Experiment 6
Operator Overloading

Objectives

What operator overloading is and how it simplifies programming.
To overload operators for user-defined classes.
To overload unary and binary operators.
To convert objects from one class to another class.

Prelab Activities

Programming Output

For each of the given program segments, read the code and write the output in the space provided below each program. [Note: Do not execute these programs on a computer.]

For Programming Output Exercises 1–2, use the following class definition.

```
1 // Array.h
2 // Simple class Array (for integers)
3 #ifndef ARRAY_H
4 #define ARRAY_H
5
6 #include <iostream>
7 using namespace std;
8
9 // class Array definition
10 class Array
11 {
12    friend ostream &operator<<( ostream &, const Array & );
13    friend istream &operator>>( istream &, Array & );
14
15 public:
16    Array( int = 10 ); // default constructor
17    Array( const Array & ); // copy constructor
18    ~Array(); // destructor
19    int getSize() const; // return size
20    const Array &operator=( const Array & ); // assignment operator
21    bool operator==( const Array & ) const; // equality operator
22
23    // determine if two arrays are not equal and
24    // return true, otherwise return false (uses operator==)
25    bool operator!=( const Array &right ) const
26    {
27       return ! ( *this == right );
28
29    } // end function operator!=
30
31    int &operator[]( int ); // subscript operator
32    const int &operator[]( int ) const; // subscript operator
33    static int getArrayCount(); // return number of
34    // arrays instantiated
35 private:
36    int size; // size of array
37    int *ptr; // pointer to first element of array
```

```
38    static int arrayCount; // number of Arrays instantiated
39
40 }; // end class Array
41
42 #endif // ARRAY_H
```

```
1  // Array.cpp
2  // Member function definitions for class Array
3  #include <iostream>
4  #include <iomanip>
5  #include <cstdlib>
6  #include <new>
7  using namespace std;
8
9  #include "Array.h"
10
11 // initialize static data member at file scope
12 int Array::arrayCount = 0; // no objects yet
13
14 // default constructor for class Array (default size 10)
15 Array::Array( int arraySize )
16 {
17    size = ( arraySize > 0 ? arraySize : 10 );
18    ptr = new int[ size ]; // create space for array
19    ++arrayCount; // count one more object
20
21    for ( int i = 0; i < size; i++ )
22       ptr[ i ] = 0; // initialize array
23
24 } // end class Array constructor
25
26 // copy constructor for class Array
27 // must receive reference to prevent infinite recursion
28 Array::Array( const Array &arrayToCopy ) : size( arrayToCopy.size )
29 {
30    ptr = new int[ size ]; // create space for array
31    ++arrayCount; // count one more object
32
33    for ( int i = 0; i < size; i++ )
34       ptr[ i ] = arrayToCopy.ptr[ i ]; // copy arayToCopy into object
35
36 } // end copy constructor
37
38 // destructor for class Array
39 Array::~Array()
40 {
41    delete [] ptr; // reclaim space for array
42    --arrayCount; // one fewer object
43
44 } // end class Array destructor
45
46 // get size of array
47 int Array::getSize() const
48 {
49    return size;
50
51 } // end function getSize
52
53 // overloaded assignment operator
54 // const return avoids: ( a1 = a2 ) = a3
55 const Array &Array::operator=( const Array &right )
```

```
{
   if ( &right != this ) { // check for self-assignment

      // for arrays of different sizes, deallocate original
      // left side array, then allocate new left side array
      if ( size != right.size ) {
         delete [] ptr; // reclaim space
         size = right.size; // resize this object
         ptr = new int[ size ]; // create space for array copy

      } // end if

      for ( int i = 0; i < size; i++ )
         ptr[ i ] = right.ptr[ i ]; // copy array into object

   } // end if

   return *this; // enables x = y = z;

} // end function operator=

// determine if two arrays are equal and
// return true, otherwise return false
bool Array::operator==( const Array &right ) const
{
   if ( size != right.size )
      return false; // arrays of different sizes

   for ( int i = 0; i < size; i++ )

      if ( ptr[ i ] != right.ptr[ i ] )
         return false; // arrays are not equal

   return true; // arrays are equal

} // end function operator==

// overloaded subscript operator for non-const Arrays
// reference return creates an lvalue
int &Array::operator[]( int subscript )
{
   // check for subscript out of range error
   if ( subscript < 0 || subscript >= size ) {
      cout << "\nError: Subscript " << subscript
           << " out of range" << endl;

      exit( 1 ); // terminate program; subscript out of range

    } // end if

    return ptr[ subscript ]; // reference return

} // end function operator[]

// overloaded subscript operator for const Arrays
// const reference return creates an rvalue
const int &Array::operator[]( int subscript ) const
{
   // check for subscript out of range error
   if ( subscript < 0 || subscript >= size ) {
      cout << "\nError: Subscript " << subscript
```

```
117          << " out of range" << endl;
118
119       exit( 1 ); // terminate program; subscript out of range
120
121    } // end if
122
123    return ptr[ subscript ]; // const reference return
124
125 } // end function operator[]
126
127 // return number of Array objects instantiated
128 // static functions cannot be const
129 int Array::getArrayCount()
130 {
131    return arrayCount;
132
133 } // end function getArrayCount
134
135 // overloaded input operator for class Array;
136 // inputs values for entire array
137 istream &operator>>( istream &input, Array &a )
138 {
139    for ( int i = 0; i < a.size; i++ )
140       input >> a.ptr[ i ];
141
142    return input; // enables cin >> x >> y;
143
144 } // end function operator>>
145
146 // overloaded output operator for class Array
147 ostream &operator<<( ostream &output, const Array &a )
148 {
149    int i;
150
151    for ( i = 0; i < a.size; i++ ) {
152       output << setw( 12 ) << a.ptr[ i ];
153
154       if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
155          output << endl;
156
157    } // end for
158
159    if ( i % 4 != 0 )
160       output << endl;
161
162    return output;
163
164 } // end function operator<<
```

