|  |
| --- |
| MSC SOFTWARE DEVELOPMMENT |
| Advanced Object Technology |
| Assignment 1 |
|  |
| **Jimmy Collins – R00145569 – Jimmy.Collins@mycit.ie** |
| **3/20/2017** |

|  |
| --- |
|  |

Contents

[Section 1 – Demonstrated Use of Object Orientated Concepts 2](#_Toc476597001)

[Re-Use 2](#_Toc476597002)

[Encapsulation 2](#_Toc476597003)

[Abstraction 2](#_Toc476597004)

[Inheritance 2](#_Toc476597005)

[Overloading 2](#_Toc476597006)

[Dynamic Binding 2](#_Toc476597007)

[Inner Class 2](#_Toc476597008)

[Nested Class 2](#_Toc476597009)

[Abstract Class 2](#_Toc476597010)

[Generic Method 3](#_Toc476597011)

[Generic Class 3](#_Toc476597012)

[Bounding 3](#_Toc476597013)

[Custom Exception Handler Type 3](#_Toc476597014)

[Section 2 – Usage of Design Patterns 4](#_Toc476597015)

[Strategy Pattern 4](#_Toc476597016)

[Observer Pattern 5](#_Toc476597017)

[Factory Pattern 6](#_Toc476597018)

[Section 3 – System UML Diagrams 6](#_Toc476597019)

[Section 4 – Code Screen Captures 6](#_Toc476597020)

[Section 5 – Evaluation of Work 6](#_Toc476597021)

## Section 1 – Demonstrated Use of Object Orientated Concepts

### Re-Use

Logging?

### Encapsulation

TODO – demonstrated by FlowFileStats.java?

### Abstraction

TODO – ensure to differentiate with Encapsulation

### Inheritance

The concept of Inheritance is demonstrated by BinetFileParser extending ParsableFile. The parsing of other types of traffic files could be supported here in the future by introducing subsequent classes that extend ParsableFile and override the necessary functionality.

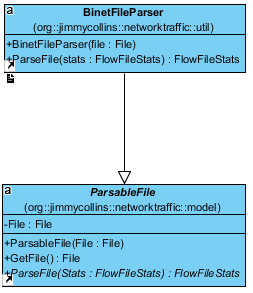


Figure X: Use of Inheritance – NEEDS REVISION

### Overloading

Function overloading is demonstrated by the logging class I use within the application. There are different logging functions that can be used depending on, for example, if a log is being written to the log file, or to the user interface for display to the user.

TODO - UML

Figure X: Logger Class

### Dynamic Binding

Dynamic Binding is used in the logic I use to parse the traffic file. In the example in Figure X below, the compiler cannot determine the type, because the instance of BinetFile is also an instance of ParsableFile, so the type is determined at run-time.

Ref: <http://www.javatpoint.com/static-binding-and-dynamic-binding>

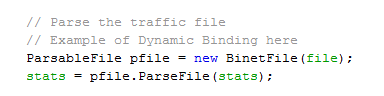


Figure X: Example of Dynamic Binding

### Nested Class

The ParsingObserver class defined in MainUIController is an example of a Nested Class. This class is used as part of my implementation of the Observer pattern (see Section 2), and contains functionality for updating the user interface as parsing of a file progresses.

TODO – Screen capture.

### Inner Class

There are a couple of examples of inner classes. Below is a snippet that illustrates the usage of an anonymous inner class in the function I use to parse the traffic file.

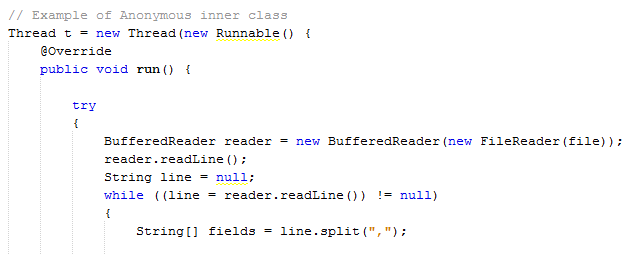


Figure X: Example of Anonymous Inner Class

### Abstract Class

A demonstrated used of an Abstract class is ParsableFile. This defines an abstract class with the relevant fields and methods relevant to parsing a file.

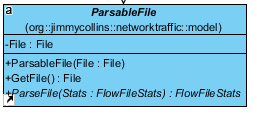


Figure X: ParsableFile Class

### Generic Method

Usage of a generic method is demonstrated by SortMapByValue contained with the static Utility class. TODO: More about this function.

### Generic Class

TODO – ArrayList?

