INFT3960 – Game Production

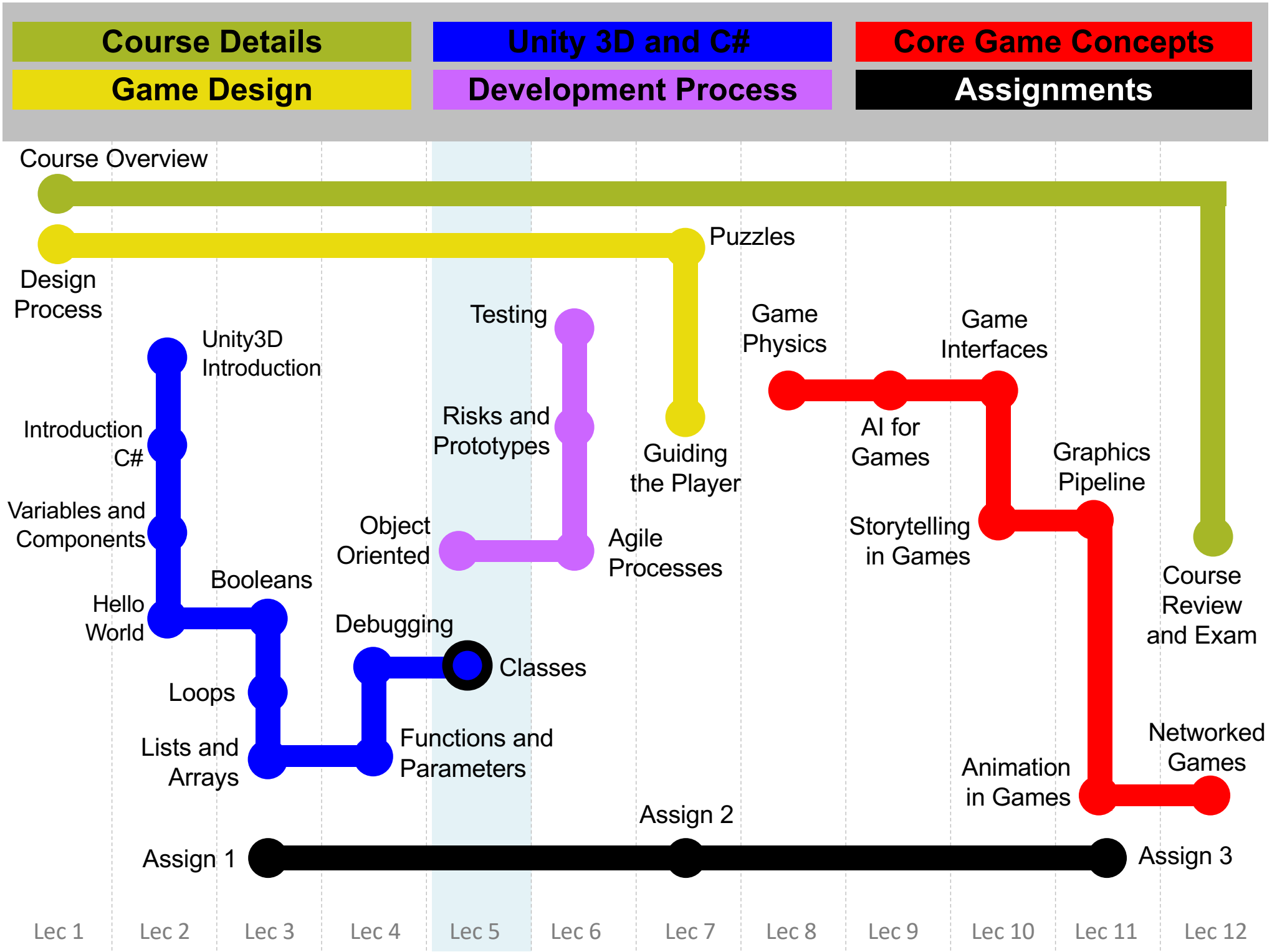
Week 04

Module 5.1

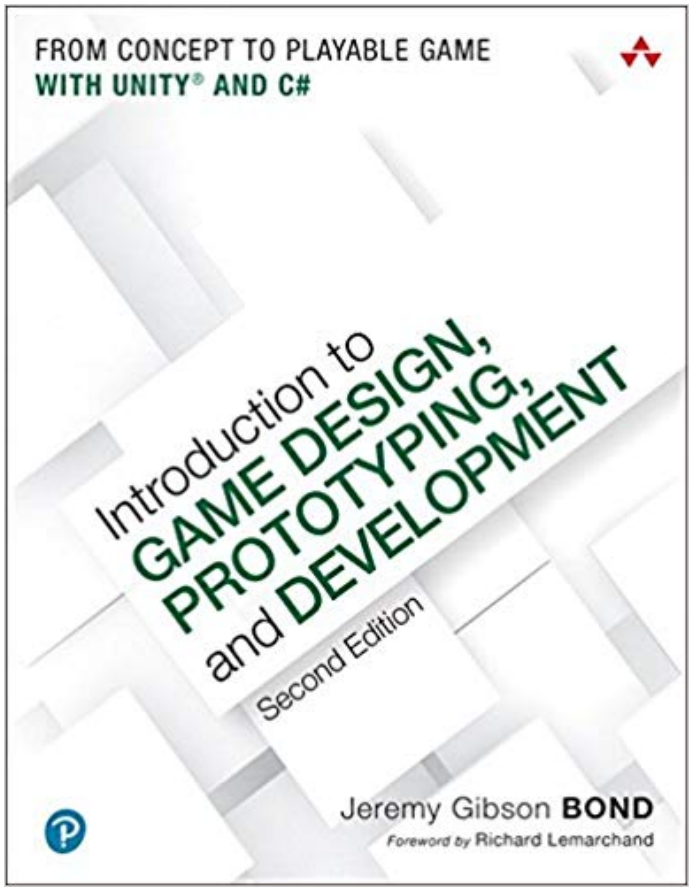
Classes

Course Overview

Lec	Start Week	Modules	Topics	Assignments
1	3 Aug	Mod 1.1, 1.2	Course Overview, Design Process	
2	10 Aug	Mod 2.1, 2.2, 2.3, 2.4	Unity3D Introduction, Introduction C#, Variables and Components, Hello World	
3	17 Aug	Mod 3.1, 3.2, 3.3	Booleans, Loops, Lists and Arrays	Assign 1 21 Aug, 11:00 pm
4	24 Aug	Mod 4.1, 4.2	Functions and Parameters, Debugging	
5	31 Aug	Mod 5.1, 5.2	Classes, Object Oriented	
6	7 Sep	Mod 6.1, 6.2, 6.3	Agile Processes, Risks and Prototypes, Testing	
7	14 Sep	Mod 7.1, 7.2	Puzzles, Guiding the Player	Assign 2 18 Sep, 11:00 pm
8	21 Sep	Mod 8.1	Game Physics	
9	12 Sep	Mod 9.1	AI for Games	
10	19 Oct	Mod 10.1, 10.2	Game Interface, Storytelling in Games	
11	26 Oct	Mod 11.1, 11.2	Graphics Pipeline, Animation in Games	Assign 3 1 Nov, 11:00pm
12	2 Nov	Mod 12.1, 12.2	Networked Games, Course Review	


Course Details
Unity 3D and C#
Core Game Concepts
Game Design
Development Process
Assignments
Course Overview
Puzzles
Design Process
Testing
Unity3D Introduction
Game Physics
Game Interfaces
Introduction C#
Risks and Prototypes
Guiding the Player
AI for Games
Graphics Pipeline
Variables and Components
Object Oriented
Agile Processes
Storytelling in Games
Course Review and Exam
Booleans
Hello World
Debugging
Classes
Loops
Lists and Arrays
Functions and Parameters
Networked Games
Animation in Games
Assign 2
Assign 1
Assign 3
Lec 1
Lec 2
Lec 3
Lec 4
Lec 5
Lec 6
Lec 7
Lec 8
Lec 9
Lec 10
Lec 11
Lec 12

Classes – (Chapter 26)



CLASSES

Classes – Topics

- Understanding Classes
  - The Anatomy of a Class
- Class Inheritance
  - Superclasses and Subclasses
  - Virtual and Override

Understanding Classes

Classes are the key concept in Object-Oriented Programming

A class is a definition of a type of object

There can be many instances of a single class

- Each person in this classroom could be thought of as an instance of the Human class

