

Name:

Midterm Exam
CMPS 20 – Game Design Experience
11 questions, 100 points, worth 15% of your total grade

1. (5 points) The following C# code creates a list, and fills it with a random number (between 0 and 19) of random integers (between 0 and 99).

Into this existing code, at the location indicated in the comments, write a loop that displays the members of this list, separated by the “|” character, to the console. Your answer *must use* the *foreach* statement. Recall the syntax of *foreach* is `foreach ({type} {identifier} in {identifier})`

```
static void Main(string[] args)
{
    List<int> myList = new List<int>();
    Random random = new Random();
    int len = random.Next(20);
    int i;

    System.Console.WriteLine("List has {0} entries", len);

    for (i = 0; i < len; i++)
    {
        myList.Add(random.Next(100));
    }

    // Add your code here to display list elements
    // Your code must use the foreach statement

    foreach (int e in myList) {
        System.Console.Write("{0} | ", e);
    }

}
```

Name:

2. (a) (5 points) To the following C# code, add new code that will display the X and Y position of all elements in myList. In your answer, you must use the *foreach* statement.

(b) (5 points) Write the output of this program.

```
namespace InterfaceForeachQuestion
{
    interface IGridPosition
    {
        int X { get; set; }
        int Y { get; set; }
    }

    class Player : IGridPosition
    {
        public int X { get; set; }
        public int Y { get; set; }
        private int lives;

        public Player(int x, int y)
        {
            X = x;
            Y = y;
            lives = 5;
        }
    }

    class Goomba : IGridPosition
    {
        public int X { get; set; }
        public int Y { get; set; }
        private int hitpoints;

        public Goomba(int initial_hp, int x, int y)
        {
            X = x;
            Y = y;
            hitpoints = initial_hp;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<IGridPosition> myList = new List<IGridPosition>();
            myList.Add(new Player(100, 250));

            for (int i = 0; i < 3; i++)
                myList.Add(new Goomba(i*25, i*10, i*10));

            // Add your code here to display the X, Y position of each element of myList

```

```
(a) foreach (IGridPosition ig in myList)
    System.Console.WriteLine("{0},{1}", ig.X, ig.Y);
    }

```

```
(b) (100,250)
    (0,0)
    (10,10)
    (20,20)
```

Name:

3. (10 points) Write the output of the following program.

```
delegate void Del(string s);

class TestClass
{
    static void Hello(string s)
    {
        System.Console.WriteLine(" Hello, {0}!", s);
    }

    static void Goodbye(string s)
    {
        System.Console.WriteLine(" Goodbye, {0}!", s);
    }

    static void Main()
    {
        Del a, b, c, d;

        a = Hello;
        b = Goodbye;
        c = a + b;
        d = c - a + b;

        a("A");
        System.Console.WriteLine("----");
        b("B");
        System.Console.WriteLine("----");
        c("C");
        System.Console.WriteLine("----");
        d("D");
    }
}
```

Program output:

```
    Hello, A!
----
    Goodbye, B!
----
    Hello, C!
    Goodbye, C!
----
    Goodbye, D!
    Goodbye, D!
```

4. (a) (5 points) In XNA, which two methods in your subclass of `Microsoft.Xna.Framework.Game` are called every clock tick?

Update() and Draw() are called every clock tick.

(b) (5 points) Using default settings, how many times a second are these methods called?

By default, XNA updates the display 60 times a second, so Update() and Draw() are each called 60 times a second, once per clock tick.

Name:

5. (5 points) What is the output of the following program?

```
interface IOffset
{
    int offset(int x);
}

public class inc : IOffset
{
    int delta;

    public inc(int delta)
    {
        this.delta = delta;
    }

    public int offset(int x)
    {
        return x + delta;
    }
}

public class dec : IOffset
{
    int delta;

    public dec(int delta)
    {
        this.delta = delta;
    }

    public int offset(int x)
    {
        return x - delta;
    }
}

public class mul : IOffset
{
    int multiplier;

    public mul(int multiplier)
    {
        this.multiplier =
            multiplier;
    }

    public int offset(int x)
    {
        return x * multiplier;
    }
}

public delegate int intop(int x);

class Program
{
    static void Main(string[] args)
    {
        intop delta;
        int number = 0;

        dec D = new dec(5);
        inc I = new inc(10);
        mul M = new mul(10);

        delta = I.offset;
        System.Console.WriteLine("Orig: {0}", number);

        number = delta(number);
        System.Console.WriteLine("1: {0}", number);

        delta = D.offset;
        number = delta(number);
        System.Console.WriteLine("2: {0}", number);

        delta = M.offset;
        number = delta(number);
        System.Console.WriteLine("3: {0}", number);
    }
}
```

