.
    classScores = [x[whichScore] for x in theClass] # Creates a list of all the students
high scores or average scores, depending on how the user wants to sort all the data in that
class.
    classSingleNames = []
    classSingleScores = []
    classHighestScores = []

    for n in range(len(classNames)): # Loops 1 time for each student in the list
"theClass".

        classSingleNames.append(classNames[n]) # Appends the students to the list
"classSingleNames" created above.
        classSingleScores.append(classScores[n]) # Appends the student's score to the list
"classSingleScores" created above.
        classHighestScores.append((classSingleNames[n], classSingleScores[n])) # Appends a
student and their score together.

    results = sorted(classHighestScores, key = itemgetter(sortBy), reverse = isReverse) #
Stores how the data was stored.

    if teacherMode == True:

        print("\nThese are the results for each student in class", str(classNumber) + ".")

    f = open("C:/Users/Custom/Desktop/Class " + str(classNumber) + " Data.txt", "w") #
Opens a file and refers to it as "f" whenever it is used in the program. This particular
link would change depending on the way future python programs would be used.
    f.write("These are all the results for each student in class " + str(classNumber) + """
who have taken the Python 10 Questions Program.\n
Below is each students highest score, average score and their last three attempts at the
questions.\n\n""" + str(theClass) + """\n
The data can also be sorted by the teacher in one of three ways, which is done inside the
program.
It can be sorted either alphabetically, by highest score or by highest average
score.\n\n""" + str(reason) + """
A total of """ + str(len(classNames)) + " students in class " + str(classNumber) + " have
taken the quiz so far.\n\n" + str(results)) # Writes all the data to a text file with the
class name as the title, as well as a short explanation of what the data is. The file also
contains one of three ways to sort the data, either by alphabetical order, highest score or
highest average score.
    f.close() # This closes the file "f" that the program has just written to.

def dataSave(): # A function used to decide how the teacher wants to sort the students
data.

    global requestType # Declares "requestType" as a global variable, allowing it to be
accessed by any part of the program.
    global teacherMode # Declares "teacherMode" as a global variable, allowing it to be
accessed by any part of the program.

    if classNumber == 1:
        theClass = class1
    elif classNumber == 2:
        theClass = class2
    elif classNumber == 3:
        theClass = class3

    if requestType == 1 and teacherMode == True: # The following code will only happen if
both conditions are met.
```

```
        print("\nThese are the results for each student in class", str(classNumber) + ",
including the students names, highest score, average score and their last three
attempts.\n")

        if classNumber == 1:
            print(class1, "\n")
        elif classNumber == 2 :
            print(class2, "\n")
        elif classNumber == 3:
            print(class3, "\n")

    elif requestType == 1 or requestType == 2: # The following code will only happen if one
of the conditions are met.
        request(1, "This is all the users sorted in alphabetical order, as well as their
highest score.", 0, False, theClass, classNumber) # All this data is sent up to the
function "request".

    elif requestType == 3: # If the variable "requestType" is equal to 3, then 6 different
parameters are sent up to the function "request", which then determine what is printed and
in what order.
        request(1, "This is all the users sorted by their highest score.", 1, True,
theClass, classNumber) # All this data is sent up to the function "request".

    elif requestType == 4: # If the variable "requestType" is equal to 4, then 6 other
parameters are sent up to the function "request", which then determine what is printed and
in what order.
        request(2, "This is all the users sorted by their highest average score.", 1, True,
theClass, classNumber) # All this data is sent up to the function "request".

    elif requestType == 5 and teacherMode == True: # If the variable "requestType" is equal
to 5, then 6 more parameters are sent up to the function "request", which then determine
what is printed and in what order.
        print("Leaving teacher mode...\n")
        teacherMode = False

    elif teacherMode == False:
        pass # Simply tells the program to do nothing, and to move on to the next line of
code.

    else:
        print("I'm sorry, that is not a valid response.\n")

    return(requestType) # Whenever this function is called, it returns the variable
"requestType".

print("""Welcome to the Python 10 Questions Program, where you will be asked in total 10
basic maths questions, with either addition, subtraction or multiplication.
The results will then be stored along with all the other students in your class.\n""")

while mainLoop == True: # This will loop the program forever, so long as mainLoop is equal
to True.

    nameLoop = restartLoop = True
    teacherMode = False
    threeTries = []
    bestTotal = averageTotal = 0

    while nameLoop == True:

        name = username()
```

```
        if name.lower() == "teacher": # If the user inputs "teacher", a list of options
instead of questions will appear.
            teacherMode = True
            print("Hello teacher, what would you like to do with the students results?\n")
        else:
            nameLoop = False

        while teacherMode == True:

            requestLoop = True

            while requestLoop == True:

                try: # The program will try and run this section of code, however if the
user inputs something that is not a number, the program will stop doing this section of
code, rather than simply breaking.

                    requestType = int(input("""1) View all of the students results,
including their lastest three tries.
2) Sort the students into alphabetical order with their highest score.
3) Sort the students results from the highest to lowest score.
4) Sort the students results from the highest to lowest average score.
5) Return back to student mode.\n""")) # The variable "requestType" is set to whatever
number the user inputs. The triple brackets are used so that the string can be written on
more than one line, makeing it easier to read when coding long strings.
                    requestLoop = False

                except ValueError:
                    print("I'm sorry, that was not a valid response.\n")

            if requestType >= 1 and requestType <= 4:
                classNumber = doesClassExist()

            requestType = dataSave()

            if requestType <= 1 and requestType >= 5:
                print("What would you like to do with the students results?\n")

    classNumber = doesClassExist()

    print("\nHello", name + ", you must now answer the 10 basic questions.\n")

    for setNumber in range(1, 4): # This loops through the 10 random questions and the
users result 3 times.

        print("Set", setNumber, "of questions.\n")
        total = 0

        for question in range(10):

            number1 = random.randrange(1, 51) # The variable "number1" is set to a random
number between 1 and 50.
            number2 = random.randrange(1, 51) # The variable "number2" is set to a random
number between 1 and 50.
            operation = random.randrange(1, 4) # # The variable "operation" is set to a
random number between 1 and 3.

            if operation == 1: # If "operation" is 1, the program asks the user to attract
the two numbers.
```

```
                guess = int(input("Question " + str(n + 1) + ": What is " + str(number1) +
" add " + str(number2) + "? "))
                answer = number1 + number2

            elif operation == 2: # If "operation" is 2, the program asks the user to
subtract the two numbers.

                guess = int(input("Question " + str(n + 1) + ": What is " + str(number1) +
" subtract " + str(number2) + "? "))
                answer = number1 - number2

            elif operation == 3: # If "operation" is 3, the program asks the user to
multiply two numbers together.

                guess = int(input("Question " + str(n + 1) + ": What is " + str(int(number1
/ 5)) + " times " + str(int(number2 / 5)) + "? "))
                answer = int(number1 / 5) * int(number2 / 5) # Divided by 5 to make the
mental calculation easier.
                # The variables "number1" and "number2" are converted from possible floats
to integers.

            try: # The program will try to convert the variable "guess" to a float, so that
if the user accidently inputs anything that is not a number, then the program will not
break.

                if guess == answer:

                    print("Correct!")
                    total += 1 # Adds 1 to the variable "total".

                else:
                    print("Incorrect! The correct answer is", str(answer) + ".")

            except ValueError:

                print("I'm sorry, that is not a valid number. The correct answer is",
str(answer) + ".")

        print("\nYou scored", total, "out of", str(question + 1) + ".\n")

        threeTries.append(total) # Adds the users total to the end of list "threeTries".

        if total > bestTotal:
            bestTotal = total

        averageTotal += total

    if averageTotal / setNumber == averageTotal // setNumber: # If the variable
"averageTotal" divided by the number of sets of questions is an interger, then it is set to
that number.

        averageTotal = int(averageTotal / setNumber) # While this is not needed in the
program for it to work, it stores the average as an integer so that it looks much better
when printed into the Python Shell. This is because it will say, for example, 7 instead of
7.0.

    else: # If the above if statement is not run, then the next line of code will run
instead.
```

```
        averageTotal = round(averageTotal / setNumber, 2) # This rounds the averageTotal to
2 decimal places, so that when printed out into the Python Shell, for example, it prints
5.33 instead of 5.33333333, which is unnecessarily long.