C# classes combine data and functionality

- Classes have variables, which are called fields
- Classes have functions, which are called methods

You're already using classes! - Each C# script you've written is a class

Classes represent objects in your game

Class Example

Example: A character in a standard RPG

Fields you would want for each character

```
string    name;       // The character's name
float     health;     // The amount of health she has
float     healthMax; // Her maximum amount of health
List<Item> inventory;// List of Items in her inventory
List<Item> equipped; // A List of Items she has equipped
```

Methods you would want

```
void Move(Vector3 newLoc) {…} // Moves her to newLoc

// Attacks target with the current weapon or spell
void Attack(Character target) {…}

void TakeDamage(float dmgAmt) {…} // Reduces health

// Adds an Item to the equipped List
void Equip(Item newItem) {…}
```

Class Example

Example: A character in a standard RPG

Character
string name; float health; float healthMax; List<Item> inventory; List<Item> equipped;
void Move(Vector3 newLoc) {…} void Attack(Character target) {…} void TakeDamage(float dmgAmt) {…} void Equip(Item newItem) {…}

Fields you would want for each character

Methods you would want

Example - Enemy Class

We'll explore each part of a class named Enemy

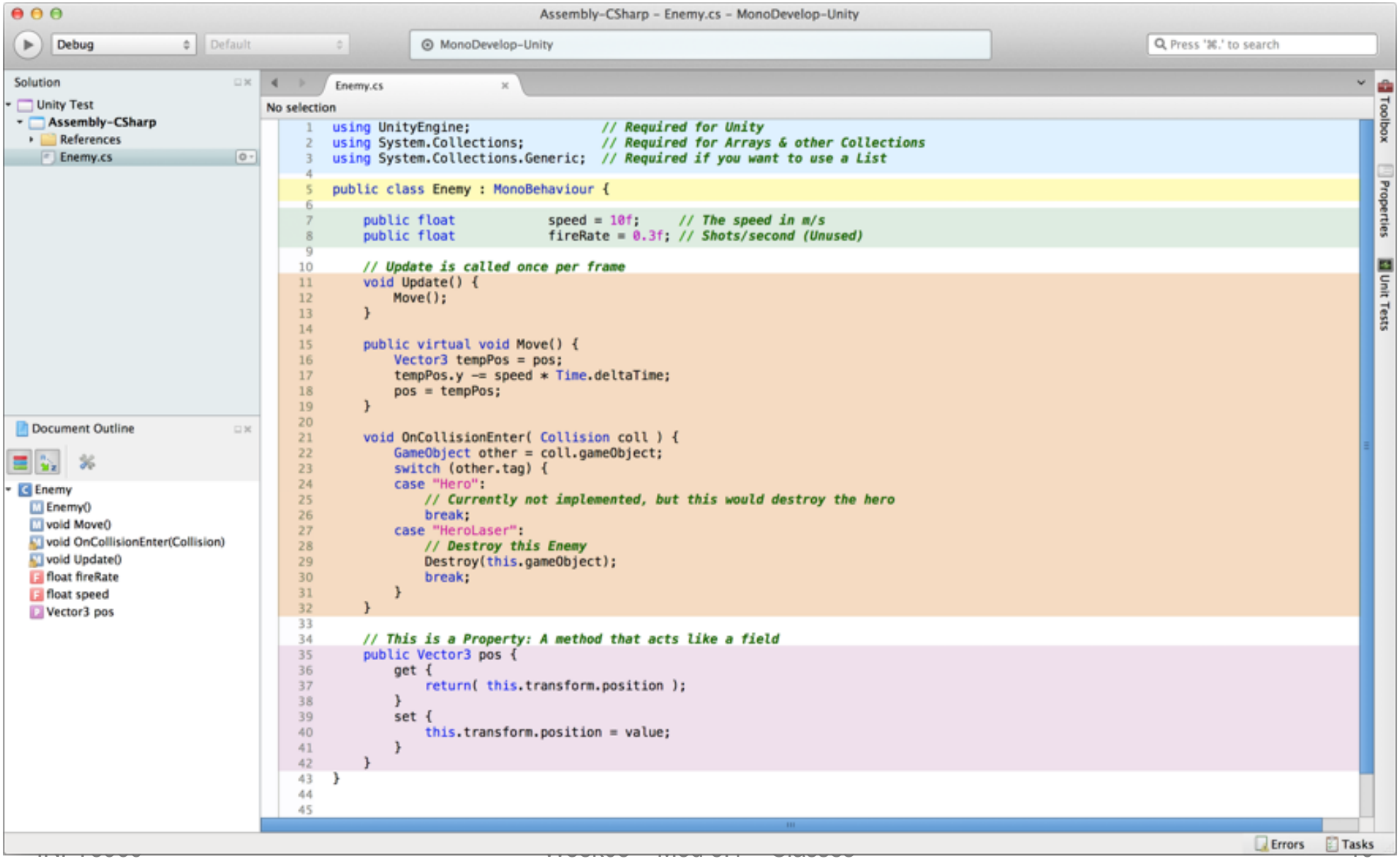
- The Enemy class is for a simple top-down space shooter game
- An Enemy instance moves down the screen at a speed of 10

