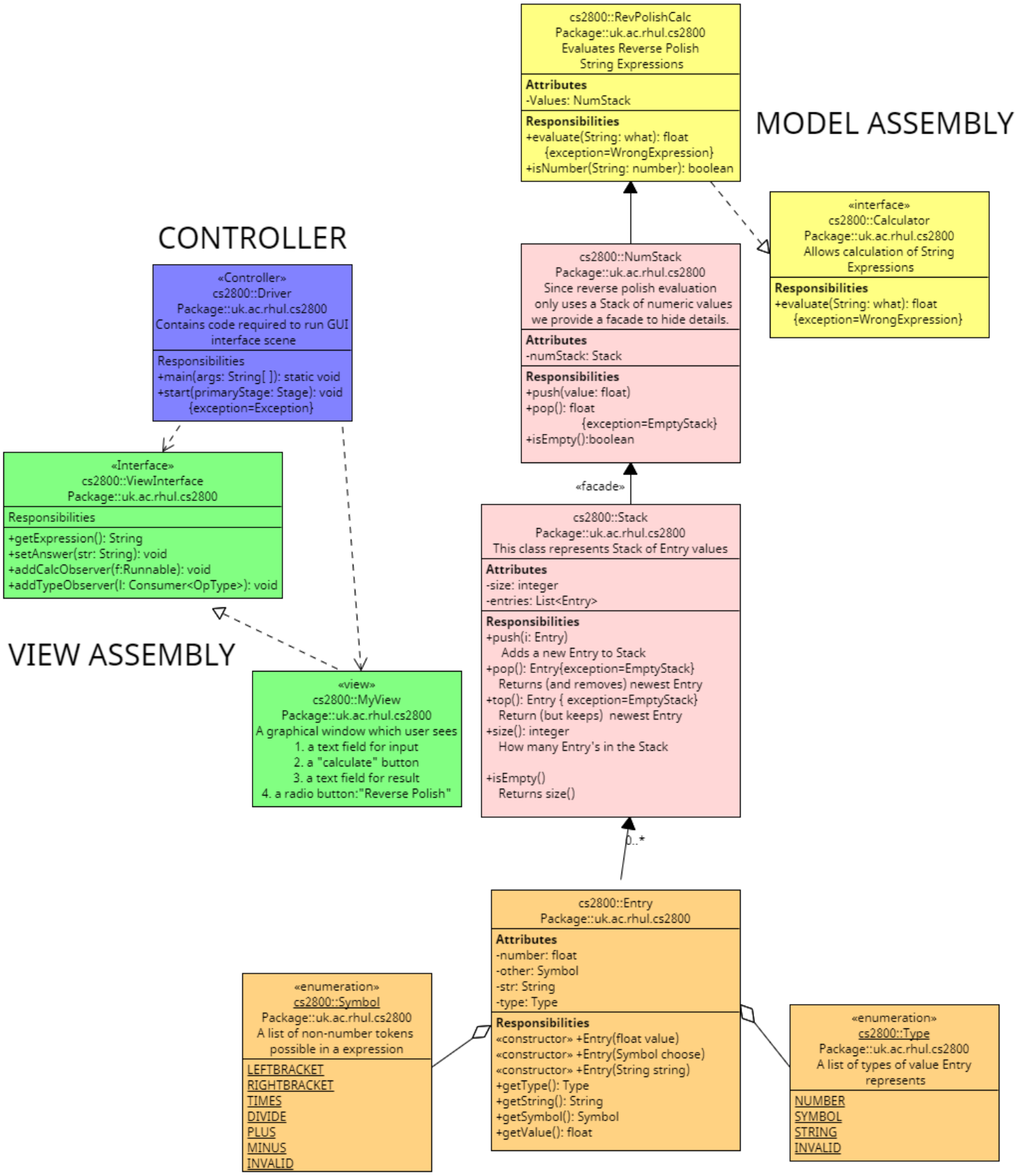
Individual Programming Project Report

Name: Aryaman Rawat
Moodle Name: Aryaman Rawat
Student Number: 100952490
Email Address: zjac180@live.rhul.ac.uk
GitLab URL: https: //gitlab.cim.rhul.ac.uk/zjac180/CS2800.git

UML diagram:


