Object-Oriented Programming

COS20007 Test

Semester 2 2019

Duration: 90 minutes

$\frac{20\frac{1}{2}}{25}$

Number of Questions: 6

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: 101533222

STUDENT NAME: LEWIS BROCKMAN-HORSLEY

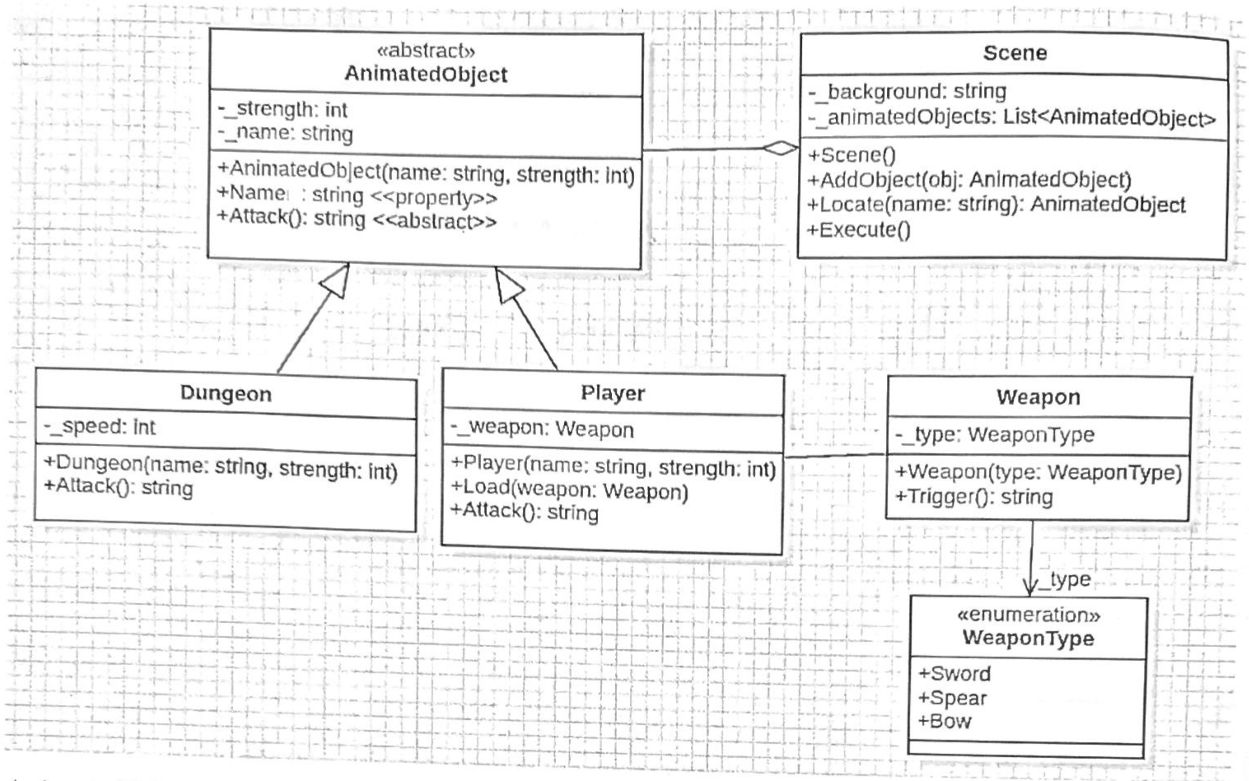
Please take note that:

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



«abstract»
AnimatedObject
-_strength: int
-_name: string
+AnimatedObject(name: string, strength: int)
+Name : string <<property>>
+Attack(): string <<abstract>>
Scene
-_background: string
-_animatedObjects: List<AnimatedObject>
+Scene()
+AddObject(obj: AnimatedObject)
+Locate(name: string): AnimatedObject
+Execute()
Dungeon
-_speed: int
+Dungeon(name: string, strength: int)
+Attack(): string
Player
-_weapon: Weapon
+Player(name: string, strength: int)
+Load(weapon: Weapon)
+Attack(): string
Weapon
-_type: WeaponType
+Weapon(type: WeaponType)
+Trigger(): string
_type
«enumeration»
WeaponType
+Sword
+Spear
+Bow

AnimatedObject: An animated object has a name and strength level. This class is implemented as abstract.

