- is a feature of inheritance that allows classes to have more than one type
- **upcasting** - declare an object as base class type and call a constructor of one of it's child classes (it will be treated as one of the base class type - can call methods only in base class[except the virtual ones])
- **downcasting** - include the name of the child class in enclosing brackets or create a reference. example: ***((ChildClass)myClass).method name*** or ***ChildClass myChild = (ChildClass)myClass;***

```
using UnityEngine;
using System.Collections;

public class Fruit
{
    public Fruit()
    {
        Debug.Log("1st Fruit Constructor Called");
    }

    public void Chop()
    {
        Debug.Log("The fruit has been chopped.");
    }

    public void SayHello()
    {
        Debug.Log("Hello, I am a fruit.");
    }
}
```

```
using UnityEngine;
using System.Collections;

public class Apple : Fruit
{
    public Apple()
    {
        Debug.Log("1st Apple Constructor Called");
    }
```

```
    //Apple has its own version of Chop() and SayHello().
    //When running the scripts, notice when Fruit's version
    //of these methods are called and when Apple's version
    //of these methods are called.
    //In this example, the "new" keyword is used to supress
    //warnings from Unity while not overriding the methods
    //in the Apple class.
    public new void Chop()
    {
        Debug.Log("The apple has been chopped.");
    }

    public new void SayHello()
    {
        Debug.Log("Hello, I am an apple.");
    }
}


using UnityEngine;
using System.Collections;

public class FruitSalad : MonoBehaviour
{
    void Start ()
    {
        //Notice here how the variable "myFruit" is of type
        //Fruit but is being assigned a reference to an Apple. This
        //works because of Polymorphism. Since an Apple is a Fruit,
        //this works just fine. While the Apple reference is stored
        //in a Fruit variable, it can only be used like a Fruit
        Fruit myFruit = new Apple();

        myFruit.SayHello();
        myFruit.Chop();

        //This is called downcasting. The variable "myFruit" which is
        //of type Fruit, actually contains a reference to an Apple. Therefore,
```

```
        //it can safely be turned back into an Apple variable. This allows
        //it to be used like an Apple, where before it could only be used
        //like a Fruit.
        Apple myApple = (Apple)myFruit;

        myApple.SayHello();
        myApple.Chop();
    }
}
```