    if classNumber == 1: # If "classNumber" is equal to 1, it will append the users name,
best score, average score and the three tries to the list "class1".
        class1.append((name, bestTotal, averageTotal, threeTries))

    elif classNumber == 2: # If "classNumber" is equal to 2, it will do the same as the
above, by appending the users name, best score, average score and the three tries, but
instead to the list "class2".
        class2.append((name, bestTotal, averageTotal, threeTries))

    elif classNumber == 3: # If "classNumber" is equal to 3, it will also instead append
the users name, best score, average score and the three tries to the list "class3".
        class3.append((name, bestTotal, averageTotal, threeTries))

    requestType = dataSave()

    print("If you would like to view all the classes results, simply input \"teacher\" when
you are asked to input your name.\n")

    while restartLoop == True:

        restart = input("Do you want to restart the program? ")

        if restart.lower() == "yes" or restart.lower() == "y":
            print("The program will now restart.\n")
            restartLoop = False

        elif restart.lower() == "no" or restart.lower() == "n":
            print("The program will now end.")
            mainLoop = restartLoop = False

        else:
            print("I'm sorry, that was not a valid response.\n")
```

3.2 Determining Password Strength

```
mainLoop = True

while mainLoop == True:

    passwordNumberValid = False
    passwordList = []
    passwordStrengthList = []
    passwords = ""

    strongest = weakest = strongestNumber = weakestNumber = reset = None

    while passwordNumberValid == False:

        try:

            passwordNumber = int(input("How many passwords would you like to test? "))

        except ValueError:

            passwordNumber = None
            print("You did not enter a valid number.\n")

        if passwordNumber != None:

            if passwordNumber == 1:
                print("You have chosen to test only 1 password.")
                passwordNumberValid = True

            elif passwordNumber <= 0:
                print("You must enter a number higher than 0. Please enter another
number.")

            elif passwordNumber > 100:
                print("You may only test up to 100 passwords. Please enter another
number.")

            else:
                print("You have chosen to test", passwordNumber, "passwords.")
                passwordNumberValid = True

    for i in range(passwordNumber):

        passwordValid = False
        lowerCase = upperCase = numbers = space = symbols = False
        notIncluded = []
        reason = ""
        improve = ""

        while passwordValid == False:

            password = input("\nInput a password between 6 and 12 characters long: ")

            if len(password) < 6:
                print("Your password is too short, as it is under 6 characters long.")

            elif len(password) > 32:
                print("Your password is too long, as it is over 32 characters long.")
```

```
            else:
                passwordValid = True
                print("Password accepted.")

        for letter in password:

            if letter >= "a" and letter <= "z":
                lowerCase = True

            elif letter >= "A" and letter <= "Z":
                upperCase = True

            elif letter >= "0" and letter <= "9":
                numbers = True

            elif letter == " ":
                space = True

            elif letter != " ":
                symbols = True

        charTypes = [lowerCase, upperCase, numbers, space, symbols]
        charTypesText = ["lower case letters", "upper case letters", "numbers", "a space",
"symbols"]

        for n in range(5):

            if charTypes[n] == True:
                included.append(charTypesText[n])

            else:
                notIncluded.append(charTypesText[n])

        for n in range(len(included)):

            if n == 0: # If this is the first loop, it only adds the first string in the
list "included" to the string "reason".
                reason = reason + included[n]
            elif n >= 1 and len(included) - n != 1: # If this is not the first and last
loop, a comma is added to the string "reason", followed by the next string in the list
"included".
                reason = reason + ", " + included[n]
            elif n >= 1 and len(included) - n == 1: # If this is not the first and is the
last loop, the string " and ", is added to the string "reason", as well as the last string
in the list "included".
                reason = reason + " and " + included[n]

        if len(notIncluded) != 0: # This onlys happens if there is anything the user can do
to improve their password.

            for n in range(len(notIncluded)):

                if n == 0: # If this is the first loop, it only adds the first string in
the list "notIncluded" to the string "improve".
                    improve = improve + notIncluded[n]
                elif n >= 1 and len(notIncluded) - n != 1: # If this is not the first and
last loop, a comma is added to the string "improve", followed by the next string in the
list "notIncluded".
                    improve = improve + ", " + notIncluded[n]
```

```
                elif n >= 1 and len(notIncluded) - n == 1: # If this is not the first and
is the last loop, the string " and ", is added to the string "improve", as well as the last
string in the list "notIncluded".
                    improve = improve + " and " + notIncluded[n]
        else:

            improve = "nothing else" # This is used later to tell the user what they could
do to improve their password.

        if len(included) == 1: # If there is only 1 item in the list "included", the
variable "strength" is set to "weak".
            strength = "weak"
        elif len(included) == 2: # If there are 2 items in the list "included", the
variable "strength" is set to "medium".
            strength = "medium"
        elif len(included) == 3: # If there are 3 items in the list "included", the
variable "strength" is set to "strong".
            strength = "strong"
        elif len(included) == 4: # If there are 4 items in the list "included", the
variable "strength" is set to "very strong".
            strength = "very strong"
        else: # This sets the variable "strength" to "extremely strong".
            strength = "extremely strong"

        print("Your password's strength is", strength + ", because you included", reason +
". \nYou could have included", improve, "to improve the strength of your password.")
        passwordList.append(password)
        passwordStrengthList.append(len(included))

        if strongestNumber == None or passwordStrengthList[i] > strongestNumber: # If the
user has only inputted one password, or if the password the user just inputted was stronger
than the previously strongest password, the next block of code will continue.

            strongest = passwordList[i]
            strongestNumber = passwordStrengthList[i]

        elif passwordStrengthList[i] == strongestNumber and len(password) >
len(passwordList[strongestNumber]): # If the current strongest password and the latest
inputted password have the same strength, the longest of the two passwords is set to the
strongest.

            strongest = passwordList[i]
            strongestNumber = passwordStrengthList[i]

        if weakestNumber == None or passwordStrengthList[i] < weakestNumber: # If the user
has only inputted one password, or if the password the user just inputted was weaker than
the previously weakest password, the next block of code will continue.

            weakest = passwordList[i]
            weakestNumber = passwordStrengthList[i]

        elif passwordStrengthList[i] == weakestNumber and len(password) <
len(passwordList[weakestNumber]): # If the current weakest password and the latest inputted
password have the same strength, the shorter of the two passwords is set to the weakest.

            weakest = passwordList[i]
            weakestNumber = passwordStrengthList[i]

        # This is the end of the main password strength loop.

    for n in range(passwordNumber):
```

```
        if n == 0:
            passwords = passwords + passwordList[n]

        elif n >= 1 and passwordNumber - n != 1:
            passwords = passwords + ", " + passwordList[n]

        elif n >= 1 and passwordNumber - n == 1:
            passwords = passwords + " and " + passwordList[n]

    if passwordNumber > 1: # If the user chooses to test more than 1 password, it will tell the user all the passwords they tried, as well as which one was the strongest and which one was the weakest.

        print("\nThe", str(passwordNumber), "passwords you inputted were", str(passwords) + ".")
        print("The strongest password was", str(strongest) + ", and the weakest password was", str(weakest) + ".")

    while reset == None: # This will keep looping until the user inputs a valid response.

        reset = input("\nDo you want to test out more passwords? ")

        if reset.lower() == "yes" or reset.lower() == "y":
            print("The program will now reset itself.")

        elif reset.lower() == "no" or reset.lower() == "n":
            print("The program will now stop.")
            mainLoop = False

        else:
            print("That was not a valid response. You must enter either \"Yes\" or \"No\".")
            reset = None
```

3.3 Classification Algorithm

3.3.1 Outlining the Code

The program in this task takes data from a provided Excel spreadsheet called “dataset” containing thousands of rows worth of data, in order to attempt to correctly filter out spam and useful data. The algorithm below was created to attempt to do this fast but with as high efficiency as possible.

```
# Importing libraries

import pandas as pand
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split, cross_val_score

# ------------------------------------------------------------------------------------- #

# Turning the CSV dataset into useable variables

data = pand.read_csv('dataset.csv', sep = ',')
inputs = data.iloc[:, :57]
outputs = data.iloc[:, 57]
```

When I began developing the algorithm for this task, I first imported the library known as “pandas” in order to read in the CSV file provided, which I stored into a variable called ‘data’. I then used a variable called ‘inputs’ to store the first 57 columns, which is what is later used to predict if an email is spam. I then made a second variable called ‘outputs’ to store the data from the last column, which is what will later determine how accurate my algorithm’s predictions were

```
# Spliting the data to be used for learning and testing (with a split of 80/20)

inputsTrain, inputsTest, outputsTrain, outputsTest = train_test_split(inputs, outputs,
    test_size = 0.2, random_state = 12)
inputsTrain = StandardScaler().fit_transform(inputsTrain)
inputsTest = StandardScaler().fit_transform(inputsTest)
```

Once I had imported all the data from the dataset, I then started to split my data between training and testing. I decided upon using an 80/20 split, as this appeared to be the industry standard upon my own further research. I also made sure to scale the input values so that they were all reasonably weighted, as I did not want to risk values with a greater magnitude outweighing any lesser values.