Includes

- Include code libraries in your project
- Enables standard Unity libraries and objects e.g., GameObject, MonoBehaviour, Transform, Renderer, etc.

```
1 using UnityEngine;                    // Required for Unity
2 using System.Collections;             // Included by Unity's default
3 using System.Collections.Generic;     // Required to use a List
```

Anatomy of a Class




```
1  using UnityEngine;                // Required for Unity
2  using System.Collections;         // Required for Arrays & other Collections
3  using System.Collections.Generic; // Required if you want to use a List
4
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 }
44
45
```

Example - Enemy Class

The Class Declaration

- Declares the name of the class and its superclass
- Enemy is a class that extends its superclass `MonoBehaviour`

```
5 public class Enemy : MonoBehaviour {
```

Fields

Fields are variables that are part of the class

Fields marked public are able to be seen by other classes and by other instances of this class

Fields marked private are only able to be seen by this one instance of a class

- Private fields are secrets
- And a safer way to program than always using public fields
- Public fields are used throughout the book so that the field values appear and are editable in the Unity Inspector

```
7     public float  speed = 10f;      // The speed in m/s
8     public float  fireRate = 0.3f; // Shots per second (Unused)
```

- Declares two public fields for all instances of the Enemy class
- Each instance has its own value for speed and fireRate

Methods

Functions that are part of the class - Can also be marked public or private

```
11      void Update() {
12          Move();
13      }
14
        // Move down the screen at speed
15      public virtual void Move() {
16          Vector3 tempPos = pos;
            // Makes it Time-Based!
17          tempPos.y -= speed * Time.deltaTime;
18          pos = tempPos;
19      }
```

Note that Move is a virtual function
Virtual functions can be overridden by functions of the same name in a subclass (we'll cover this shortly)

Properties

Properties are methods masquerading as fields
Properties can only exist within classes

```
35      public Vector3 pos {
36          get {
37              return( this.transform.position );
38           }

39          set {
40              this.transform.position = value;
41          }
42      }
```

This property simplifies setting the transform.position of this Enemy

Class Instances as Components

In Unity, all class instances are treated as GameObject Components

The class instance can be accessed using

`GetComponent<>()`

```
Enemy thisEnemy = this.gameObject.GetComponent<Enemy>();
```

From there, any public variable can be accessed

```
thisEnemy.speed = 20f;  // Increase speed of Enemy to 20
```

Many C# scripts can be attached to a single GameObject

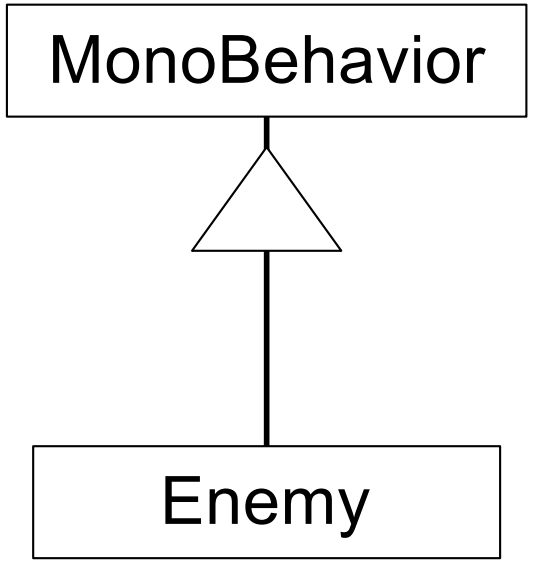
Class Inheritance

Most classes inherit from another class

```
5 public class Enemy : MonoBehavior {…}
```

