Object-Oriented Programming

COS20007 Test

Semester 1 2019

Duration: 90 minutes

Number of Questions: 5

23½/25

Instructions

- Answer **ALL** the questions
- Answer the questions in the space provided at the end of each question
- Hand in the entire question paper when you have finished
- No books, papers or computer access are allowed during the test.
- Use a pen or pencil to write your answers.

STUDENT ID Number: 100073484

STUDENT NAME: Lee Zong Yang

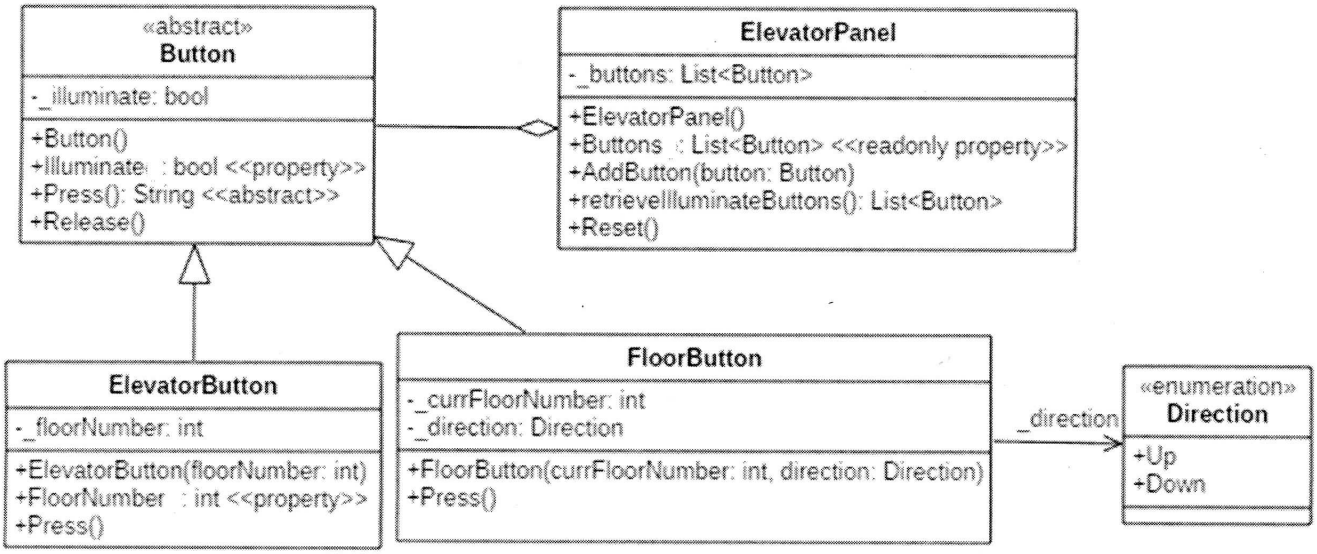
Please take note that:

Points are allocated to determine whether you have passed the test or are granted a resit. You need to obtain 15 out of 25 points to pass the test. The points will be included in your final aggregated marks.

For official use:

☑ Pass ☐ Resit

Section A



The above UML class diagram is an object-oriented design for elevator simulation. An **Elevator Panel** is to be installed in an elevator to control movement between floors upon instructions received. An elevator panel has buttons (**Elevator buttons**) inside the cabin to let the user who got in the elevator to select his/her desired floor. Likewise, each floor has buttons (**Floor buttons**) to summon the elevator to go floor above and floor below respectively. These buttons will illuminate indicating the request is accepted.

Button: Any object that can be pressed.

Each Button will:

- know its illuminate status,
- can be constructed with a default illuminate status of false
- can be pressed (which returns a message indicating what occurred).
- can be released (which sets the illuminate status to false).

This class is implemented as abstract.

Elevator Button: An elevator button is a kind of button which has a _floorNumber field. Its _floorNumber is initialized upon the value passed in when the button is created. When the elevator button is pressed, it will set its illuminate status to true and returns the string indicating the floor number pressed. For instance, if the floor number is 2, it shall return a string "You have pressed floor number: 2".