```
# Network 1 is a Rectified Linear Unit (relu) Neural Network

network1 = MLPClassifier(hidden_layer_sizes = (57, 10, 5), activation = 'relu',
    solver = 'adam', max_iter = 1000)

# Network 2 is a tanh(x) Neural Network

network2 = MLPClassifier(hidden_layer_sizes = (57, 10, 5), activation = 'tanu',
    solver = 'adam', max_iter = 1000)

# Network 3 is a Logistic Neural Network

network3 = MLPClassifier(hidden_layer_sizes = (57, 10, 5), activation = 'logistic',
    solver = 'adam', max_iter = 1000)
```

After scaling the inputs for my algorithm, I began creating three separate neural networks. I decided to use the same solver for each network, but with different activations. My first neural network uses a ‘rectified linear unit (*RELU*)’ activation, a min / max function between the value 0 and the provided inputs. My second neural network uses a ‘tanh’ activation, which takes the inputs provided from the dataset and puts them through a tan function. My final neural network uses a ‘logistic’ activation,

using the supplied inputs alongside a coefficient of k for a sigmoid function to produce an output. All three of my networks had three hidden layers (which are 57, 10 and 5). I also wanted to make sure that my algorithm had enough iterations to produce a sufficiently accurate result but didn't spend too much time once it had reached that point, which is why I settled for 1000 max iterations.

```
# Training the networks

network1.fit(inputsTrain, outputsTrain)
network2.fit(inputsTrain, outputsTrain)
network3.fit(inputsTrain, outputsTrain)

# Determining the cross-validation scores

network1Results = cross_val_score(network1, inputsTrain, outputsTrain, cv = 10)
network2Results = cross_val_score(network2, inputsTrain, outputsTrain, cv = 10)
network3Results = cross_val_score(network3, inputsTrain, outputsTrain, cv = 10)

# ------------------------------------------------------------------------------------- #

# Calculating each networks average accuracy results

network1Accuracy = round((network1Results.mean() * 100), 3)
network2Accuracy = round((network2Results.mean() * 100), 3)
network3Accuracy = round((network3Results.mean() * 100), 3)
totalAccuracy = round(((network1Average + network2Average + network3Average) / 3) * 100, 3)
```

Once all my neural networks had been trained, I then started to estimate the accuracy using a cross-fold validation function that used 10 folds for each network. As discussed later on in the report, I decided to use 10-fold, as this gave me a better and more realistic output when compared to using a 5-fold approach. I then calculated and stored the average result of each individual neural network.

```
# Calculating each networks predicted values

network1Predictions = network1.predict(inputsTest)
network2Predictions = network2.predict(inputsTest)
network3Predictions = network3.predict(inputsTest)

# Calculating each networks score

network1Score = round((network1.score(inputsTest, outputsTest) * 100), 3)
network2Score = round((network2.score(inputsTest, outputsTest) * 100), 3)
network3Score = round((network3.score(inputsTest, outputsTest) * 100), 3)
networkAverageScore = round((network1Score + network2Score + network3Score) / 3, 3)
```

After calculating the accuracy of each network, I made each network predict values based on their supplied inputs, enabling me to work out a score for of the individual networks. Similar to before, I also averaged out all three of the neural networks in order to find a value for the average score.

```
# Calculating each networks accuracy difference

network1Difference = round(network1Accuracy - network1Score, 3)
network2Difference = round(network2Accuracy - network2Score, 3)
network3Difference = round(network3Accuracy - network3Score, 3)

# Determining the average accuracy difference

networkAverageDifference = round(totalAccuracy - networkAverageScore, 3)

# ------------------------------------------------------------------------------------- #
```

Now that I had a prediction and a score for each network, I was able to calculate the difference between each network's estimated accuracy and their actual accuracy. I would then be able to output all of these variables later on in order to show the efficiency of my developed algorithm.

3.3.2 Outputting the Results

```
print("Network 1 Estimated Accuracy: ", network1Accuracy)
print("Network 2 Estimated Accuracy: ", network2Accuracy)
print("Network 3 Estimated Accuracy: ", network3Accuracy)
print()
print("Network 1 Prediction Accuracy: ", network1Score)
print("Network 2 Prediction Accuracy: ", network2Score)
print("Network 3 Prediction Accuracy: ", network3Score)
print()
print("Network 1 Accuracy Difference: ", network1Difference)
print("Network 2 Accuracy Difference: ", network2Difference)
print("Network 3 Accuracy Difference: ", network3Difference)
print()
print("All Networks Estimated Accuracy: ", totalNetworkAccuracy)
print("All Networks Prediction Accuracy: ", networkAverageScore)
print("All Networks Accuracy Difference: ", networkAverageDifference)
print()
```

When the algorithm is run, this is how the output looks:

```
Network 1 Estimated Accuracy: 93.843
Network 2 Estimated Accuracy: 93.987
Network 3 Estimated Accuracy: 94.212

Network 1 Prediction Accuracy: 93.941
Network 2 Prediction Accuracy: 94.238
Network 3 Prediction Accuracy: 95.169

Network 1 Accuracy Difference: -0.098
Network 2 Accuracy Difference: -0.251
Network 3 Accuracy Difference: -0.957

All Networks Estimated Accuracy: 94.014
All Networks Prediction Accuracy: 94.449
All Networks Accuracy Difference: -0.435
```

I originally thought about using a 5-fold cross validation approach, as this produced closer estimates to their actual values in my initial tests. Whilst I was developing the algorithm, however, it became apparent that the estimated results were always outputting higher than the actual results. I then tried using a 10-fold approach, which although produced slightly less accurate actual results in comparison, it did produce much closer estimates to the actual results. After comparing the two approaches, I decided that the slight trade off was definitely worth it, as it became clear to me that the 10-fold cross validation was the more realistic and therefore better solution for this algorithm.

After following the weekly tutorials and completing the assignment, I now have a much better understand of the functionality of python libraries. I have learned how to process a dataset, feeding the data into a developed classification algorithm, in order to produce a spam filter with an average accuracy of > 94%. I found understanding how to use neural networks to be a particular struggle at first, but I believe that it was definitely worth learning about as I believe it helped me vastly improve the efficiency of my algorithm. Overall, I am very pleased with all the progress that I have made.

4 C# Code

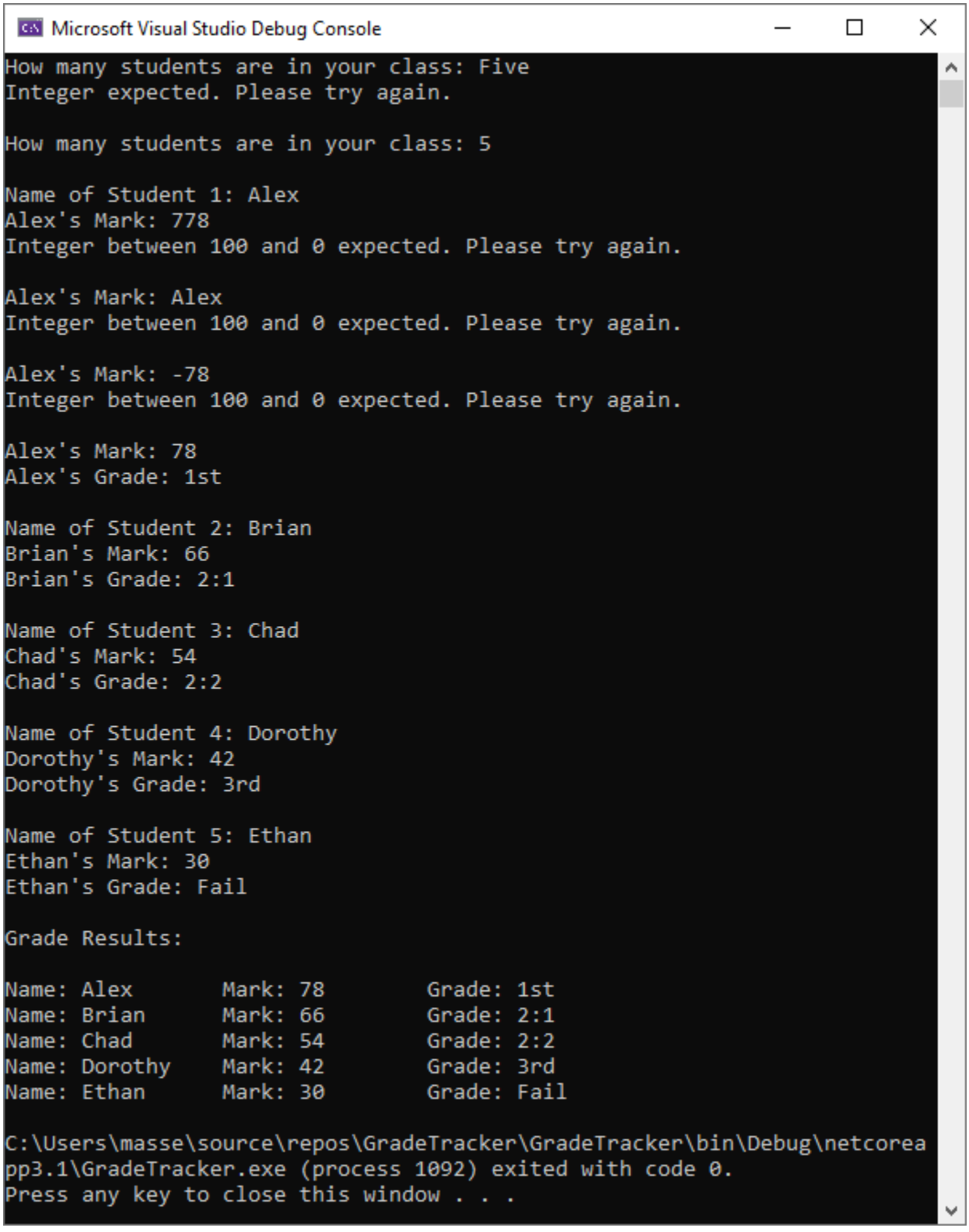
The following C# tasks were created to showcase various features that have not yet been highlighted in previous coding tasks, with the few first created to formalise myself with the C# specific syntax.

4.1 Syntax Familiarisation

4.1.1 Program Class

