

COMP140-GAM160: Further Programming

3: Inheritance and Polymorphism

Learning outcomes

- ▶ **Understand** Inheritance in Object Orientated Programming
- ▶ **Understand** Polymorphism role in creating Games
- ▶ **Apply** your knowledge of Inheritance and Polymorphism to programming problems

Classes Review

Classes

- ▶ Let us look at Classes again
- ▶ Classes allow us to create our own data types
- ▶ They consist of a series of data(variables) and functions that operate on the data
- ▶ Functions and variables inside the class can be marked with the following **access specifiers**
 - ▶ **Public**: Can be accessed directly
 - ▶ **Private**: Can only be accessed inside the class
 - ▶ **Protected**: Acts like private, but child classes can access

Class Examples - C++

```
class Player
{
public:
    Player()
    {
        Health=100;
    };

    void TakeDamage(int health)
    {
        Health-=health;
    };

    void HealDamage(int health)
    {
        Health+=health;
    };

    ~Player() {};
private:
    int Health;
};
```

Class Examples - C# Unity

```
class Player
{
    private int Health;

    public Player()
    {
        Health=100;
    }

    public void TakeDamage(int health)
    {
        Health-=health;
    }

    public void HealDamage(int health)
    {
        Health+=health;
    }
}
```

Classes vs Structs

- ▶ A **Struct** is pretty much the same as a **Class**
- ▶ The only difference in functionality, by default:
 - ▶ Everything in a **Class** is **private**
 - ▶ Everything in a **Struct** is **public**
- ▶ Difference by convention:
 - ▶ Structs are used for holding related data and tend not to have functions
 - ▶ , Classes hold data and functions

Creating an Instance - C++

```
\\Creating on the stack, this will be deleted when it drops out of scope  
Player player1=Player();
```

```
\\Call take damage function, notice we use . to access functions  
player1.TakeDamage(20);
```

```
\\Creating on the Heap, please delete!!  
Player * player2=new Player();
```

```
\\Call take damage function, note we use -> to access functions  
player2->TakeDamage(20);
```

```
\\Deleting player2 on the heap  
if (player2)  
{  
    delete player2;  
    player2=nullptr;  
}
```


Creating an Instance - C#

```
\\Create a player  
Player player1=new Player();
```

```
\\Call take Damage  
player1.TakeDamage(50);
```

Constructor & Deconstructor

- ▶ **Constructors** are called when you create an instance
- ▶ Constructors can take in zero or many parameters
- ▶ You need to declare different version of the constructor
- ▶ Deconstructors are called when the instance has been deleted (by the dropping out of scope, or deleted in C++)
- ▶ Constructors and Deconstructors have to be names the same as the class

Constructors C#

```
class Player
{
    private int Health;
    private int Strength;

    public Player()
    {
        Health=100;
        Strength=10;
    }

    public Player(int health)
    {
        Health=health;
        Strength=10;
    }

    public Player(int health,int strength)
    {
        Health=health;
        Strength=strength;
    }
}
```

Class Design Principles

Inheritance

Polymorphism

Collections & Polymorphism

Coffee Break

Static Keyword & Singletons

Exercise

References