REPORT 3

JINGMIAN ZHANG

2016 DECEMBER

Table of Contents

Contents

[1 Introduction 1](#_Toc468664978)

[2 How to use the calculator 2](#_Toc468664979)

[3 The static design of the calculator 3](#_Toc468664980)

[3.1 The static design before modification 3](#_Toc468664981)

[3.2 The static design after modification 4](#_Toc468664982)

[4 Dynamic model of some operations 5](#_Toc468664983)

[4.1Dynamic model before modification 5](#_Toc468664984)

[4.2Dynamic model after modification 6](#_Toc468664985)

# 1 Introduction

Design pattern and framework are one of the fundamental cornerstones in designing reliable software system in computer science. It is imperative that all programmers can understand these crucial concepts in object-oriented programming. This is report 3 of the course EECS 3311 which is generated corresponding to the specification in the report 3 specification as well as the program text given. In report 3, I Modify the routine descriptionOfProgram by replacing the two if…then…else statements with a better design, and I also modify the calculator MVC model to use the visitor pattern and the popOperandOffProgramStack. Through the process of this report, I have learned about the features and properties of Eiffel language and object-oriented concept considerably.

# 2 How to use the calculator

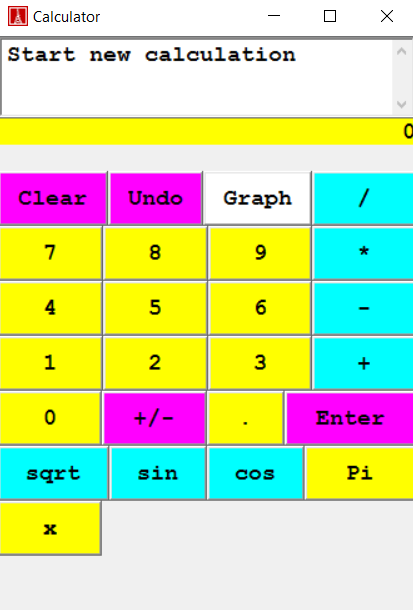


Figure1: how to use the calculator

The interface of the MVC calculator is illustrated in Figure 1. There are two views in the calculator, graph view and normal view, which can be switched through clicking the GRAPH button on the calculator. The entry of the formula is in postfix notation, enter is used purposely to indicate the end of a number. A simple example of using the calculator to sketch the graph y = x \* cos(x), then x / cos(x), then x \* sin(x) will be the instructions as followed: x x cos \* Graph CalculatorPad Undo / Graph CalculatorPad Undo Undo sin \* Graph CalculatorPad.

# 3 The static design of the calculator

# 3.1 The static design before modification

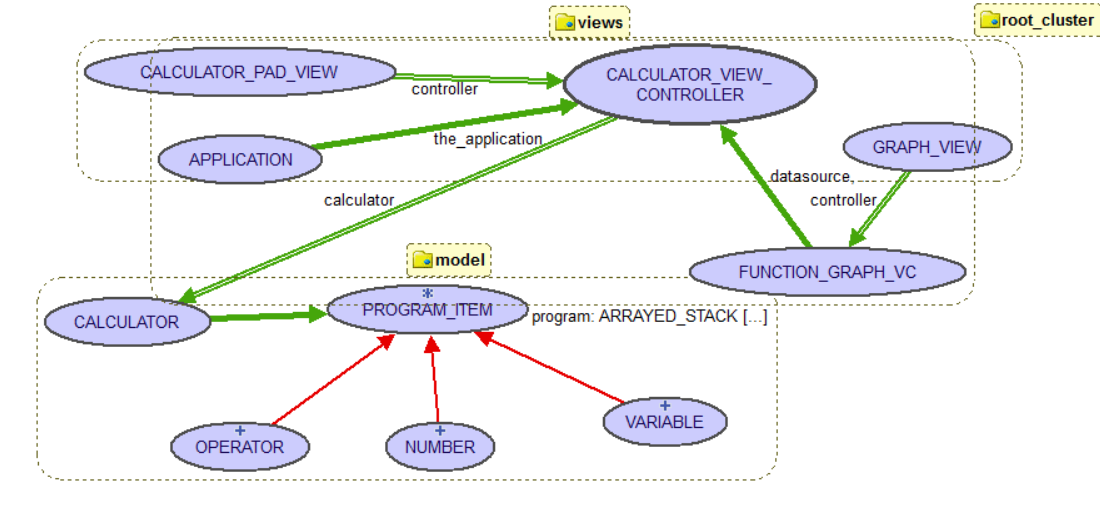


Figure2: static design of the calculator before modification

Figure2 illustrates the static design of the calculator before the modification. As a typical MVC program, the calculator has three components: model, view and control. Model, which does all the computation is the class CALCUALTOR. Since the calculator have the calculator view and the function view, two class for view exist, which are GRAPH\_VIEW and CALCULATOR\_PAD\_VIEW. To provide communications for the two view to their common model, the classes CALCULATOR\_VIEW\_CONTROLLER and FUNCTION\_GRAPH\_VC are created. Client use the class APPLICATION to use the calculator. Moreover, the controller has a computational model CACULATOR, which has a deferred class PROGRAM\_ITEM to serve its own functionalities. Class OPERATOR, NUMBER, and VARIABLE inherits from PROGRAM\_ITEM to implement its features and are used by CALCULATOR.