Floor Button: A floor button is a kind of button which has a _currFloorNumber field and a _direction field. Both of these fields are initialized upon the values passed in when the button is created. When the floor button is pressed and if the direction is moving up, it will set its illuminate status to true and returns the string "You are moving up from floor number: *current floor number*". Otherwise, it returns the string "You are moving down from floor number: *current floor number*".

***current floor number* – you are required to display the current floor number of the Floor Button pressed.

Elevator Panel: An elevator panel contains a collection of buttons. Buttons can be added to the elevator panel. Buttons which illuminate status is true can be retrieved from the collections. The

elevator panel can be reset which will set the illuminate status of all the buttons in the collection to false.

Question 1 – Class Implementation [14 points]

Write the code for all of the classes and enumeration based upon the given UML class diagram and its descriptions accordingly. You must follow naming conventions given and indent your code appropriately.

```
Namespace SectionA
{
    public enum Direction
    {
        Up,
        Down
    }
}

Namespace SectionA
{
    public abstract class Button
    {
        private bool _illuminate;

        public Button()
        {   _illuminate = false;
        }

        public bool Illuminate
        {
            get { return _illuminate; }
            set { _illuminate = value; }
        }

        public abstract String Press();
        public void Release()
        {
            _illuminate = false;
        }
```

```
using System;
Namespace SectionA
{
    public class ElevatorButton : Button
    {
        private int _floorNumber;
        public ElevatorButton(int floorNumber)
        {
            _floorNumber = floorNumber;
        }
        public int FloorNumber
        {
            get { return _floorNumber; }
            set { _floorNumber = value; }
        }
        public override String Press()
        {
            Illuminate = true;
            return "You have pressed floor number: " +
                   _floorNumber;
        }}}
using System;
Namespace SectionA
{
    public class FloorButton : Button
    {
      private int _currFloorNumber;
      private Direction _direction;
```

```
public FloorButton ( int currFloorNumber, Direction direction)
{
    _currFloorNumber = currFloorNumber;
    _direction = Direction;
}
public overriden String Press ()
{
    Illuminate : true;
    if ( _direction == Direction.Up )
      {
        return "You are moving up from floor
                number : " + _currFloorNumber;
      }
    else {
        return "You are moving down from floor
                number : " + _currFloorNumber; }
    }
}
using System;
using System.Collections.Generic;
Namespace SectionA
{
    public class ElevatorPanel
    {
        private List<Button> _buttons;
        public ElevatorPanel ()
        { _buttons = new List<Button>();
        }
```

Question 2 – Unit Test [3 points]

Develop a test class called **ElevatorSimulationTest**, which contains two tests as detailed below.

- Create a unit test called **TestFloorButtonPressed** () to test the Press() method in the FloorButton class:
  - Instantiate a FloorButton object
  - Use "Assert" to check equality on the returned string when FloorButton object is pressed.
- Create a unit test called **TestButtonCollectionCounts()** to the test on the number of buttons added in the ElevatorPanel class:
  - Instantiate a FloorButton object
  - Instantiate an ElevatorButton object
  - Instantiate an ElevatorPanel object and add the two button objects created into the ElevatorPanel object created
  - Finally, use "Assert" to check equality for the number of buttons added.

```
[TestFixture()]
public class ElevatorSimulationTest
{
    [Test()]
    public void TestFloorButtonPressed()
    {
        Button _floorButton = new FloorButton(3, Direction.Up);
        Assert.AreEqual("You are moving up from floor
                         number : 3", _floorButton.Press());
    }
    [Test()]
    public void TestButtonCollectionCounts()
    {
        Button _floorButton2 = new FloorButton(1, Direction.Up);
        Button _elevatorButton = new ElevatorButton(1);
        ElevatorPanel _elevatorPanel = new ElevatorPanel();
        _elevatorPanel.AddButton(_floorButton2);
        _elevatorPanel.AddButton(_elevatorButton);
        Assert.AreEqual(2, _elevatorPanel.Buttons.Count);
    }
```

Question 3 – Main Program [3 points]

