**Assignment 9**

**Rasika Mohod**

rmohod@gmu.edu

G01044774

**Overview:**

**Variable** class is a generic class which implements a variable of any generic type and parameter <T> stores the type of variable. Thus T is a generic type parameter which can accept any type of variable. **VariableCient** class is a client class for Class Variable and it exercises the operations on Variable class. **VariableTest** is a JUnit Test class for implementation of Class Variable. This class has two different types of Test cases to verify the operations on Variable class.

Variable class contains a **swap** method to perform swap operation on two different variables of same type. This method takes input a parameter **other**, the variable with which **this** variable is to be swapped. The parameter **other** is also variable of type T. Thus two variables of any (same) type can be initialized with class Variable and then use its swap method to swap the values of those two variables**.**

**Design Benefits:**

* The Variable class creates a platform for any type of variables to be created and their values being swapped any number of times efficiently.
* The swap method accepts a parameter of type Variable<T> which is same as type of *this* variable and thus ensures that same type of values are swapped and gives compile time error if two different types of variables are given to be swapped.
* The single Variable class enables clients to perform swapping operation on any type of variables.
* This class provides efficient code to perform swapping of variables frequently.

**Design Details: Variable.java**

This class has two constructors:

The **default constructor** is designed for clients who wants to initialize the variable value later and not at the time of its creation.

**public** Variable(){}

The **parameterized constructor** is designed for clients who wants to initialize the variable value at the time of its creation of Variable object. This constructor sets the value of variable and also verifies the representation of variable. ID representation is not correct the client is notified with proper exception.

**public** Variable(T \_value) **throws** Exception{

T temp\_value = \_value;

**this**.value = temp\_value;

RepOk();

}

The **method RepOk()** is used to check if representation is correct and throws Exception if representation is not correct.

**public** **void** RepOk() **throws** Exception{

T temp\_value = **this**.value;

**if**(temp\_value==**null**){

**throw** **new** Exception("Rep is not correct!! Value of a variable cannot be null.");

}

}

The **method setValue()** takes an input the value of type T and sets the value of variable.

**public** **void** setValue(T newvalue){

T temp\_value = newvalue;

**this**.value = temp\_value;

}

The **method getValue()** first verifies if representation of variable is correct or not. If representation is correct then it returns the value of variable else notifies user by throwing an exception. This method ensures that no client can retrieve value of a variable which does not abide by the representation rule.

**public** T getValue() **throws** Exception{

RepOk();

T temp\_value = **this**.value;

**return** temp\_value;

}

The **method swap()** is designed to perform swap operation on two different variables of same type. This method swaps the value of this variable with the value of other variable taken as input which is also of same type as of this variable.

**public** **void** swap(Variable<T> other) **throws** Exception

{

T tmp = **this**.value;

**this**.setValue((other).getValue());

other.setValue(tmp); }

The **method toString()** is overriden to return the String representation of variable.

**public** String toString(){

String variableValue = "";

**try** {

variableValue = ""+**this**.getValue()+"";

}

**catch** (Exception e) {

System.***out***.println("Exception occured: "+e);

}

**return** variableValue;}