```
using System;

namespace GradeTracker
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            int studentCount;

            Console.Write("How many students are in your class: ");

            while (!int.TryParse(Console.ReadLine(), out studentCount))
            {
                Console.WriteLine("Integer expected. Please try again.\n");
                Console.Write("How many students are in your class: ");
            }

            var studentNames = new string[studentCount];
            var studentMarks = new int[studentCount];
            var studentGrades = new string[studentCount];

            for (int i = 0; i < studentCount; i++)
            {
                Console.Write("\nName of Student " + (i + 1) + ": ");
                studentNames[i] = Console.ReadLine();

                Console.Write(studentNames[i] + "'s Mark: ");

                while (!int.TryParse(Console.ReadLine(), out studentMarks[i])
                    || studentMarks[i] > 100 || studentMarks[i] < 0)
                {
                    Console.WriteLine("Integer between 100 and 0 expected. Please try again.\n");
                    Console.Write(studentNames[i] + "'s Mark: ");
                }

                studentGrades[i] = studentMarks[i] switch
                {
                    int n when n >= 70 => "1st",
                    int n when n >= 60 => "2:1",
                    int n when n >= 50 => "2:2",
                    int n when n >= 40 => "3rd",
                    _ => "Fail",
                };

                Console.WriteLine(studentNames[i] + "'s Grade: " + studentGrades[i]);
            }

            Console.WriteLine("\nGrade Results:\n");

            for (int i = 0; i < studentCount; i++)
            {
                Console.WriteLine("Name: {0}\t Mark: {1}\t Grade: {2}",
                    studentNames[i], studentMarks[i], studentGrades[i]);
            }
        }
    }
}
```

4.1.2 Testing Outputs



This simple program asks the user for the total number of students in their class, followed by each student's name and mark in order to calculate their grade. If the user tries to input an invalid input at any stage, the program will detect the error, disregard their answer, and ask them the question again. Once all the data is successfully inputted, it is then outputted back in a user-friendly layout.

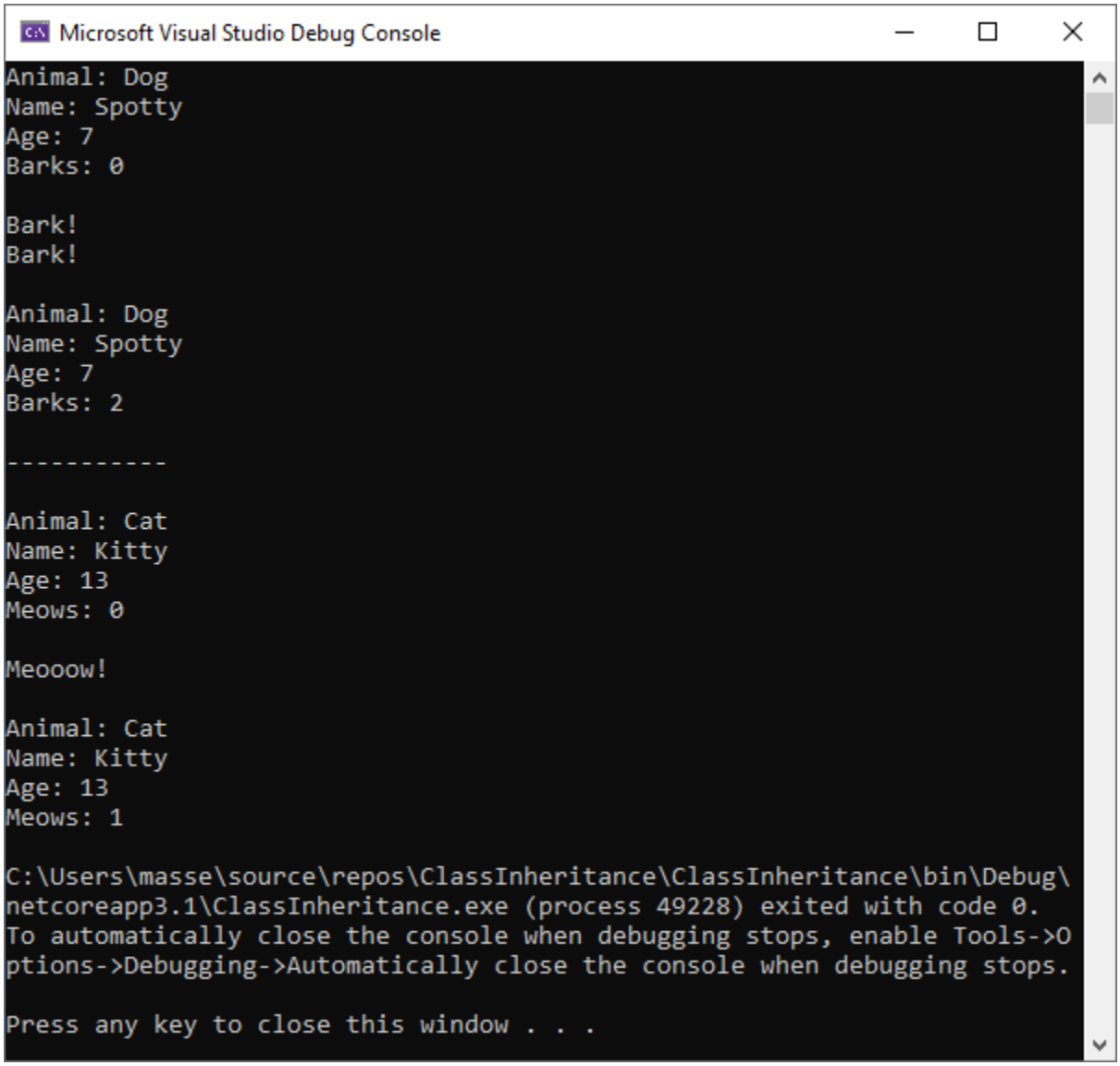
4.2 Class Inheritance

4.2.1 Animal Classes

```
using System;

namespace ClassInheritance
{
    0 references
    class MainClass
    {
        2 references
        class Animal
        {
            public string animal;
            public string name;
            public int age;