1. What is output by the following code? Use the definition of class Array provided above.

```
1  #include "Array.h"
2
3  int main()
4  {
5     cout << "# of arrays instantiated = "
6          << Array::getArrayCount()  << '\n';
7
8     Array integers1( 4 );
9     Array integers2;
10
```

```
11     cout << "# of arrays instantiated = "
12          << Array::getArrayCount()  << "\n";
13
14     Array integers3( 8 ), *intptr = &integers2;
15
16     cout << "# of arrays instantiated = "
17          << Array::getArrayCount()  << "\n\n";
18  } // end main
```

2. What is the output of the following program? Use the Array class shown above.

```
1  #include "Array.h"
2
3  int main()
4  {
5     Array integers1( 4 );
6     Array integers2( 4 );
7
8     if ( integers1 != integers2 )
9        cout << "Hello";
10    else
11       cout << "Goodbye" << endl;
12 } // end main
```

Lab Exercises

Lab Exercise 1 — Vector Implementation

The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. UML Diagram
4. Sample Output
5. Test Code Template
6. Problem-Solving Tips

The test program template represents a complete working C++ program test application. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, write your C++ code. Compile and execute the program. Compare your output with the sample output provided.

Lab Objectives

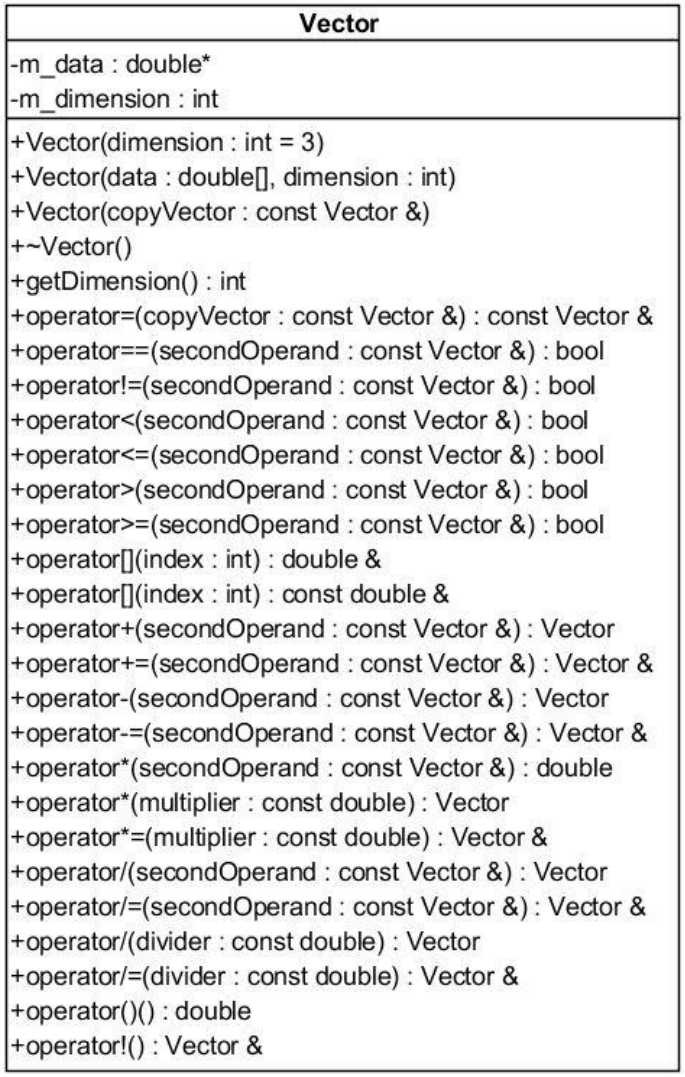
In this lab, you will practice:

- Overloading the + operator to allow String objects to be concatenated.
- Writing function prototypes for overloaded operators.
- Using overloaded operators.

Description of the Problem

Implement a Vector class that represents a mathematical vector. Look at the UML Diagram and implement required functionalities. Required functions is explained in the UML Diagram section. You are given a non-complete test code. Complete the test code according to the sample output. Test your implementation.

UML Diagram



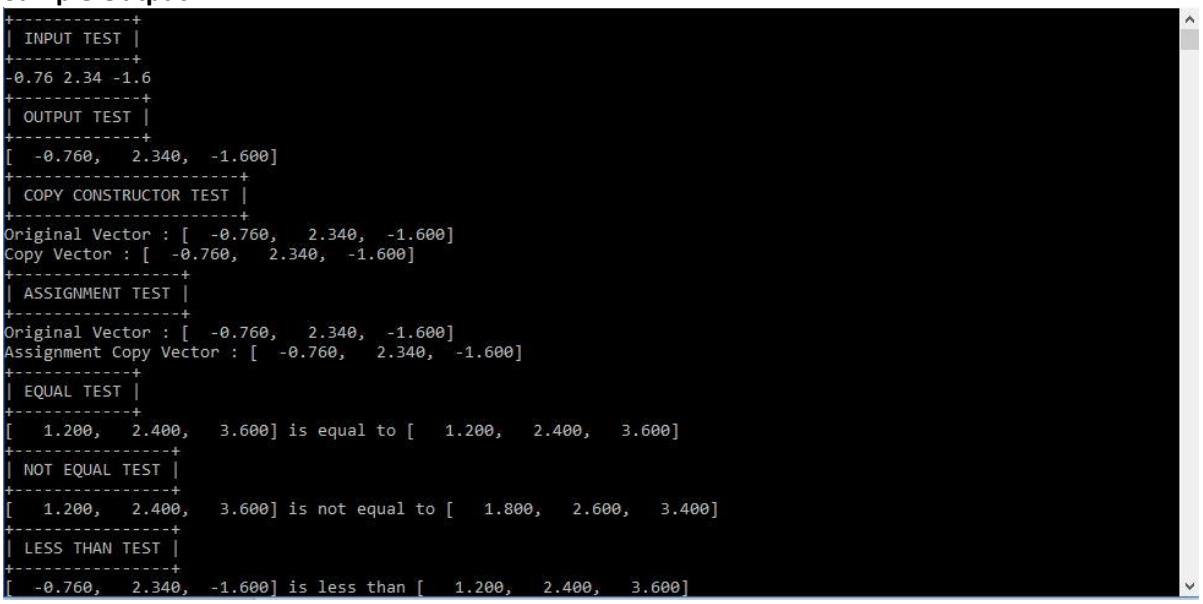
Members:

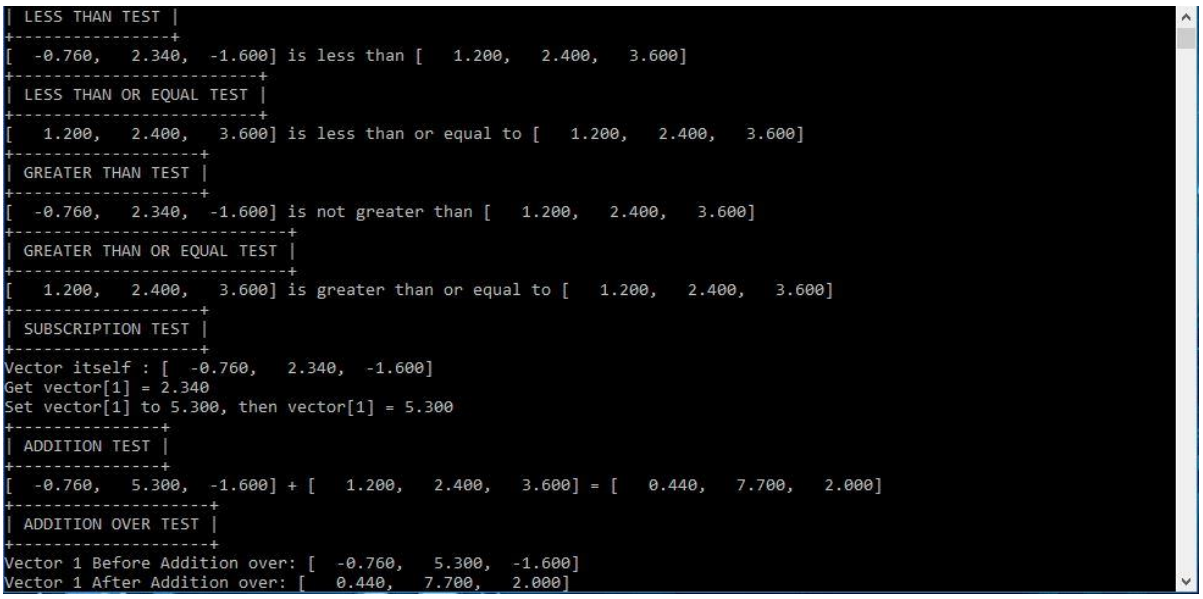
- m_dimension: size of the vector
- m_data: a double array to keep the raw data

Functions:

- Vector(dimension = 3): Default parameter constructor
- Vector(data,dimension): Overloaded constructor with a double array and dimension parameter
- Vector(copyVector): Copy constructor
- ~Vector(): Destructor. You have to free the data array to handle the memory leak
- getDimension() : Returns the dimension of the vector
- Equal operator: Return true, if the vectors are equal
- Not equal operator: Returns true if the vectors are not equal
- Less than, greater than operators decides according to the magnitude of the vectors.
- Assign operator: Copies the vector given in the argument
- Subscription operator: Returns the vector element according to the given index

Sample Output





```
+--------------------+
Vector 1 Before Addition over: [  -0.760,   5.300,  -1.600]
Vector 1 After Addition over: [   0.440,   7.700,   2.000]
+------------------+
| SUBTRACTION TEST |
+------------------+
[  -0.760,   5.300,  -1.600] - [   1.200,   2.400,   3.600] = [  -1.960,   2.900,  -5.200]
+-----------------------+
| SUBTRACTION OVER TEST |
+-----------------------+
Vector 1 Before Substruction over: [  -0.760,   5.300,  -1.600]
Vector 1 After Substruction over: [  -1.960,   2.900,  -5.200]
+------------------+
| DOT-PRODUCT TEST |
+------------------+
[  -0.760,   5.300,  -1.600] * [   1.200,   2.400,   3.600] = 6.048
+-----------------------------+
| CONSTANT MULTIPLICATIN TEST |
+-----------------------------+
[  -0.760,   5.300,  -1.600] * 2.000 = [  -1.520,  10.600,  -3.200]
+-----------------------------+
| CONSTANT MULTIPLICATIN OVER |
+-----------------------------+
Vector 1 Before Constant Multiplication over: [  -0.760,   5.300,  -1.600]
Vector 1 After Constant Multiplication over: [  -1.520,  10.600,  -3.200]
+---------------+
| DIVISION TEST |
+---------------+
[  -0.760,   5.300,  -1.600] / [   1.200,   2.400,   3.600] = [  -0.633,   2.208,  -0.444]
+---------------+
```

```
Vector 1 After Constant Multiplication over: [  -1.520,  10.600,  -3.200]
+---------------+
| DIVISION TEST |
+---------------+
[  -0.760,   5.300,  -1.600] / [   1.200,   2.400,   3.600] = [  -0.633,   2.208,  -0.444]
+---------------+
| DIVISION OVER |
+---------------+
Vector 1 Before Division over: [  -0.760,   5.300,  -1.600]
Vector 1 After Division over: [  -0.633,   2.208,  -0.444]
+------------------------+
| CONSTANT DIVISION TEST |
+------------------------+
[  -0.760,   5.300,  -1.600] / 2.000 = [  -0.380,   2.650,  -0.800]
+------------------------+
| CONSTANT DIVISION OVER |
+------------------------+
Vector 1 Before Division over: [  -0.760,   5.300,  -1.600]
Vector 1 After Division over: [  -0.380,   2.650,  -0.800]
+----------------+
| MAGNITUDE TEST |
+----------------+
MAG( [  -0.760,   5.300,  -1.600] ) = 5.588
+-------------------+
| INVERSE DIRECTION |
+-------------------+
Original Vector: [  -0.760,   5.300,  -1.600]
Inversed Vector: [   0.760,  -5.300,   1.600]
Press any key to continue . . .
```