# 3.2 The static design after modification

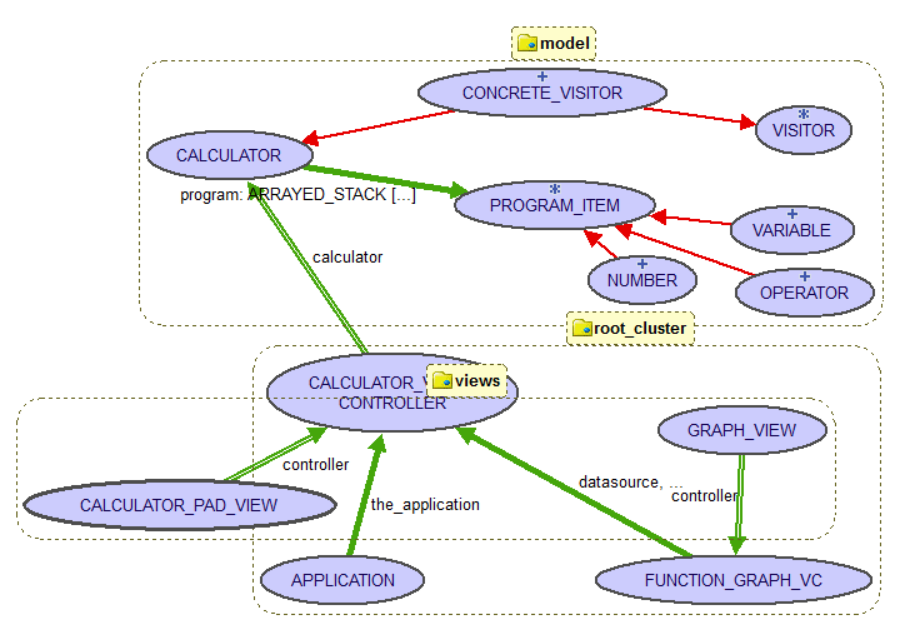


Figure3: static design of the calculator after modification

Figure3 illustrates the static design of the calculator after the modification. As a typical MVC program, the calculator has three components: model, view and control. The primarily different part of this implementation, which follows the visitor patterns are related to the model part of the design. To implement the visitor pattern, the VISITOR deferred class is first created to provide the interface for three visitor methods, respectively for visiting OPERATOR, NUMBER and VARIABLE. Then , make CONCRETE\_VISITOR inherits VISITOR and implements those methods, whose content are extracted from the original *popOperandOffProgramStack* feature. Then, the  *accept* features are added to the class PROGRAM\_ITEM first as a deferred feature, then to the three subclasses as effective features. Moreover, a *describe* feature is added the same way the extract the content of the method *descriptionOfProgram*, as a usage of polymorphism. Finally, make CONCRETE\_VISTOR inherits CALCULATOR so it can use the features and attributes in CALCULATOR when implementing the visitor features, which facilitates the process.

# 4 Dynamic model of some operations

# 4.1Dynamic model before modification

CALCULATOR

1

2

PROGRAM\_ITEM

Scenario 1: popOperandOffProgramStack

1. Calculator use has-a relationship to obtain an instance of Program\_item

2. Program\_item is used to attach object to and carry out corresponding calculations.

CALCULATOR

1

2

PROGRAM\_ITEM

Scenario 2: descriptionOfProgram

1. Calculator use has-a relationship to obtain an instance of Program\_item
2. Program\_item is used to attach object to and carry out corresponding calculations.

# 4.2Dynamic model after modification

OPERATOR

NUMBER

VARIABLE

ARRAYED\_STACK[PROGRAM\_ITEM]

CONCRETE\_VISITOR

PROGRAM\_ITEM

CALCULATOR

1

2

3

7 4 8 5 6 9

Scenario 1: popOperandOffProgramStack

1. Calculator use has-a relationship to obtain an instance of Program\_item

2,3:PROGRAM\_ITEM uses accept method passing instances of CONCRETE\_VISITOR and ARRAY\_STACK[PROGRAM\_ITEM]

4,5,6:CONCRETE\_VISITOR uses accept method depending on polymorphism of class OPERATOR, NUMBER and VARIABLE.

7,8,9: In their respective accept method, the three class call visit\_operator, visit\_number, visit\_variable respectively.

ARRAYED\_STACK[PROGRAM\_ITEM

OPERATOR

NUMBER

VARIABLE

CALCULATOR

1

2

PROGRAM\_ITEM

3 4 5

Scenario 2: descriptionOfProgram

1. Calculator use has-a relationship to obtain an instance of ARRAYED\_STACK[PROGRAM\_ITEM
2. Traverse over ARRAY\_STACKED[PROGRAM\_ITEM] and obtain an instance of PROGRAM\_ITEM each iteration

3,4,5: Instance of PROGRAM\_ITEM call the feature describe that are implemented in class OPERATOR, VARIABLE and NUMBER based on polymorphism