Output of this program:

**Orig: 0
1: 10
2: 5
3: 50**

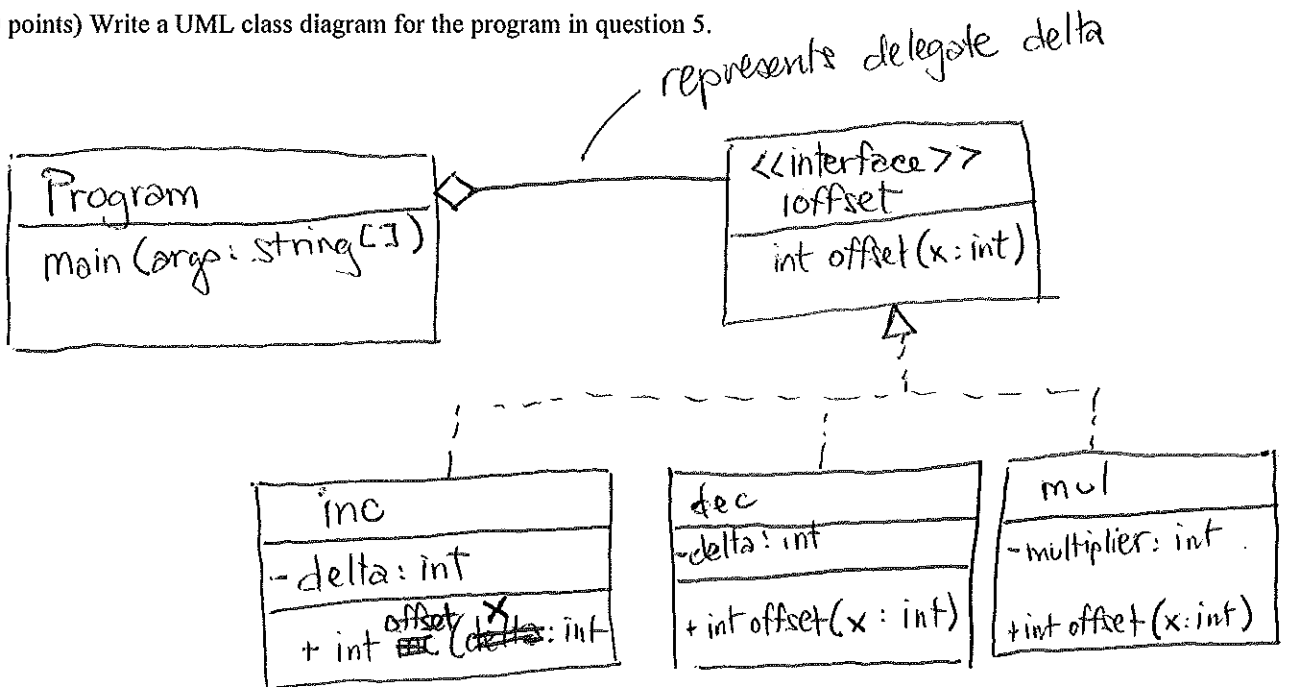
6. (10 points) The program in question 5 implements a well known software design pattern. (4 out of 10 points) Which one? (6 out of 10 points) Briefly explain why the program exemplifies this design pattern.

Strategy pattern.

A client (method Main in Class Program) composes an algorithm (method offset()) from one of the subclasses of IOffset. IOffset and its implementing classes act as the strategy class hierarchy. Instead of a reference to a class instance (which is used to access the strategy algorithm), this implementation of strategy uses a delegate to hold a reference to a specific algorithm method. The underlying idea is still the same: use composition to flexibly attach an algorithm to a client.

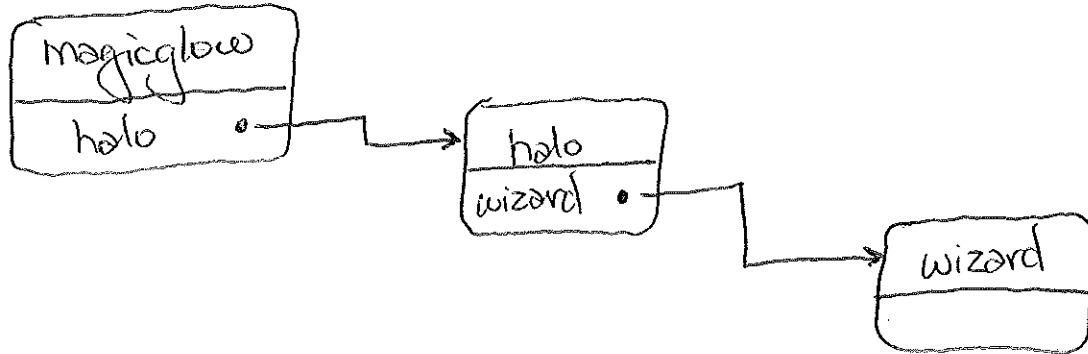
Name:

7. (10 points) Write a UML class diagram for the program in question 5.