Test Code Template

```
/****************************************
*****************************************
* IDE : Visual Studio 2015              *
* Author : Cihan UYANIK                 *
* Experiment 4: Operator Overloading    *
*****************************************/
#include "Vector.h"

void TEST_Input(Vector& vector)
{
	cout << "+------------+" << endl
		<< "| INPUT TEST |" << endl
		<< "+------------+" << endl;
	cin >> vector;
}
```

```
void TEST_Output(Vector& vector)
{
	cout << "+-------------+" << endl
		<< "| OUTPUT TEST |" << endl
		<< "+-------------+" << endl;
	cout << vector << endl;
}

void TEST_CopyConstructor(Vector& vector)
{
	cout <<"+-----------------------+" << endl
		<< "| COPY CONSTRUCTOR TEST |" << endl
		<< "+-----------------------+" << endl;

	Vector copy_vector(vector);

	cout << "Original Vector : " <<vector<< endl<< "Copy Vector : "<<copy_vector
<< endl;
}

void TEST_Assignment(Vector& vector)
{

	cout<< "+-----------------+" << endl
		<< "| ASSIGNMENT TEST |" << endl
		<< "+-----------------+" << endl;

	Vector copy_vector;

	copy_vector = vector;

	cout << "Original Vector : " << vector << endl << "Assignment Copy Vector : "
<< copy_vector << endl;
}

void TEST_Equal(Vector& vector1, Vector& vector2)
{
	cout <<"+------------+" << endl
		<< "| EQUAL TEST |" << endl
		<< "+------------+" << endl;

	if (vector1 == vector2)
	{
		cout << vector1 << " is equal to " << vector2 << endl;
	}
	else
	{
		cout << vector1 << " is not equal to " << vector2 << endl;
	}
}

void TEST_Not_Equal(Vector& vector1, Vector& vector2)
{
// Implement the function
}
```

```
void TEST_LESS_THAN(Vector& vector1, Vector& vector2)
{
	cout << "+----------------+" << endl
		<< "| LESS THAN TEST |" << endl
		<< "+----------------+" << endl;

	if (vector1 < vector2)
	{
		cout << vector1 << " is less than " << vector2 << endl;
	}
	else
	{
		cout << vector1 << " is not less than " << vector2 << endl;
	}
}

void TEST_LESS_THAN_OR_EQUAL(Vector& vector1, Vector& vector2)
{
// Implement the function
}

void TEST_GREATER_THAN(Vector& vector1, Vector& vector2)
{
// Implement the function
}

void TEST_GREATER_THAN_OR_EQUAL(Vector& vector1, Vector& vector2)
{
// Implement the function
}

void TEST_Subscription(Vector& vector, int i, double newValue)
{
	cout <<"+--------------------+" << endl
		<< "| SUBSCRIPTION TEST |" << endl
		<< "+--------------------+" << endl;

	cout << "Vector itself : " << vector << endl;
	cout << "Get vector[" << i << "] = " << vector[i] << endl;
	vector[i] = newValue;
	cout << "Set vector[" << i << "] to "<<newValue<<", then vector[" << i << "] =
" << vector[i] << endl;
}

void TEST_Addition(Vector& vector1, Vector& vector2)
{
	cout <<"+---------------+" << endl
		<< "| ADDITION TEST |" << endl
		<< "+---------------+" << endl;

	Vector result = vector1 + vector2;

	cout << vector1 << " + " << vector2 << " = " << result << endl;
}
```

```