Enemy inherits from MonoBehaviour

Enemy is the subclass of MonoBehavior



Class Inheritance

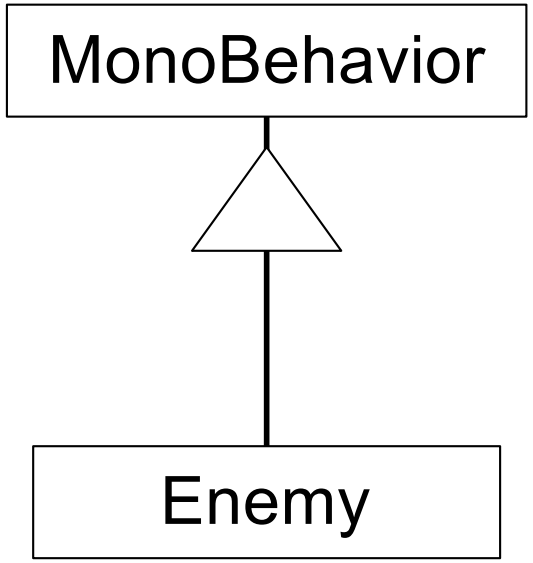
`MonoBehavior` is called the superclass, base class, or parent class of `Enemy`

This means that `Enemy` inherits all of `MonoBehaviour's` fields and methods

Example inherited fields:
`gameObject, transform, renderer,...`

Example inherited methods:
`GetComponent<>(), Invoke(), StartCoroutine(), etc`

Inheriting from `MonoBehaviour` is what makes Enemy able to act like a GameObject component



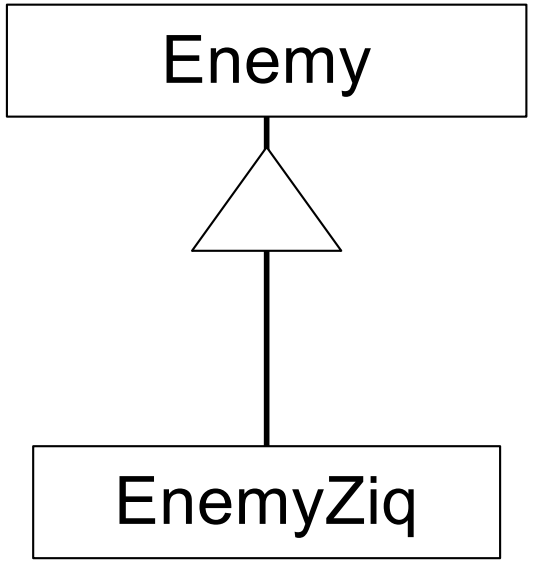
Class Inheritance

We can create a class that inherits from Enemy!

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class EnemyZig : Enemy {
5      // EnemyZig inherits ALL its behavior from Enemy
6  }
```

If this class is attached to a different GameObject, that GameObject will act exactly like an Enemy - It will also move down the screen at a rate of 10m/second

Move() can be overridden because it is a virtual function - This means that EnemyZig can have its own version of Move()!



Polymorphism

`EnemyZig.Move()` overrides `Enemy.Move()`

```
 4  public class EnemyZig : Enemy {
 5      public override void Move () {
 6          Vector3 tempPos = pos;
 7          tempPos.x = Mathf.Sin(Time.time * Mathf.PI*2) * 4;
 8          pos = tempPos;      // Uses the pos property of the superclass
 9          base.Move();        // Calls Move() on the superclass
10      }
11  }
```

Now, when the `Update()` method in Enemy calls `Move()`, `EnemyZig` instances will use `EnemyZig.Move()` instead

- This moves the `EnemyZig` instance back and forth horizontally
- On line 9, `base.Move()` calls the `Move()` function on `EnemyZig`'s base class, `Enemy` -This causes `EnemyZig` instances to continue to move downward as well

Summary

Classes combine data (fields) and functionality (methods)

Classes can inherit from each other

Classes are used in Unity as GameObject Components

Understanding classes is the key to object-oriented programming (OOP)

- Before OOP, games were often a single, very large function
- With OOP, each object in the game is a class, and each class can think for itself