### Bounding

TODO used in Utility.java – GetTopElements

### Custom Exception Handler Type

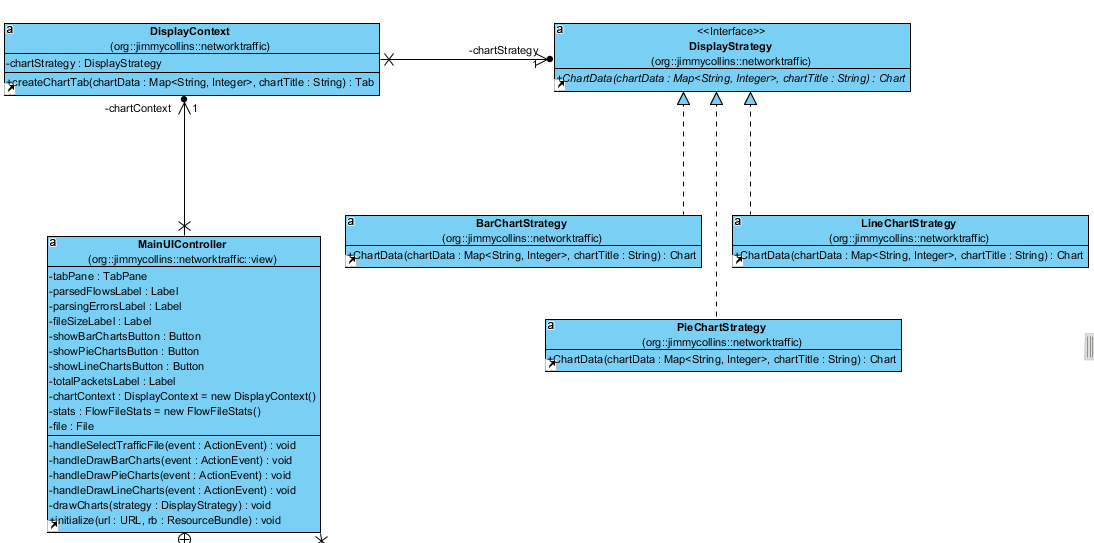
TODO – for parsing bad info from file?

## Section 2 – Usage of Design Patterns

### Strategy Pattern

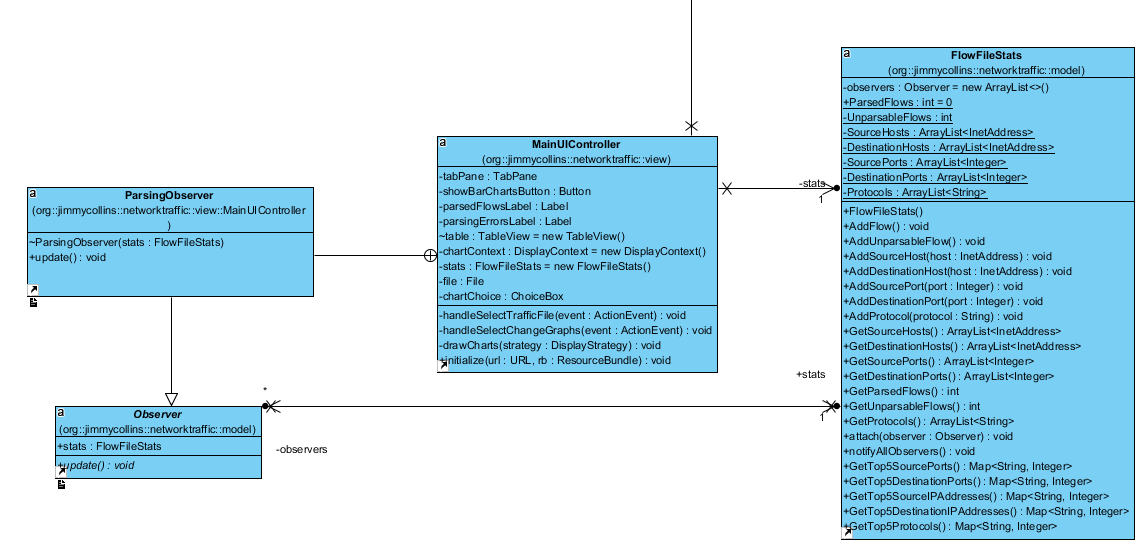
This is used to allow choosing of different chart types to show the data – pie charts and bar charts are supported out-of-the-box. Pie Charts are shown initially in the user interface, but there is an option to show Bar Charts instead.

Support for other charts could be added easily by adding another class and implementing DisplayStrategy.



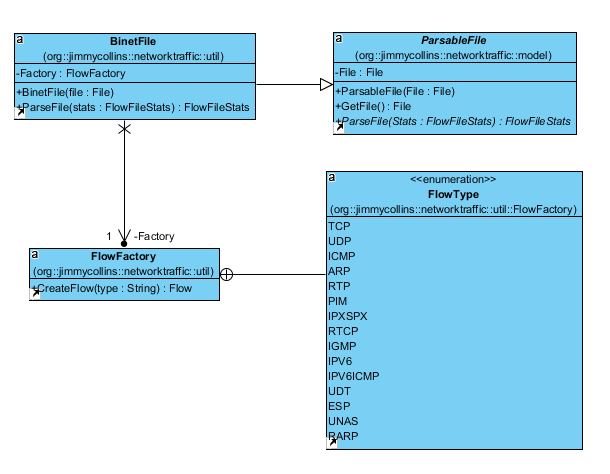
### Observer Pattern

The Observer pattern is used to monitor file parsing information as a file is being parsed, and display a summary on the user interface. FlowFileStats is Observable, and certain fields are observed by MainUIController and updated on the user interface as they change.



### Factory Pattern

The Factory pattern is used in the application when parsing the file. A class called FlowFactory is used to set the type field on the Flow object being created and return a new Flow object.



## Section 3 – System UML Diagrams

TODO – use Visual Paradigm.

## Section 4 – Code Screen Captures

TODO – 1) Generic function.

TODO – 2) File Parsing? Inner class in Main Controller? Threading?

## Section 5 – Evaluation of Work

TODO

Performance – show CPU utilization graph when parsing 300mb file

Better data model. i.e. the objects we use to parse the flows.

Better implementation of Factory pattern.

More intelligent – use to find Botnet activity?

Couple if i18n issues – e.g. MB used in MainController.

Include some graphs from GitHub?