SE2250 – Software Construction

Lecture #5

C# and Unity – part 2

February 4, 2019

Outline

- C# in Unity – functions
- C# in Unity – object oriented language
  - Objects and classes
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Composition
  - Design patterns - singleton and delegate

C# in Unity

- Question 1
- Which one of the following statements is correct about static functions?

A. Static functions are invoked using objects of a class.
B. Static functions can access static data as well as instance data.
C. Static functions are outside the class scope.
D. Static functions are invoked using the class.

C# in Unity

- Question 2
- Which one of the following statements is NOT correct about instance variables?

A. They are accessed using the class
B. Instance variables are referred to as fields
C. Each instance can have different value
D. They are tied directly to a single instance of the class
E. They are accessed using the instance

C# in Unity

- Question 3
- What will this code print?

A. 3 and 3
B. 2 and 3
C. 2 3
D. 2 and 2
E. 1 and 3

```
public class Main2 : MonoBehaviour {
    void Start() {
        MyClass m1 = new MyClass ();
        m1.increment ();
        MyClass m2 = new MyClass ();
        m2.increment ();
        m2.increment ();
        print (m2.x+ " and " + MyClass.y);
    }
}
public class MyClass {
    public int x = 0;
    public static int y = 0;
    public void increment() {
        x++;
        y++;
    }
}
```

C# in Unity

- Question 4
- Which of the following adds 10 to every value in a 16-element integer array named points?

A. `for (int sub = 0; sub > 15; ++sub) points[sub] += 10;`
B. `foreach(int sub in points) points += 10;`
C. both of these
D. neither of these

C# in Unity - Functions

- Functions
- A function - a chunk of code that does something
  - Built in functions `Start(), Update()`
  - Custom functions

```
void Update() {
        numTimesCalled++;
        PrintUpdates();
}
void PrintUpdates() {
        string outputMessage = "Updates: "+numTimesCalled;
        print( outputMessage );
}
```

C# in Unity - Functions

- Functions
  - Arguments and return values
  - Can return `void`

```
void Awake() {
    int num = Add( 2, 5 );
    print( num );
}

int Add( int numA, int numB ) {
    int sum = numA + numB;
    return( sum );
}
```

Why use functions?

C# in Unity - Functions

- Function Overloading
  - Functions with the same name, but different in the type and number of arguments

…

```
print( Add( 1.0f, 2.5f ));
```

```
float Add( float f0, float f1 ) {return( f0 + f1 );}
```

C# in Unity - Functions

- Optional arguments

```
    …
    SetX( this.gameObject, 25 );
    SetX( this.gameObject );
    …
void SetX( GameObject go, float eX=0.0f ) {
    Vector3 tempPos = go.transform.position;
    tempPos.x = eX;
    go.transform.position = tempPos;
}
```

C# in Unity - Functions

- **`params`** keyword
- Allows functions to accept any number of parameters os the same type

```
…
print(Add(1, 2));
print(Add(1, 2, 3));
print(Add(1, 2, 3, 4, 5));
…
int Add (params int[] ints){
    int sum = 0;
    foreach (int i in ints){
       sum += i;
    }
        return (sum);
}
```

C# in Unity - Functions

- Recursive Functions
  - A function designed to call itself repeatedly
  - 5! = 5 * 4 * 3 * 2 * 1 = 120

```
print( Fac(5) );
…
```

Other way to do this?

C# in Unity – object oriented language

- Object-oriented
  - Objects and classes
  - Inheritance
  - Encapsulation
  - Polymorphism
  - Composition

C# in Unity – object oriented language

- Objects and classes
  - GameObject – base class for all entities in Unity scenes
  - GameObject in the scene (car, wall…) has script(s) attached to define its behaviour
  - MonoBehaviour is the base class from which every Unity script derives.
  - Prefabs are blueprints for creating instances
    - House prefab
    - Different houses in the scene (instances)

C# in Unity – object oriented language

- Classes
  - Class declaration
  - Fields
  - Methods
  - Properties

```
5  public class Enemy : MonoBehaviour {
6
7      public float            speed = 10f;     // The speed in m/s
8      public float            fireRate = 0.3f; // Shots/second (Unused)
9
10     // Update is called once per frame
11     void Update() {
12         Move();
13     }
14
15     public virtual void Move() {
16         Vector3 tempPos = pos;
17         tempPos.y -= speed * Time.deltaTime;
18         pos = tempPos;
19     }
20
21     void OnCollisionEnter( Collision coll ) {
22         GameObject other = coll.gameObject;
23         switch (other.tag) {
24         case "Hero":
25             // Currently not implemented, but this would destroy the hero
26             break;
27         case "HeroLaser":
28             // Destroy this Enemy
29             Destroy(this.gameObject);
30             break;
31         }
32     }
33
34     // This is a Property: A method that acts like a field
35     public Vector3 pos {
36         get {
37             return( this.transform.position );
38         }
39         set {
40             this.transform.position = value;
41         }
42     }
43 }
```

The Class declaration

Fields: Variables local to this class

Methods: Functions local to this class

Note that Move() is a *virtual* function

A Property: A method masquerading as a field through get & set accessors

C# in Unity – object oriented language

- **Properties**: Methods that work like fields
  - A flexible mechanism to read, write, or compute the value of a private field
  - Enable a class to expose a public way of getting and setting values, while hiding implementation
  - Enables data to be accessed easily and still helps promote the safety and flexibility of methods

C# in Unity – object oriented language

- Properties

```
5  public class CountIt : MonoBehaviour {
6      private int         _num = 0;
7
8      void Update() {
9          print( nextNum );
10     }
11     public int nextNum {
12         get {
13             _num++;
14             return( _num );
15         }
16     }
17     public int currentNum {
18         get { return( _num ); }
19         set { _num = value;  }
20     }
21 }
```

