What is a class?
• A container for variables that represent an object along with the methods used to operate on those variables.
• An organizational structure that allows a programmer to think of code as an object similar to the way we think of the objects in our life.

In real life an object might be say a pencil. It has certain properties that are similar to private variables such as the hardness and length of its lead, the amount of lead left, the eraser type, etc. There are also a number of operations I can do with a pencil which are similar to methods. I can use a pencil to write or erase so those things are the pencil methods.

You can also think of a class as a new variable type just like int, float, or double. Suppose for example, I want to create a new variable of type Point so that I can declare a variable in C# to be a Point in a manner similar to the way in which I declare a variable to be of type int. A Point class would have two private variables (in two dimensions) which represent the points position in Cartesian coordinates. We could call these variables x and y and make them floats. The C# syntax would look like this:

```csharp
class Point
{
    private float x;          //private variables
    private float y;
    //
    public floatGetX()
    { return x; }
    public void SetX(float x1)
    { x = x1; }
    public floatGetY()
    { return y; }
    public void SetY(float y1)
    { y = y1; }
}
```

As you might expect, the private variables are not available outside the class so to make them available we create accessor methods that are public like this:

```csharp
class Point
{
    private float x;          //private variables
    private float y;
    //
    public float GetX()       //x accessor methods
    { return x; }
    public void SetX(float x1)
    { x = x1; }
    public float GetY()       //y accessor methods
    { return y; }
    public void SetY(float y1)
    { y = y1; }
}
```

If we add this class to a project we can declare variables to be of type Point and each Point variable will have an x and a y coordinate. The syntax for such a declaration might look like this:

```csharp
Point p = new Point();
p.SetX(1.0f);
p.SetY(2.0f);
```
Point p1 = new Point();

We don't have direct access to the private methods and variables in p1 but we can access the public methods using the dot operator like this:

p1.SetX();

We could create a number of other public methods that allow us to access and manipulate points. For example, we might create a Distance method that provides the distance between two points, etc.

Accessor methods for the private variables are used so often that they have been turned into something called the class's properties. Normally, we use lower case letters for variables and upper case letters for properties. A property for the x private variable might be named X for example.

In our example of the point class we can replace the accessor methods with properties where the syntax for a property for X looks like this:

```csharp
public float X
{
    get
    {
        return x;
    }
    set
    {
        x = value;
    }
}
```

Notice that this property is written like a method except it has no argument list. The property encapsulates two methods which are always called get and set. The get method returns the value of the property and the set method allows a user to set the value of a property through the value key word. For example, in a main program I could write:

```csharp
int x1;
Point p1 = new Point();
p1.X = 5;    //set accessor
x1 = p1.X;   //get accessor
```

**Constructors:**

We are still missing a good way to initialize the private variables in a class. This is done with a public method which is called a constructor. When you create a variable such as a double you write something like this:

double x = 22.3;

When the C# compiler sees this statement it effectively runs a constructor for x. The constructor creates a memory space for a variable of type double and assigns it an address which you call by the name x. For this example, it also puts the data 22.3 into that memory space. The same procedure has to happen for a class. For our Point class, there are two private variables of type float so the compiler has to make room for those. If you don't write your own constructor, C#
will create a default constructor which will create the space in memory for the private variables but it will not assign these variable any value. If you want to initialize the private variables in a class then you must create your own constructor for the class and do the initialization yourself. For our Point class I have created two constructors that look like this:

```java
class Point
{
    public Point() //Empty constructor
    {x = 0;
     y = 0;
    }
    public Point(float x1, float y1) //constructor with arguments
    {x = x1;
     y = y1;
    }
}
```

Notice that the constructor is created like a method except that is is always public and it has no return type.

The first constructor has no arguments but it initializes the private variables to 0. I could invoke it in a main program like this:

```java
Point p1 = new Point();
```

The second constructor takes two arguments and it allows the user to set the private variables to any initial value. It is used like this:

```java
Point p2 = new Point(5.2, 4.6);
```

The two constructors have the same name, Point, but have a different argument list and this allows the compiler to distinguish between them. We say the constructor is overloaded.

A complete version of the Point class is on the following page.
class Point
{
    public Point() //Empty constructor
    {
        x = 0;
        y = 0;
    }
    public Point(float x1, float y1) //constructor with arguments
    {
        x = x1;
        y = y1;
    }
    private float x; //private variables
    private float y;
    
    public float GetX() //x accessor methods
    {
        return x;
    }
    public void SetX(float x1)
    {
        x = x1;
    }
    public float GetY() //y accessor methods
    {
        return y;
    }
    public void SetY(float y1)
    {
        y = y1;
    }
    
    //Properties
    public float X
    {
        get
        {
            return x;
        }
        set
        {
            x = value;
        }
    }
    public float Y
    {
        get
        {
            return y;
        }
        set
        {
            y = value;
        }
    }
}