            4 references
            public void PrintBaseDetails()
            {
                Console.WriteLine("Animal: " + animal);
                Console.WriteLine("Name: " + name);
                Console.WriteLine("Age: " + age);
            }
        }

        2 references
        class Dog : Animal
        {
            public int barks;
            public int barkNumber;

            2 references
            public void PrintDogDetails()
            {
                Console.WriteLine("Barks: " + barks);
            }

            1 reference
            public void Bark()
            {
                System.Random random = new System.Random();
                barkNumber = random.Next(1, 3);

                Console.WriteLine();

                for (int i = 0; i < barkNumber; i++)
                {
                    Console.WriteLine("Bark!");
                    barks += 1;
                }

                Console.WriteLine();
            }
        }

        2 references
        class Cat : Animal
        {
            public int meows;
            public int meowLength;

            2 references
            public void PrintCatDetails()
            {
                Console.WriteLine("Meows: " + meows);
            }

            1 reference
            public void Meow()
            {
```

```
60                  System.Random random = new System.Random();
61                  meowLength = random.Next(1, 5);
62
63                  Console.Write("\nMe");
64
65                  for (int i = 0; i < meowLength; i++)
66                  {
67                      Console.Write("o");
68                  }
69
70                  Console.WriteLine("w!\n");
71                  meows += 1;
72              }
73          }
74
            0 references
75          public static void Main(string[] args)
76          {
77              Dog spotty = new Dog
78              {
79                  animal = "Dog",
80                  name = "Spotty",
81                  age = 7,
82                  barks = 0
83              };
84
85              Cat kitty = new Cat
86              {
87                  animal = "Cat",
88                  name = "Kitty",
89                  age = 13,
90                  meows = 0
91              };
92
93              spotty.PrintBaseDetails();
94              spotty.PrintDogDetails();
95
96              spotty.Bark();
97
98              spotty.PrintBaseDetails();
99              spotty.PrintDogDetails();
100
101             Console.WriteLine("\n-----------\n");
102
103             kitty.PrintBaseDetails();
104             kitty.PrintCatDetails();
105
106             kitty.Meow();
107
108             kitty.PrintBaseDetails();
109             kitty.PrintCatDetails();
110         }
111     }
112 }
```

4.2.2 Testing Output



This program is an example of utilising inheritance *(as discussed in Section 1.3 of this booklet)* in C#, through the use of a base animal class, followed by cat and dog subclasses. Both animals have their animal type, name, and age, followed by their animal specific features. The dog has an extra *bark()* function, whilst the cat has an extra *meow()* function. The *bark()* function picks a random number between 1 and 3, outputting barks equal to that number, whilst the *meow()* function picks a random number between 1 and 5, which picks how many *o's* should be in the singularly outputted "meow". Once each of these outputted, a new "bark" and "meow" total is outputted to the console.

4.3 Languages Features

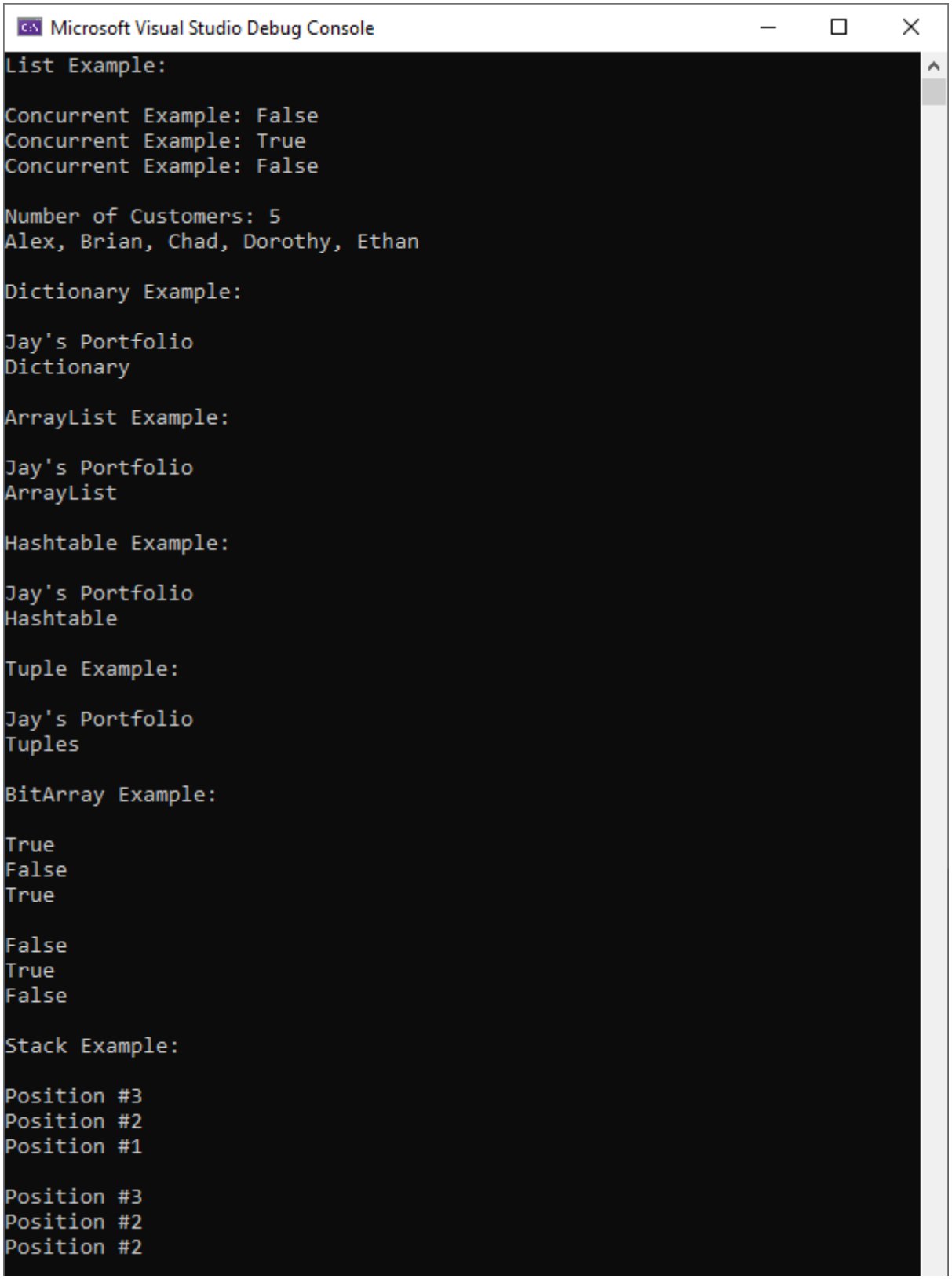
4.3.1 Program Class

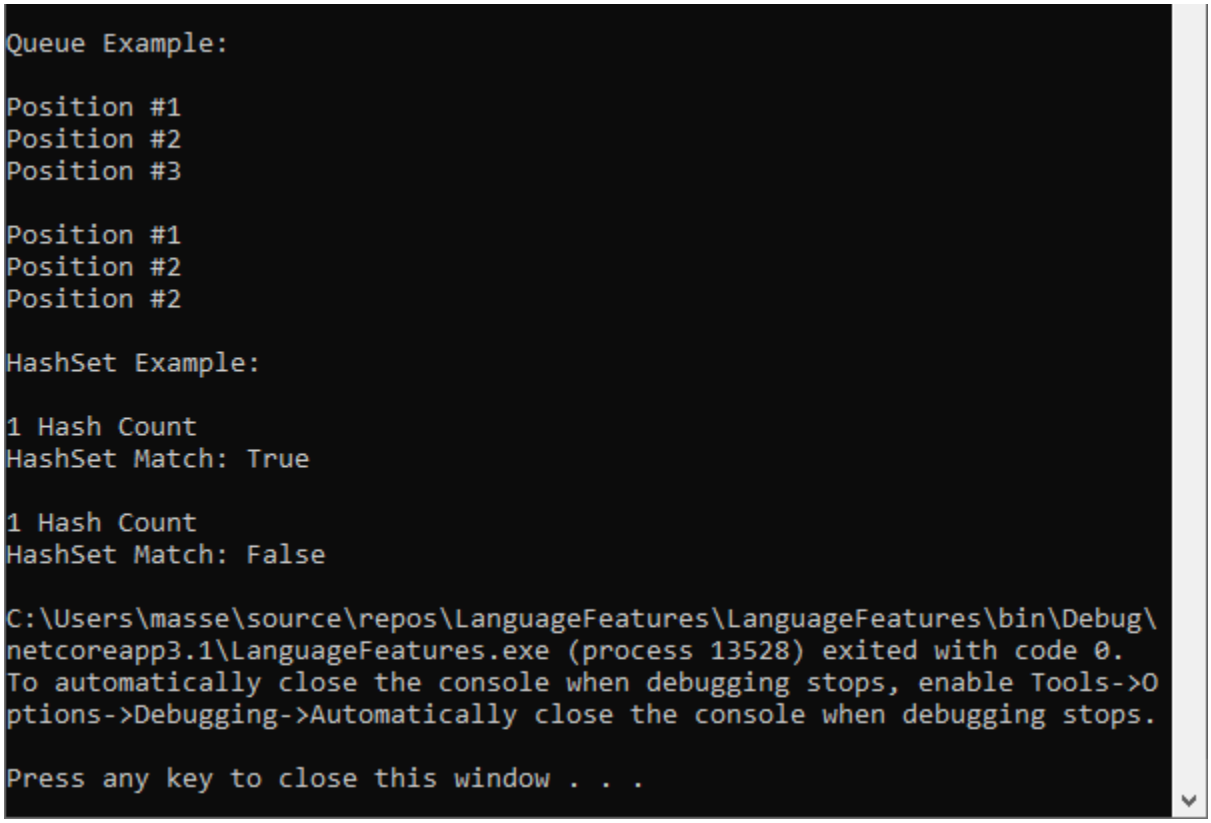
```
using System;
using System.Collections;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Threading;