The animated object has a constructor, which takes two parameters (name and strength) and assigned to its field. The name field can be modified and queried using their respective property.

The animated object can attack their enemies accordingly. This method is implemented as abstract.

Player: is a kind of AnimatedObject and has an additional field, weapon.

Player has a constructor, which takes two parameters (name and strength)

- The constructor will call the base constructor and pass the name and strength as arguments,
- The default weapon is null.

The weapon field can be loaded; the loaded weapon object is stored in the Player's weapon field.

Player has an override method, Attack which will trigger the weapon field and return the relevant string value.

Dungeon: is a kind of AnimatedObject and has an additional field, speed.

Dungeon has a constructor, which takes two parameters (name and strength)

- The constructor will call the base constructor and pass the name and strength as arguments,
- The default speed is 10.

Dungeon has an override method, Attack, which will reduce the speed by 1 and return the string "Attack with speed level: *current speed level*"

** *current speed level* - you are required to display the current speed level after the decrement.

Weapon: A weapon has a type field.

Weapon has a constructor which takes a parameter type field and initialize the field accordingly.

Weapon can be triggered. If the weapon type is a Sword, it will return the string "Swords drawn!". If the weapon type is a Spear, it will return the string "Beware of its sharpness!". If the weapon type is a Bow, it will return the string "Ready to release!".

Scene: A scene contains a background field and a collection of animated objects.

Scene has a default constructor which perform the following:

- Initialize the background field to a default string value "Battlefield"
- Instantiate a List of AnimatedObject

Animated object can be added to the Scene at any time.

Scene can also locate or find an animated object from its collection of animated objects through the name field.

Scene can also be executed which will display its background field and having its collection of animated objects to trigger their attacks.

Question 1 – Class Implementation [14 points]

Write the **C# 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. using System;

```
public enum WeaponType {
      Sword,
      Spear,
      Bow                      ✓ ½
}
public class Weapon {            ✓ ½
   private WeaponType _type;          ✓ ½
   Public Weapon (WeaponType type){
       _type = type;               ✓ ½
   }
   public string Trigger () {
      switch (_type){
         case (Sword):
              return "Swords drawn!";
         case (Spear):
              return "Beware of its Sharpness!";      ½
         case (Bow):
              return "Ready to release!";
      }
   return ""; }

} //end of class
```

```
public class Player : AnimatedObject {  ✓ ½
   private Weapon _weapon;  ✓ ½
   public Player (String name, int strength) :
   base (name, strength) {
   _weapon = null;  ✓ ½
   }
   public void Load (Weapon weapon){
      _weapon = weapon;  ✓ ½
   }
   public override string Attack() {
      return _weapon.Trigger();  ✓ ½
   }
} //end of Player Class
public class Dungeon : AnimatedObject {  ✓ ½
  private int _speed;  ✓ ½
  public Dungeon (string name, int strength):
  base(name, strength){
     _speed = 10;  ✓ ½
  }
  public override string Attack () {
    _speed--;
    return "Attack with speed level:" + _speed;  ✓ ½
  }
} //end of Dungeon
```

```
public abstract class AnimatedObject {   ✓ ½
   private int _strength;
   private string _name;   ✓ ½
   public AnimatedObject (string name, int strength) {
      _name = name;
      _strength = strength;   ✓ ½
   }
   public string Name {
         get { return _name;}
         set {_name = value;}   ✓ ½
   }
   public^ string Attack();
}
using System.Collections.Generic;
public class Scene {   ✓ ½
   private string _background;   ✓ ½
   private List<AnimatedObject> _animatedObjects;   ✓
   public Scene () {
      _background = "Battlefield";   ✓ ½
      _animatedObjects = new List<AnimatedObject>();   ✓
   }
   public void AddObject (AnimatedObject obj) {   ✓ ½
      _animatedObjects.Add(obj);
   }
   public AnimatedObject Locate (string name) {   ✓
      return _animatedObjects.Find(o => O.Name == name);   ✓ 1
   }
   public void Execute () {   ✓
      Console.WriteLine(_background);   ✓
      foreach (AnimatedObject a in _animatedObjects){
         a.Attack();
      }^   ½
   }
} //end of Scene
```

