Inheritance

This lesson discusses inheritance in detail as well as constructors in derived and base classes

WE'LL COVER THE FOLLOWING

What is Inheritance

- Provides a way to create a **new** class from an **existing** class.
- **New** class is a *specialized* version of the **existing** class.
- Allows the **new** class to **overload** *methods* from the **existing** class.

Terminology

- **Base Class**(or Parent): *inherited* by **child** class.
- **Derived Class**(or child): *inherits* from base class.

Characteristics

A **derived** class has:

- All *members* defined in the **derived** class.
- All *members* declared in the **base** class.

A **derived** class can:

- Use all `public` members defined in the **derived** class.
- Use all `public` members defined in the **base** class.
- **Override** an *inherited* member

A **derived** class cannot:

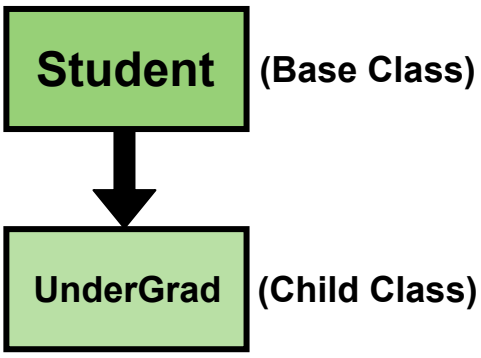
- *Inherit constructors* and **destructors**
- Change the *definition* of an *inherited member*

Notation

Let's take a look at the notation for these **two** types.

```
class Student{ //base class
 //body
}

//derived class inherits from another class syntactically by using the : operator
class UnderGrad : Student{
      //body
}
```



Example

Let's consider an example with *base* class `Shape` and *derived* class `Square`.

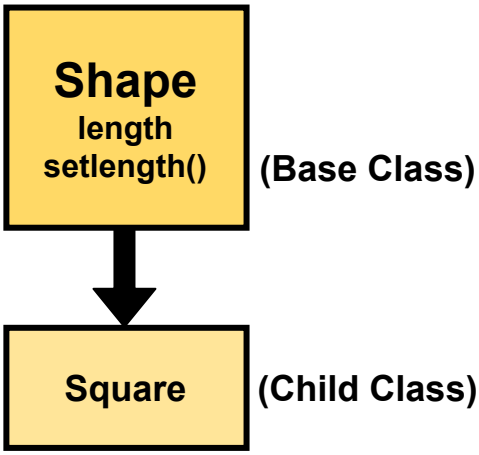
```
using System;

// Base class
class Shape {
   public Shape(){length = 0;} //default constructor
   public void setlength(int l) {length = l;}
   protected int length;
}

// Derived class
class Sqaure: Shape {
   public Sqaure()  {length = 0;} //declaring and initializing derived class constructor
```

```
   public int get_Area(){ return (length * length); }
}

class Program
{
  static void Main(){
   Sqaure sq = new Sqaure(); //making object of child class Sqaure
   sq.setlength(5); //setting length equal to 5
   // Print the area of the object.
   Console.WriteLine("Total area of sqaure is: {0}",sq.get_Area());
  }
}
```



As you can see in the example above,

- The `shape` class is the *parent* class whereas the `sqaure` class is the *child* class *derived* from it.
- In our *child* class `Square`, we use *members* from the *parent* class such as
  - the `protected` `length` variable which gets *initialized* to **zero** in the *default* constructor.
  - `Length` also gets used in *child* class function `get_Area` to compute the *area* of the *square*.
- In `Main` the `setlength` function which is a `public` *member* function of the *parent* class is accessible to the *child* class object `sq`
  - The **dot** operator is used to access `setlength` in the `Main`.

Constructors

When creating an **instance** of `Square` *class*, the **base** *class* **default** *constructor* (**without parameters**) will be called if there is no explicit call to another *constructor* in the **parent** class.

In our case, `Shape` class *constructor* will be called and then `Square` class *constructor*.

In the above example, our `Square` class *constructor* calls the *default constructor* of the `Shape` class. If you want, you can specify which *constructor* should be called: it is possible to call **any** *constructor* which is *defined* in the **parent** class using the `base` *keyword*.

Example

Consider the *example* below for better understanding.

```
using System;

class Shape {
  protected int length;
  public Shape() { //default constructor
    Console.WriteLine("Shape's default constructor");
  }
  public Shape(int length) { //constructor with parameters
    this.length = length;
    Console.WriteLine("Shape's constructor with 1 parameter");
    Console.WriteLine(this.length);
  }
}

class Square: Shape {
  public Square(): base() { //calling Shape class default constructor using base
    Console.WriteLine("Square's default constructor");
  }
  public Square(int length): base(length) { //calling Shape class constructor with parameters
    Console.WriteLine("Sqaure's constructor with 1 parameter");
    Console.WriteLine(this.length);
  }
  public int get_Area(){ return (length * length); } //method computing area of square
}

class Program {
  static void Main() {
    Square sq1 = new Square(); //making object of child class Sqaure
    Square sq2 = new Square(5); //setting length equal to 5
    Console.WriteLine("Area of sq1 is: {0}",sq1.get_Area());
    Console.WriteLine("Area of sq2 is: {0}",sq2.get_Area());
  }
}
```

In the code above:

- We have **2** *constructors* in each *class*.
- We are using `base` *keyword* which is a *reference* to the **parent** class.
- In our case, when we create an instance of `Square` class in **line 28**
    - The runtime first calls the `Square()`, which is the **parameterless** *constructor*. But its *body* doesn't work immediately.
    - After the *parentheses* of the *constructor*, we have a call: `base()`, which means that when we call the **default** `Square` *constructor*, it will, in turn, call the *parent's* **default** *constructor*.
    - After the **parent's** *constructor* runs, it will return and then, finally, run the `Square()` *constructor's* body.
- Members in the **parent** *class* which are not `private` are *inherited* by the **child** *class*, meaning that `Square` will also have the `length` field.
    - In this case in **line 29**, we passed an *argument* to our *constructor*.
    - It then passes the *argument* to the **parent** class *constructor* with a *parameter*, which *initializes* the `length` field.

Interesting so far? In the next lesson we will discuss the *polymorphism* and *virtual methods*. Keep on reading to learn more!