namespace LanguageFeatures
{
    0 references
    class Program
    {
        // Concurrency

        static readonly ConcurrentDictionary<int, int> items = new ConcurrentDictionary<int, int>();

        0 references
        static void Main(string[] args)
        {
            Thread thread1 = new Thread(new ThreadStart(AddItem));
            Thread thread2 = new Thread(new ThreadStart(AddItem));
            Thread thread3 = new Thread(new ThreadStart(AddItem));

            thread1.Start();
            thread2.Start();
            thread3.Start();

            // Lists

            Console.WriteLine("List Example:\n");

            List<String> customers = new List<string>
            {
                "Alex",
                "Brian",
                "Chad"
            };

            customers.Add("Dorothy");
            customers.Add("Ethan");

            Console.WriteLine("\nNumber of Customers: " + customers.Count);

            for (int i = 0; i < customers.Count - 1; i++)
            {
                Console.Write(customers[i] + ", ");
            }

            Console.Write(customers[^1]); // Prints final item in list "customers".

            // Dictionaries

            Console.WriteLine("\n\nDictionary Example:\n");

            Dictionary<String, String> config = new Dictionary<string, string>
            {
                { "title", "Jay's Portfolio" }
            };

            config.Add("example", "Dictionary");

            Console.WriteLine(config["title"]);
            Console.WriteLine(config["example"]);

            // ArrayLists

            Console.WriteLine("\nArrayList Example:\n");
```

```
66          ArrayList list = new ArrayList
67          {
68              "Jay's Portfolio"
69          };
70          list.Add("ArrayList");
71          String s1a = (String)list[0];
72          String s1b = (String)list[1];
73
74          Console.WriteLine(s1a + "\n" + s1b);
75
76          // Hashtables
77
78          Console.WriteLine("\nHashtable Example:\n");
79
80          Hashtable table = new Hashtable
81          {
82              { "title", "Jay's Portfolio" }
83          };
84
85          table.Add("example", "Hashtable");
86
87          var s2a = (String)table["title"];
88          Console.WriteLine(s2a);
89
90          var s2b = (String)table["example"];
91          Console.WriteLine(s2b);
92
93          // Tuples
94
95          Console.WriteLine("\nTuple Example:\n");
96
97          var tuple1 = Tuple.Create(1, "Jay's Portfolio", true);
98          Tuple<int, String, bool> tuple2 = new Tuple<int, string, bool>(1, "Tuples", true);
99
100         Console.WriteLine(tuple1.Item2);
101         Console.WriteLine(tuple2.Item2);
102
103         // BitArrays
104
105         Console.WriteLine("\nBitArray Example:\n");
106
107         bool[] preload = new bool[3] { true, false, true };
108
109         BitArray booleanValues = new BitArray(preload);
110
111         foreach (var item in booleanValues)
112         {
113             Console.WriteLine(item);
114         }
115
116         booleanValues[0] = false;
117         booleanValues[1] = true;
118         booleanValues.Set(2, false);
119
120         Console.WriteLine();
121
122         foreach (var item in booleanValues)
123         {
124             Console.WriteLine(item);
125         }
126
127         // Stacks
128
129         Console.WriteLine("\nStack Example:\n");
130
131         Stack<String> stackExample = new Stack<string>();
132
133         stackExample.Push("Position #1");
134         stackExample.Push("Position #2");
135         stackExample.Push("Position #3");
136
137         foreach (var pancake in stackExample)
```

```
            {
                Console.WriteLine(pancake);
            }

            Console.WriteLine();
            Console.WriteLine(stackExample.Pop());
            Console.WriteLine(stackExample.Peek());
            Console.WriteLine(stackExample.Peek());

            // Queues

            Console.WriteLine("\nQueue Example:\n");

            Queue<int> queueExample = new Queue<int>();

            queueExample.Enqueue(1);
            queueExample.Enqueue(2);
            queueExample.Enqueue(3);

            foreach (var enqueued in queueExample)
            {
                Console.WriteLine("Position #" + enqueued);
            }

            Console.WriteLine();
            Console.WriteLine("Position #" + queueExample.Dequeue());
            Console.WriteLine("Position #" + queueExample.Peek());
            Console.WriteLine("Position #" + queueExample.Peek());

            // HashSet

            Console.WriteLine("\nHashSet Example:\n");

            var hashExample1 = new HashSet<String>
            {
                "Jay's Portfolio"
            };

            hashExample1.Add("Jay's Portfolio");

            String[] s3 = new String[] { "Jay's Portfolio" };

            Console.WriteLine(hashExample1.Count + " Hash Count");
            Console.WriteLine("HashSet Match: " + hashExample1.Overlaps(s3));

            var hashExample2 = new HashSet<String>
            {
                "HashSet Example"
            };

            hashExample2.Add("HashSet Example");

            Console.WriteLine("\n" + hashExample2.Count + " Hash Count");
            Console.WriteLine("HashSet Match: " + hashExample2.Overlaps(s3));