void TEST_AdditionOver(Vector vector1, Vector vector2)
{
	cout << "+--------------------+" << endl
		<< "| ADDITION OVER TEST |" << endl
		<< "+--------------------+" << endl;
	cout << "Vector 1 Before Addition over: " << vector1 << endl;
	vector1 += vector2;
	cout << "Vector 1 After Addition over: " << vector1 << endl;
}

void TEST_Substraction(Vector& vector1, Vector& vector2)
{
// Implement the function}

void TEST_SubstractionOver(Vector vector1, Vector vector2)
{
// Implement the function
}

void TEST_DotProduct(Vector& vector1, Vector& vector2)
{
// Implement the function}

void TEST_Constant_Multiplication(Vector& vector1,  double constant_value)
{
// Implement the function}

void TEST_Constant_MultiplicationOver(Vector vector, double constant_value)
{
	cout << "+-----------------------------+" << endl
		<< "| CONSTANT MULTIPLICATIN OVER |" << endl
		<< "+-----------------------------+" << endl;
	cout << "Vector 1 Before Constant Multiplication over: " << vector << endl;
	vector *= constant_value;
	cout << "Vector 1 After Constant Multiplication over: " << vector << endl;
}

void TEST_Division(Vector& vector1, Vector& vector2)
{
// Implement the function}

void TEST_DivisionOver(Vector vector1, Vector vector2)
{
// Implement the function}

void TEST_Constant_Division(Vector& vector1, double constant_value)
{
// Implement the function
}

void TEST_Constant_DivisionOver(Vector vector, double constant_value)
{
// Implement the function
}
```

```
void TEST_Magnitude(Vector& vector1)
{
	cout <<"+----------------+" << endl
		<< "| MAGNITUDE TEST |" << endl
		<< "+----------------+" << endl;
	double result = vector1();
	cout << "MAG( " << vector1 << " ) = " << result << endl;
}

void TEST_InverseDirection(Vector vector)
{
	cout << "+-------------------+" << endl
		<< "| INVERSE DIRECTION |" << endl
		<< "+-------------------+" << endl;
	cout << "Original Vector: " << vector << endl;
	cout << "Inversed Vector: " << !vector << endl;
}

int main()
{
	double firstTestData[] {1.2, 2.4, 3.6};
	double secondTestData[] {1.8, 2.6, 3.4};
	// Implement the function
	Vector v1(3);
	Vector v2(firstTestData, 3);
	Vector v3(firstTestData, 3);
	Vector v4(secondTestData, 3);
	TEST_Input(v1);
	TEST_Output(v1);
	TEST_CopyConstructor(v1);
	TEST_Assignment(v1);
	TEST_Equal(v2, v3);
	TEST_Not_Equal(v3, v4);
	TEST_LESS_THAN(v1, v2);
	TEST_LESS_THAN_OR_EQUAL(v2, v3);
	TEST_GREATER_THAN(v1, v2);
	TEST_GREATER_THAN_OR_EQUAL(v2, v3);
	TEST_Subscription(v1, 1, 5.3);
	TEST_Addition(v1, v2);
	TEST_AdditionOver(v1, v2);
	TEST_Substraction(v1, v2);
	TEST_SubstractionOver(v1, v2);
	TEST_DotProduct(v1, v2);
	TEST_Constant_Multiplication(v1, 2);
	TEST_Constant_MultiplicationOver(v1, 2);
	TEST_Division(v1, v2);
	TEST_DivisionOver(v1, v2);
	TEST_Constant_Division(v1, 2);
	TEST_Constant_DivisionOver(v1, 2);
	TEST_Magnitude(v1);
	TEST_InverseDirection(v1);
	return 0;
}
```

Problem-Solving Tips

1- Use given UML Diagram and Test Code Template.