After completing the codes for the system described above, write a small main program that creates objects of each of the classes, sets up any collaborations, and calls each of the methods based on the comments given.

```
using System;
using System.Collections.Generic;

namespace SemesterTest
{
  class MainClass
  {
     public static void Main(string[] args)
     {
       // Instantiate an ElevatorButton object and a FloorButton object
          Button _floorButton = new FloorButton ( 1, Direction.Up);
          Button _elevatorButton = new ElevatorButton (1);

       // Press the FloorButton object created and display the
          appropriate message
          Console.WriteLine (_floorButton.Press());

       // Instantiate an ElevatorPanel object and add the two Button objects
          into ElevatorPanel
          ElevatorPanel _elevatorPanel = new ElevatorPanel();
          _elevatorPanel.AddButton (_floorButton);
          _elevatorPanel.AddButton (_elevatorButton);
       // Display the number of illuminate buttons in the ElevatorPanel

          Consol.Writeline (" number : {0}",
                                _elevatorPanel.retrieveIlluminateButtons().Count);

       // Reset the ElevatorPanel

          _elevatorPanel.Reset()


       // Display the number of illuminate buttons in the ElevatorPanel

          Consol.Writeline (" number : {0}",

                                _elevatorPanel.retrieveIlluminateButtons().Count);

          Console.ReadLine();
     }
  }
}
```

Section B

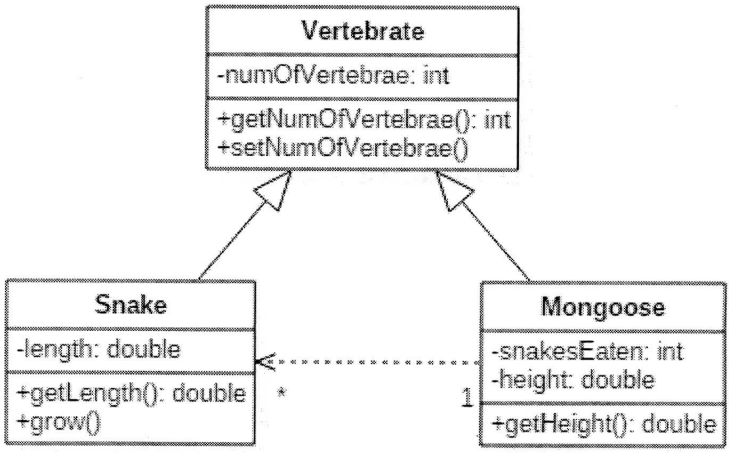
Question 4 – Encapsulation [2 points]

Define encapsulation. Explain how encapsulation relates to object-oriented programming. Relate your answer to the relevant parts of the code written in Section A.

Encapsulation is a process to hide certain internal data and method from access of other class. Encapsulation is important in object-oriented programming to reduce the chance of writing buggy code and protect the privacy of important data from anonymous access. In the case of ElevatorPanel, _buttons is set as private which forbidden other class from acessing it. While its properties are set as public but without setter. Thus, other class could only gain the access to the list of button but doesn't have the privilege to modify it without using other method. (1) (1)

Section C

Question 5 – Reasoning [3 points]



(a) What is the relationship between Snake and Vertebrate?

Inheritance (1)

(b) What is the association type between Mongoose and Snake?

one to many realization X

(c) According to the class diagram, can 2 mongooses eat 1 snake? Justify your answer.

No, because the relationship between snake and mongooses is one to many. (1)

== END OF QUESTIONS PAPER ==

Question 1 : Lee Zong Yang 10007348K part of Elevator Panel Class.

```
public List<Button> Buttons
{
    get { return _buttons; }
}

public void AddButton(Button button)
{
    _buttons.Add(button);
}

public List<Button> retrieveIlluminateButtons()
{   public List<Button> _illuminateButtons = new List<Button>();
    foreach (Button button in _buttons) {
        if (button.Illuminate == true)
        {
            _illuminateButtons.Add(button);
        }
    }
    return _illuminateButtons;
}

public void Reset()
{
    foreach (Button button in _buttons)
    {
        button.Illuminate = false;
    }
}
}
```