        }

        // Concurrency

        3 references
        static void AddItem()
        {
            var result = items.TryAdd(1, 2);
            Console.WriteLine("Concurrent Example: " + result);
        }
    }
}
```

4.3.2 Testing Output





This program contains a short selection of features that are available to use in C# (as well as in other languages), to demonstrate how each of them are written. As seen in the debug console above, each section Is outputted in chronological order, except for the concurrency section. This section of code is executed right at the top of the debug console, due to the nature how threads are executed.

5 C++ Code

The following C++ tasks were created to showcase various features via coding tasks that are similar to the previous C# section, recreated in C++ order to formalise myself with the C++ specific syntax.

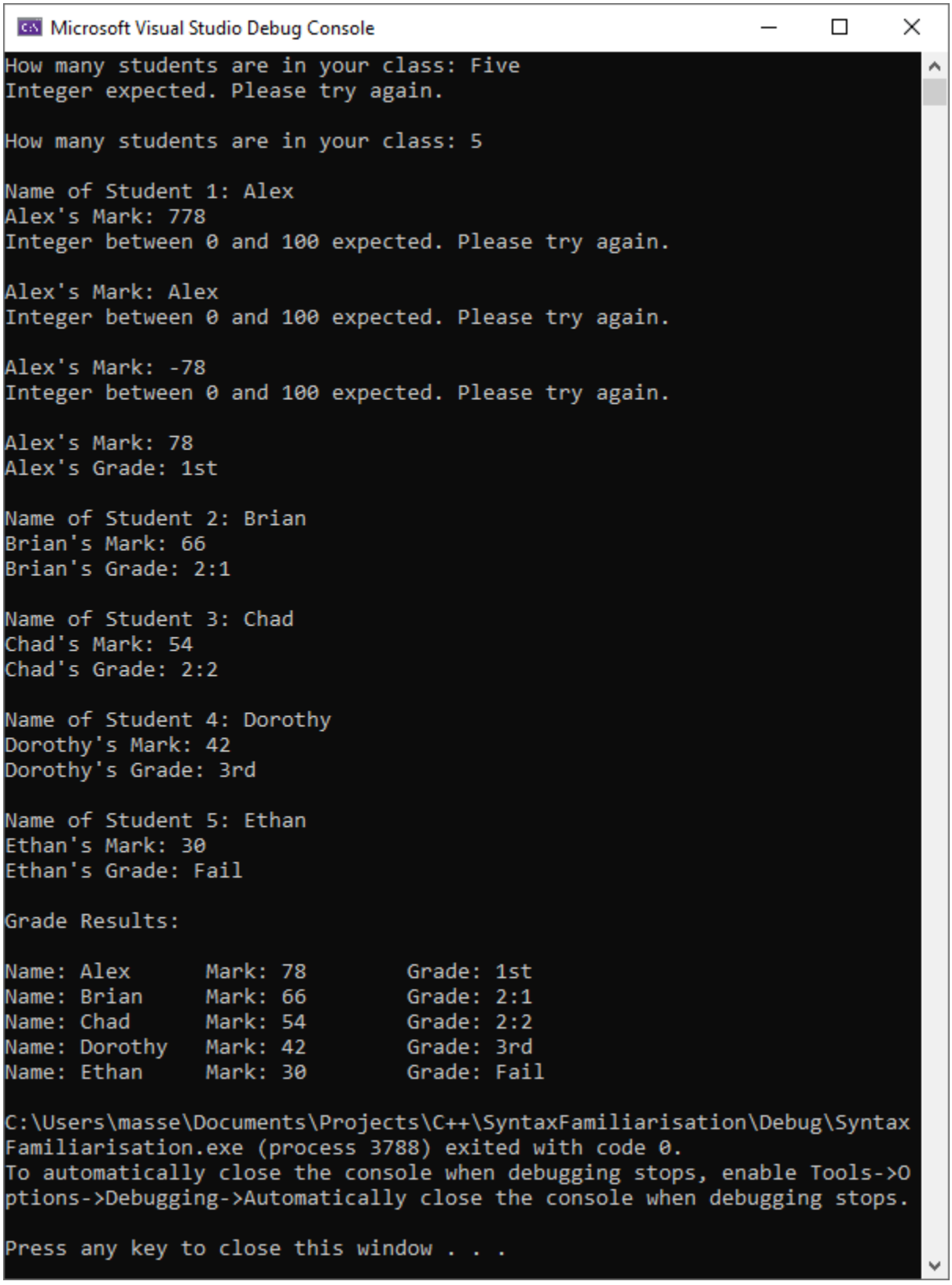
5.1 Syntax Familiarisation

5.1.1 Program Class

```cpp
#include <iostream> // imports cout function
#include <sstream> // imports getline function
#include <vector> // imports vector function

using namespace std;

template <class T> // template added for the following tryparse function

bool try_parse(string input, T& var) // checks input type without crashing if invalid
{
    static const string ws(" \t\f\v\n\r");  // non-user inputs, characters used in formatting

    size_t pos = input.find_last_not_of(ws);

    if (pos != string::npos)
        input.erase(pos + 1);
    else input.clear();

    stringstream buffer(input);

    return buffer >> var && buffer.eof();
}

int main()
{
    int n;
    string input;
    bool inputValid = false;
    bool nameKnown = false;

    vector<int> studentCount;
    vector<string> studentNames;
    vector<int> studentMarks;
    vector<string> studentGrades;

    while (inputValid == false)
    {
        cout << "How many students are in your class: ";
        getline(cin, input);

        if (try_parse(input, n)) { inputValid = true; }
        else { cout << "Integer expected. Please try again.\n" << endl; }
    }

    inputValid = false;

    int students = stoi(input); // converts the now validated number input into an integer

    studentCount.resize(students);
    studentNames.resize(students);
    studentMarks.resize(students);
    studentGrades.resize(students);

    for (std::size_t i = 0; i < studentCount.size(); i++) // std::size_t prevents unsigned mismatch
    {
        studentCount[i] = i + 1;

        while (inputValid == false)
        {
```

```
60              while (nameKnown == false)
61              {
62                  cout << "\nName of Student " << studentCount[i] << ": ";
63                  cin >> studentNames[i];
64                  nameKnown = true;
65
66                  cin.ignore(); // Needed to reset for getline() function.
67              }
68
69              cout << studentNames[i] << "'s Mark: ";
70              getline(cin, input);
71
72              if (try_parse(input, n) && stoi(input) <= 100 && stoi(input) >= 0)
73              {
74                  studentMarks[i] = stoi(input);
75                  inputValid = true;
76              }
77              else
78              {
79                  cout << "Integer between 0 and 100 expected. Please try again.\n" << endl;
80              }
81          }
82
83          nameKnown = false;
84          inputValid = false;
85
86          if (studentMarks[i] >= 70)      { studentGrades[i] = "1st"; }
87          else if (studentMarks[i] >= 60) { studentGrades[i] = "2:1"; }
88          else if (studentMarks[i] >= 50) { studentGrades[i] = "2:2"; }
89          else if (studentMarks[i] >= 40) { studentGrades[i] = "3rd"; }
90          else                            { studentGrades[i] = "Fail"; }
91
92          cout << studentNames[i] << "'s Grade: " << studentGrades[i] << "\n";
93      }
94
95      cout << "\nGrade Results:\n\n";
96
97      for (std::size_t i = 0; i < studentCount.size(); i++) // std::size_t prevents unsigned mismatch
98      {
99          cout << "Name: " << studentNames[i] << "\tMark: " <<
100             studentMarks[i] << "\tGrade: " << studentGrades[i] << "\n";
101     }
102
103     return 0;
104 }
```

5.1.2 Testing Output



This program is the same concept as the syntax familiarisation in the previous C# section. If the user tries to input an invalid input at any stage, the program will disregard their answer, and ask for the input again. Once all the data is successfully inputted, it is outputted back in a user-friendly layout.

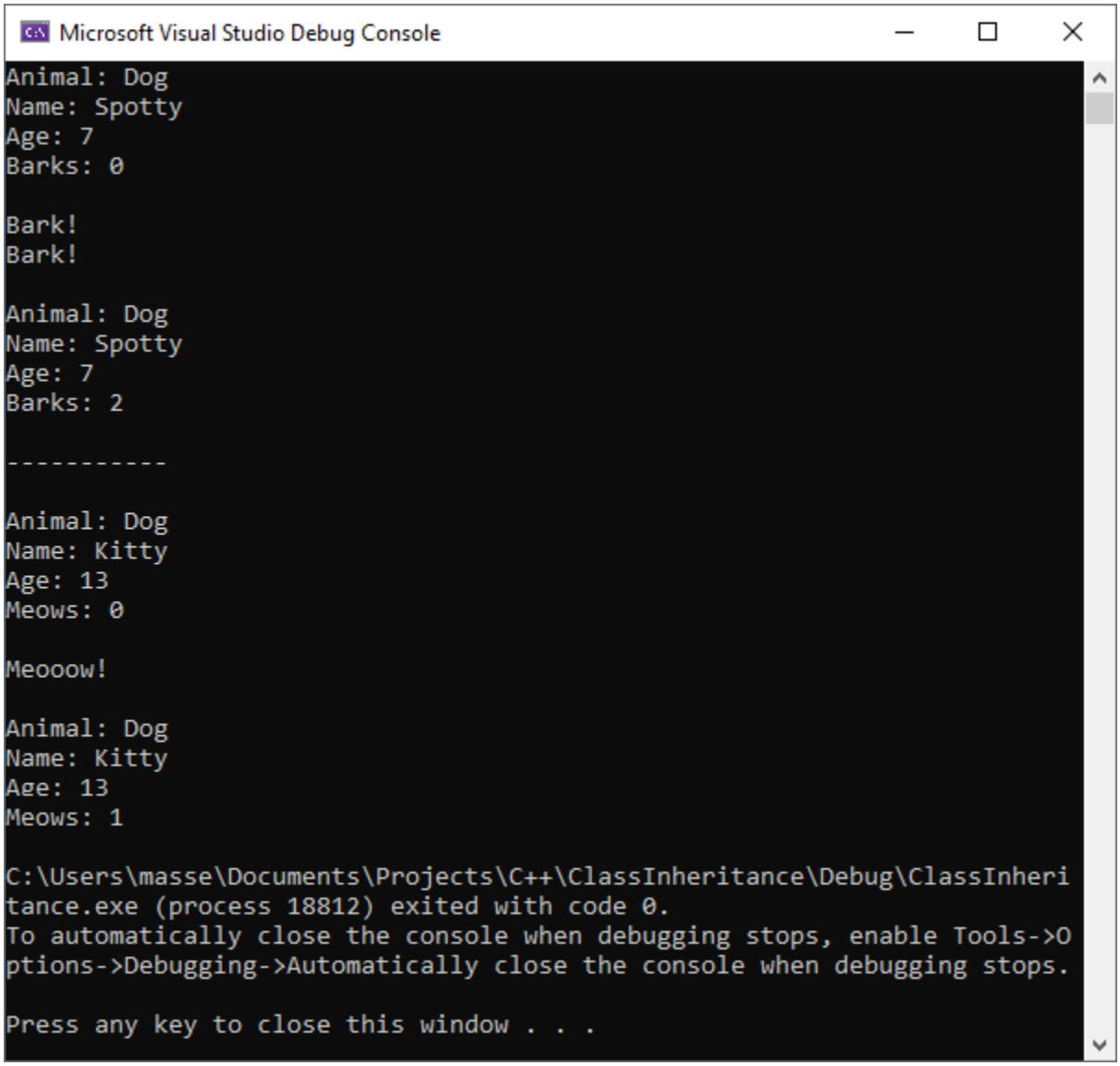
5.2 Class Inheritance

5.2.1 Animal Classes