Question 2 – C# Unit Test [3 points]

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

- Create a unit test called **TestPlayerAttack ()** to test the Attack() method in the Player class:
  - Instantiate a Weapon object
  - Instantiate a Player object and load the Weapon objects created to the Player object
  - Finally, use "Assert" to check equality for the string value returned after triggering the Player's Attack method.
- Create a unit test called **TestDungeonName()** to test on the Dungeon name initialized in the AnimatedObject base class:
  - Instantiate a Dungeon object
  - Finally, use "Assert" to check equality on the name assigned to the Dungeon object created.

```
using NUnit.Framework;                                        ½
[TestFixture]
public class GameSceneTest {
   [Test]
   public void TestPlayerAttack() {
   Weapon weapon = new Weapon(Weapontype.Sword);
   Player player = new Player("Lewis", 10);
   player.Load(weapon);

   Assert.AreEqual("Swords drawn!", player.Attack(),          1
      "Player has correct attack");
   }
   [Test]
   public void TestDungeonName() {
   Dungeon dungeon = new Dungeon("Room", 5);
   Assert.AreEqual("Room", Dungeon.Name,
      "Dungeon Name is correct");                             1
   }
} // end of class
```

Question 3 – C# 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;

namespace SemesterTest
{
  class MainClass
  {
     public static void Main(string[] args)
     {
        // Instantiate a Weapon object of type Bow and a Dungeon object
        Weapon weapon = new Weapon(WeaponType.Bow); ✓
        Dungeon dungeon = new Dungeon("Room", 5); ✓        ½

        // Instantiate a Player object and have the player object to load
           the weapon object created
        Player player = new Player("Lewis", 10); ✓
        player.Load(weapon); ✓        ½

        // Instantiate a Scene object and add the Player and Dungeon objects
           created to it
        Scene scene = new Scene(); ✓
        scene.AddObject(player); ✓
        scene.AddObject(dungeon); ✓        ½

        // Locate the Dungeon object through the Scene object created and
           assign it to an AnimatedObject    .
        AnimatedObject obj = Scene.Locate("Room"); ✓        1

        // Execute the Scene object created
        scene.Execute(); ✓        ½

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

Section B

Question 4 [2 points]

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

When a child Object inherits from a parent it allows the child object to have all the values and actions of the parent that have either public or protected access modifiers. All child objects share the traits of the parent class. Represents an "Is A" relationship. ✓ 1
In Section A the Player and Dungeon classes inherit from the AnimatedObject class. ✓ 1

Section C

Question 5 [2 points]

Describe and explain the use of principle of polymorphism in object-oriented design. Identify a piece of work that you have completed for this unit and explain how it demonstrates the principle.

Static Polymorphism ✓
method overloading is used to call the same method name with different parameters passed.
For example Add(int a, int b) and Add(int a, int b, int c) ✗

Dynamic Polymorphism ✓
child classes must implement methods that override a method signature of a parents class,
For example the Attack method in Section A. ✗

½

Question 6 [1 point]

In UML class diagram, what is the notation used to represent dependency relationship?

$$\begin{tikzcd}\text{Weapon} \arrow[d, "\text{- type}"] \\ \text{WeaponType}\end{tikzcd}$$

═ END OF QUESTIONS PAPER ═