1/2 1/2 1/2 1

Question 1

```
using System;
namespace OOP_TestFix
{
    public enum Direction
    {
        Up,
        Down
    }
}
```

```
using System;
namespace OOP_TestFix
{
    public abstract class Button
    {
        private bool _illuminate;

        public Button()
        {
            _illuminate = false;
        }

        public bool Illuminate
        {
            get { return _illuminate; }
            set { _illuminate = value; }
        }

        public abstract String Press();

        public void Release()
        {
            _illuminate = false;
        }
    }
}
```

```
using System;
namespace OOP_TestFix
{
    public class ElevatorButton: Button
    {
        private int _floorNumber;

        public ElevatorButton(int floorNumber)
        {
```

```
            _floorNumber = floorNumber;
        }

        public int FloorNumber
        {
            get { return _floorNumber; }
            set { _floorNumber = value; }
        }

        public override String Press()
        {
            Illuminate = true;
            return "You have pressed floor number :" + _floorNumber;
        }

    }
}


using System;
namespace OOP_TestFix
{
    public class FloorButton: Button
    {
        private int _currFloorNumber;
        private Direction _direction;

        public FloorButton(int currFloorNumber, Direction direction)
        {
            _currFloorNumber = currFloorNumber;
            _direction = direction;
        }

        public override string Press()
        {
            Illuminate = true;
            if (_direction == Direction.Up)
            {
                return "You are moving up from floor number: " + _cu
rrFloorNumber;
            }
            else
            {
                return "You are moving down from floor number: " + _
currFloorNumber;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace OOP_TestFix
{
    public class ElevatorPanel
    {
        private List<Button> _buttons;

        public ElevatorPanel()
        {
            _buttons = new List<Button>();
        }

        public List<Button> Buttons
        {
            get { return _buttons; }
        }

        public void AddButton(Button button)
        {
            _buttons.Add(button);
        }

        public List<Button> retrieveIlluminateButtons()
        {
            List<Button> _illuminateButtons = new List<Button>();
            foreach (Button button in _buttons)
            {
                if (button.Illuminate == true)
                {
                    _illuminateButtons.Add(button);
                }
            }
            return _illuminateButtons;
        }

        public void Reset()
        {
            foreach (Button button in _buttons)
            {
                button.Illuminate = false;
            }
        }
    }
}
```

Question 2

```
using System;
using NUnit.Framework;

namespace OOP_TestFix
{
    [TestFixture()]
    public class ElevatorSimulationTest
    {
        [Test()]
        public void TestFloorButtonPressed()
        {
            Button _floorButton = new FloorButton(3, Direction.Up);
            Assert.AreEqual("You are moving up from floor number : 3
", _floorButton.Press());
        }

        [Test()]
        public void TestButtonCollectionCounts()
        {
            Button _floorButton2 = new FloorButton(1, Direction.Up);
            Button _elevatorButton = new ElevatorButton(1);
            ElevatorPanel _elevatorPanel = new ElevatorPanel();
            _elevatorPanel.AddButton(_floorButton2);
            _elevatorPanel.AddButton(_elevatorButton);
            Assert.AreEqual(2, _elevatorPanel.Buttons.Count);
        }
    }
}
```

Question 3

```
using System;

namespace OOP_TestFix
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Button _floorButton = new FloorButton(1, Direction.Up);
            Button _elevatorButton = new ElevatorButton(1);

            Console.WriteLine(_floorButton.Press());

            ElevatorPanel _elevatorPanel = new ElevatorPanel();
            _elevatorPanel.AddButton(_floorButton);
            _elevatorPanel.AddButton(_elevatorButton);
```

```
            Console.WriteLine("number :{0}", _elevatorPanel.retrieve
IlluminateButtons().Count);

            _elevatorPanel.Reset();

            Console.WriteLine("number :{0}", _elevatorPanel.retrieve
IlluminateButtons().Count);

            Console.ReadLine();
        }
    }
}
```

Section C

b)What is the association type between Mongoose and Snake?

dependency