cs2800::RevPolishCalc
Package::uk.ac.rhul.cs2800
Evaluates Reverse Polish String Expressions
Attributes
-Values: NumStack
Responsibilities
+evaluate(String: what): float {exception=WrongExpression}
+isNumber(String: number): boolean
MODEL ASSEMBLY
«interface»
cs2800::Calculator
Package::uk.ac.rhul.cs2800
Allows calculation of String Expressions
Responsibilities
+evaluate(String: what): float {exception=WrongExpression}
CONTROLLER
«Controller»
cs2800::Driver
Package::uk.ac.rhul.cs2800
Contains code required to run GUI interface scene
Responsibilities
+main(args: String[ ]): static void
+start(primaryStage: Stage): void {exception=Exception}
cs2800::NumStack
Package::uk.ac.rhul.cs2800
Since reverse polish evaluation only uses a Stack of numeric values we provide a facade to hide details.
Attributes
-numStack: Stack
Responsibilities
+push(value: float)
+pop(): float {exception=EmptyStack}
+isEmpty():boolean
«Interface»
cs2800::ViewInterface
Package::uk.ac.rhul.cs2800
Responsibilities
+getExpression(): String
+setAnswer(str: String): void
+addCalcObserver(f:Runnable): void
+addTypeObserver(I: Consumer<OpType>): void
«facade»
cs2800::Stack
Package::uk.ac.rhul.cs2800
This class represents Stack of Entry values
Attributes
-size: integer
-entries: List<Entry>
Responsibilities
+push(i: Entry)
Adds a new Entry to Stack
+pop(): Entry{exception=EmptyStack}
Returns (and removes) newest Entry
+top(): Entry { exception=EmptyStack}
Return (but keeps) newest Entry
+size(): integer
How many Entry's in the Stack
+isEmpty()
Returns size()
VIEW ASSEMBLY
«view»
cs2800::MyView
Package::uk.ac.rhul.cs2800
A graphical window which user sees
1. a text field for input
2. a "calculate" button
3. a text field for result
4. a radio button:"Reverse Polish"
0..*
cs2800::Entry
Package::uk.ac.rhul.cs2800
Attributes
-number: float
-other: Symbol
-str: String
-type: Type
Responsibilities
«constructor» +Entry(float value)
«constructor» +Entry(Symbol choose)
«constructor» +Entry(String string)
+getType(): Type
+getString(): String
+getSymbol(): Symbol
+getValue(): float
«enumeration»
cs2800::Symbol
Package::uk.ac.rhul.cs2800
A list of non-number tokens possible in a expression
LEFTBRACKET
RIGHTBRACKET
TIMES
DIVIDE
PLUS
MINUS
INVALID
«enumeration»
cs2800::Type
Package::uk.ac.rhul.cs2800
A list of types of value Entry represents
NUMBER
SYMBOL
STRING
INVALID

To start off the CS2800 project I delivered a horizontal slice for the stack assembly to implement a fully working stack which represents Entry values.Using TDD approach,I created the Stack class along with the Entry class.My UML diagram consists of two assemblies view and model.I decided to use bottom to top approach to fulfill the requirements specified in the UML.

This UML diagram was created using class diagrams which contain the name of the class,attributes and methods.Once the vertical slicing for stack was implemented with limited functionality, a horizontal slicing was conducted on each single class to be perfected and gradually strengthen the core components for the calculator.

The Entry class uses two instances of other enum classes.The use of aggregation method benefits the program as there is no lifetime implication as they can be built and destroyed separately.

Association is a key relationship used; the class Stack depends on Entry, NumStack depends on Stack and RevPolishCalc depends on NumStack using one class without another will result in the program not working.These classes are designed to work together.

The view assembly classes use dependency as it makes calls to methods of other classes.MyView class is dependent on ViewInterface as that's how the users will see and interact with the interface.

There were no additional classes added to the UML other than improving the UML based on my modifications.Since my program only implements NumStack,I didn’t see the need for adding OpStack and StrStack classes.Due to the deadline,these features were left out as I didn’t have time to add a standardCalc which was in the original UML.Refactoring to remove code smells allowed my program become more efficient by reducing cohesion and increase coherence.The facade design pattern was used which is a type of structural pattern as its required to hide complexities by adding interface to current system.This pattern used the class Stack which essentially provided for NumStack,OpStack and StrStack although due to minimum time it was only used in NumStack.

Main reason for the UML not being similar to the original is simply because of being time constraint,given more time I would have implemented similar/additional classes according to the original UML.

Exam Questions:

1) Aggregation relationship is when an object includes objects from other classes.Aggregation relationship is a special type of association in which objects are assembled together to create more complex objects.This is useful because it describes a group of objects and how to interact with them.

Composition relationship is when an object is built with an object from anotherclass.It is a form of aggregation.Composition association relationship specifies lifetime of part classifier and is dependent on whole classifiers lifetime meaning if if the whole classifier dies the part should die too.

Both aggregation and composition are subsets of association.This means an object of the owner class can be the owner of an object from another class.Both relationships have a child object belonging to a single parent object.The similarities show both concepts are somewhat connected.

2a) Primitive obsession smell is a code smell where the primitive data types are used unnecessarily.It can occur in a single field or a group of related fields as having a primitive data types for example string representing multiple fields name,address and ID,they can't contain behavior therefore has to be stored in containing class.Losing their type safety means a value can be assigned to a wrong field as a string is representing many fields name, address and ID.

2b) For example we have a class Person which contains fields id,first name,second name, address, city and postcode all being string type data.At some point a wrong value may be assigned to a wrong parameter slot.Refactoring can help majorly here as we can reconstruct the class so we have a common group put together for example address,postcode and city can all be grouped so we can refactor these into separate class address for example.This allows encapsulation as all the logic to do with address is inside the address class.We can follow similar approach for other parameters.Refactoring primitive obsession increases cohesion as as it increases the degree to witch elements of certain module belong together as shown in the given example.

3a)Configuration testing is required as configuration faults often occur and cause failures if not corrected. This allows assessing overall quality of code and improving it by fixing errors.This type of testing is conducted by recording a set of constraints between different components and any change in component ,we should check the configuration is maintained.

3b)Compliance testing is done to identify areas which have problems and create tests to check that set of conditions to fix those problems before being released into production.This test is conducted on standardized test sets that often have good coverage of logic of the system.

4a)Using a factory which is a creational pattern is useful to create flyweight instances because it allows controlling the object creation and will help prevent inefficiency.Factory pattern is mainly used in graphical objects and is a variety of objects implementing the same interface.

4b)Using a singleton pattern is suitable here because ImageObjectFactory contains a list of Flyweight Sprite objects.Singleton will allow instantiation of class to just one object allowing global point of access to unique object which in this case is needed for the FlyweightSprite object.This allows the instance to be used anywhere in the code.