C# in Unity – object oriented language

- **Properties**: Methods that work like fields
  - Used as if they are public data members, but they are actually special methods called accessors (get, set)
  - *Value* keyword defines the value being assigned by the set
  - Can be read, write, or both

C# in Unity – object oriented language

- MonoBehaviour
  - The base class from which every Unity script derives
  - Important functions in MonoBehaviour:
    - Update()
      - Called every frame
      - Update intervals vary
      - For non-physics objects
      - Receiving inputs
    - FixedUpdate() - every fixed framerate frame
      - Called every physics step
      - Intervals are consistent
      - Adjusting physics objects
    - LateUpdate() – after all updates

C# in Unity – object oriented language

- MonoBehaviour
  - Important functions in MonoBehaviour:
    - Start() - just before any of the Update methods is called the first time
    - Awake() - when the script instance is being loaded, before Start()
    - OnCollisionEnter ()
    - OnCollisionExit ()
    - OnMouseDown ()
    - OnMouseEnter ()
    - …

C# in Unity – object oriented language

- Inheritance
  - An object acquires all the properties and behaviours of the parent object
  - The class can inherit from only one class
  - The class can implement any number of interfaces

C# in Unity – object oriented language

- Inheritance

```
public class Enemy : MonoBehaviour {
    public float    speed = 10f;   // The speed in m/s
    public float    fireRate = 0.3f; // Shots/second
    void Update() {
        Move();
    }
    public virtual void Move() {… }
    void OnCollisionEnter( Collision coll ) {…}
    public Vector3 pos {…}
}
```

C# in Unity – object oriented language

- Inheritance
- Child acquires all the properties and behaviours of the parent

```
public class EnemyZig : Enemy {
    public override void Move () {
        Vector3 tempPos = pos;
        tempPos.x = Mathf.Sin(Time.time * Mathf.PI*2) * 4;
        pos = tempPos;
        base.Move();
    }
}
```

- A method must be declared as virtual in the superclass for C# to allow it to be overridden in a subclass.

C# in Unity – object oriented language

- Encapsulation
  - Bind together the data and functions that manipulate the data
  - External code is not concerned with the internal workings of an object
  - Unity - A GameObject should be responsible for its actions

C# in Unity – object oriented language

- Polymorphism
  - a single interface to entities of different types
  - https://unity3d.com/learn/tutorials/topics/scripting/polymorphism

  - Demo – fruit

C# in Unity – object oriented language

- Inheritance - **Interfaces**
  - The class can implement any number of interfaces.
  - An interface contains definitions for a group of related functionalities that a class can implement.
  - An interface is a contract between itself and any class that implements it. This contract states that any class that implements the interface will implement the interface's properties, methods and/or events.

C# in Unity – object oriented language

- Inheritance - **Interfaces**
  - Interfaces contain no implementation of methods (only signatures).
  - An interface can't be instantiated directly. Its members are implemented by any class or struct that implements the interface.
  - Interfaces can contain events, indexers, methods, and properties.
  - A class can implement multiple interfaces. A class can inherit from a base class and also implement one or more interfaces.
  - Naming - IMyName

C# in Unity – object oriented language

- Composition
  - Objects can contain other objects in their instance variables
- In Unity
  - Composition may not require an instance variable
  - GameObject is composed of Transform, Rigidbody, Collider…
  - GameObject can contain other game objects (create empty GaimObject and add component objects)

C# in Unity – object oriented language

- Composition
  - Cube Spawner example

```
public class CubeSpawner : MonoBehaviour {

    public GameObject         cubePrefabVar;
    public List<GameObject>   gameObjectList;
    public float              scalingFactor = 0.95f;
    public int                numCubes = 0;


    void Start() {
        gameObjectList = new List<GameObject>();
    }

    void Update () {
```

Design pattern - Singleton

- There will only ever be a single instance of a given class
- A static variable of that class type

```
public class Hero : MonoBehaviour {
    static public Hero S;
    void Awake() {
        if (S == null) {
         S = this;
        } else {
            Debug.LogError("The singleton S of Hero already exists!");
        }
    }
}
```

Design pattern - Singleton

- Access from anywhere

```
void Update() {




}
```

Design pattern - Delegate

- Function delegate
  - A container for similar functions (or methods) that can all be called at once
  - Enable a single call to the fireDelegate() delegate to fire all weapons attached to a player's ship

- Define the delegate type

```
public delegate float FloatOpDelegate( float f0, float f1 );
```

Design pattern - Delegate

- Function delegates
  - Create functions with the same parameters and return types

```
public float FloatAdd(float f0,float f1 ) {
   float result = f0+f1;
   print("The sum of "+f0+" & "+f1+" is "+result+".");
   return( result );
}
public float FloatMultiply(float f0,float f1 ) {
   float result = f0 * f1;
   print("The product of "+f0+" & "+f1+" is "+result+".");
   return( result );
}
```

Design pattern - Delegate

- Can be multicast (more then one method)

```
public FloatOpDelegate fod;
…
fod = FloatAdd;
fod += FloatMultiply;
if (fod != null) {
   float result = fod(3, 4);
   print( result );
}
```

What is printed?

Race conditions

- **Race conditions**
  - A race condition is the behavior of a software system where the output is dependent on the sequence or timing of other uncontrollable events
  - Example
    - Events A and B
    - Sometimes A happens before B and sometimes the other way around
    - Cannot put code in B that relies on A
  - Demo
  - Unity: Edit/Project Settings/Script Execution Order