Name:

8. (10 points) An RPG game has a wizard character that, at certain times during the game, can have a halo over her head, a magic glow surrounding her body, or both. The implementation of this game uses the Decorator design pattern. Draw a diagram showing just the object instances, and the objects they reference, for a Decorator chain when the wizard has a halo and a magic glow. Note that this question is not asking for a UML diagram (which concerns classes), but instead wants to see a diagram of class instances, and their connections.



9. (a) (5 points) Explain the meaning of the design principle, “favor composition over inheritance.”

Favor composition over inheritance means, when faced with a situation where a class needs to have multiple behaviors, it is better to create a second class holding the various behaviors, and then compose instances of that behavior class with the original class. The strategy pattern exemplifies this.

The alternative is to use inheritance to create multiple subclasses of the original class, one for each new feature.

(b) (5 points) What is the benefit of favoring composition over inheritance? That is, why is this a good thing?

Some benefits of composition:

Behaviors can be changed while a program is running (dynamic behavior modification)

Behaviors can be used by multiple classes

Avoids combinatoric expansion of behaviors when you need combined behaviors.

Use of inheritance introduces problems:

Cannot change behavior while program is running, unless object instance is destroyed, and replaced with another object of a different type.

Behaviors are locked-in to a specific subclass.

May need more classes to compose behavior.

Name:

10. (10 points) What is the console output of the following program, if the input is the Konami Code, followed by the "Back" button, entered into an Xbox 360 controller? For this assignment, the arrows correspond to DPad directions (e.g., up arrow is the DPad pushed up), etc. The Konami Code is:



Followed by BACK.

Program Output: *Final: 220*

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    int total = 0;
    bool up_pressed = false;
    bool right_pressed = false;
    bool b_pressed = false;

    public Game1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
    }

    protected override void Initialize()
    {
        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
    }

    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }

    protected override void Update(GameTime gameTime)
    {
        // Allows the game to exit
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        {
            System.Console.WriteLine("Final: {0}", total);
            this.Exit();
        }

        if (GamePad.GetState(PlayerIndex.One).DPad.Up == ButtonState.Pressed)
            up_pressed = true;

        if (up_pressed && GamePad.GetState(PlayerIndex.One).DPad.Up == ButtonState.Released) {
            up_pressed = false;
            total++;
        }

        if (GamePad.GetState(PlayerIndex.One).DPad.Right == ButtonState.Pressed)
            right_pressed = true;

        if (right_pressed && GamePad.GetState(PlayerIndex.One).DPad.Right == ButtonState.Released) {
            right_pressed = false;
            total += 10;
        }

        if (GamePad.GetState(PlayerIndex.One).Buttons.B == ButtonState.Pressed)
            b_pressed = true;

        if (b_pressed && GamePad.GetState(PlayerIndex.One).Buttons.B == ButtonState.Released) {
            b_pressed = false;
            total *= 10;
        }

        base.Update(gameTime);
    } // code continues in box above and to the right.
}
```

```
protected override void Draw(GameTime
gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

Name:

11. (10 points) Describe two differences between C# interfaces and abstract classes.

A C# interface can inherit from other interfaces. An abstract class cannot inherit from another abstract class.

A C# interface can provide no implementation of methods or properties. An abstract class may implement some (or all) of its methods and properties.

Extra Credit (5 points). What is the output of the following program?

```
class Program
{
    public delegate int decrement(int dec);

    static public decrement myDecrement;

    static public int decrementer(int dec)
    {
        System.Console.WriteLine("{0}|", dec);
        if (dec > 0)
        {
            return myDecrement(dec - 1);
        }
        else
        {
            return 0;
        }
    }

    static public int writeBlank(int lines)
    {
        System.Console.WriteLine();
        return 0;
    }

    static void Main(string[] args)
    {
        myDecrement += decrementer;
        myDecrement += writeBlank;
        myDecrement += decrementer;

        myDecrement(2);
    }
}
```

```
2|1|0|
0|
1|0|
0|
2|1|0|
0|
1|0|
0|
```