```
#include <iostream> // imports base functions

using namespace std;

int random(int min, int max)
{
    srand((unsigned)time(0));
    int randomNumber;
    randomNumber = (rand() % max) + min;
    return randomNumber;
}

class Animal
{
    public:
        string animal;
        string name;
        int age = NULL;

        void PrintBaseDetails()
        {
            cout << "\nAnimal: " << animal << endl;
            cout << "Name: " << name << endl;
            cout << "Age: " << age << endl;
        }
};

class Dog : public Animal
{
    public:
        int barks = NULL;
        int barkNumber = NULL;

        void PrintDogDetails()
        {
            cout << "Barks: " << barks << endl << endl;
        }

        void Bark()
        {
            barkNumber = random(1, 3);

            cout << "";

            for (int i = 0; i < barkNumber; i++)
            {
                cout << "Bark!" << endl;
                barks += 1;
            }

            cout << "";
        }
};

class Cat : public Animal
{
    public:
        int meows = NULL;
        int meowLength = NULL;

        void PrintCatDetails()
        {
            cout << "Meows: " << meows << endl;
        }

```

```
66          void Meow()
67          {
68              meowLength = random(1, 5);
69
70              cout << "\nMe";
71
72              for (int i = 0; i < meowLength; i++)
73              {
74                  cout << "o";
75              }
76
77              cout << "w!" << endl;
78              meows += 1;
79          }
80  };
81
82  int main()
83  {
84      Dog spotty;
85      spotty.animal = "Dog";
86      spotty.name = "Spotty";
87      spotty.age = 7;
88      spotty.barks = 0;
89
90      Cat kitty;
91      kitty.animal = "Dog";
92      kitty.name = "Kitty";
93      kitty.age = 13;
94      kitty.meows = 0;
95
96      spotty.PrintBaseDetails();
97      spotty.PrintDogDetails();
98
99      spotty.Bark();
100
101     spotty.PrintBaseDetails();
102     spotty.PrintDogDetails();
103
104     cout << "-----------" << endl;
105
106     kitty.PrintBaseDetails();
107     kitty.PrintCatDetails();
108
109     kitty.Meow();
110
111     kitty.PrintBaseDetails();
112     kitty.PrintCatDetails();
113
114     return 0;
115 }
```

5.2.2 Testing Output



Just like with the previous C++ comparison, this C++ code is similar to the C# example on inheritance of classes. This Is done through the use of a base animal class, followed by cat and dog subclasses. Both animals have their animal type, name, and age, followed by their animal specific features. The dog has an extra *bark()* function, whilst the cat has an extra meow() function. The *bark()* function picks a random number between 1 and 3, outputting barks equal to that number, whilst the *meow()* function picks a random number between 1 and 5, which picks how many o's there should be in the meow. Once each of these outputted, a new "bark" and "meow" total is outputted to the console.

6 References

OBS Studio Contributors. (2020). *OBS Studio*. Retrieved August 2020, from https://obsproject.com/welcome

Aldrich, J. (2004). *A Solution to the Fragile Base Class Problem.* Carnegie Mellon University. Retrieved from http://www.cs.cmu.edu/~aldrich/papers/selective-open-recursion.pdf

Gade, K. (2011, April 6). *Twitter Search is Now 3x Faster*. Retrieved from Twitter Blog: https://blog.twitter.com/engineering/en_us/a/2011/twitter-search-is-now-3x-faster.html

GitHub Inc. (2019, November). *The State of the Octoverse*. Retrieved from Octoverse: https://octoverse.github.com

Goetz, Brian; Peierls, Timothy; Bloch, Joshua J.; Bowbeer, Joseph; Holmes, D.; Lea, D.;. (2006). *Java Concurrency in Practice.* Retrieved from Semantic Scholar: https://pdfs.semanticscholar.org/3650/4bc31d3b2c5c00e5bfee28ffc5d403cc8edd.pdf

Grinnell College. (2000, September). *Reuse Through Inheritance and Polymorphism*. Retrieved from Fundamentals of Computer Science: https://rebelsky.cs.grinnell.edu/Courses/CS152/2000F/Outlines/outline.09.html

Gupta, K. (2018, August 29). *What Big Companies Still Code in Java?* Retrieved from Freelancing Gig: https://www.freelancinggig.com/blog/2018/08/29

Gupta, L. (2019). *Inheritance*. Retrieved from How To Do In Java: https://howtodoinjava.com/oops/java-inheritance

Hartley, R. (2003). *C++ Inheritance*. Retrieved from New Mexico State University: https://www.cs.nmsu.edu/~rth/cs/cs177/map/inheritd.html

JavaTPoint. (2018). *Inheritance in Java*. Retrieved from JavaTPoint: https://www.javatpoint.com/inheritance-in-java

Lemay, L. (1996). *Object-Oriented Programming and Java*. Retrieved from Comenius University: http://www.dmc.fmph.uniba.sk/public_html/doc/Java/ch2.htm

Marlow, S. (2010). *Haskell 2010 Language Report.* Haskell Community. Retrieved from http://www.haskell.org/definition/haskell2010.pdf

Marlow, S. (2013). *Parallel and Concurrent Programming in Haskell.* O'Reilly Media. Retrieved from https://www.oreilly.com/library/view/parallel-and-concurrent/9781449335939

Mikhalchenko, A. (2017). *What is JVM and Why it is Worth to Develop Apps on Java Platform*. Retrieved from Anadea: https://anadea.info/blog/what-is-jvm-and-why-develop-apps-on-java

Nayuki. (2017). *Java SE 5 is the Most Significant Release*. Retrieved from Project Nayuki: https://www.nayuki.io/page/java-se-5-is-the-most-significant-release

Oracle. (2013). *The Java® Language Specification - Java SE 7 Edition.* Redwood City: Oracle America. Retrieved from http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf

Oracle. (2019). *Learning the Java Language*. Retrieved from Oracle Java Documentation: https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Quest. (2020). *Quest Software*. Retrieved July 2020, from http://textadventures.co.uk/quest

Tardi, C. (2019). *What is Moore's Law?* Retrieved from Investopedia: https://www.investopedia.com/terms/m/mooreslaw.asp

Tech Insider. (2007). *JavaSoft Ships*. Retrieved from Programming Environment: https://tech-insider.org/java/research/1996/0123.html

Tempero, E., Yang, H. Y., & Noble, J. (2013). *What Programmers do with Inheritance in Java.* Retrieved from University of Auckland: https://www.cs.auckland.ac.nz/~ewan/qualitas/studies/inheritance/TemperoYangNobleECOOP2013-pre.pdf

U.S. Department of Transportation. (2015). *Traffic Safety Facts*. Retrieved July 2020, from https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115

Vandermeer, J., & Zerfoss, J. (2018). In *Wonderbook: The Illustrated Guide to Creating Imaginative Fiction.* Abrams Image.

Yogen, R. (2018). *Problems With Inheritance in Java*. Retrieved from Java Zone: https://dzone.com/articles/problems-with-inheritance-in-java