Encapsulation
WHAT, WHY and HOW

What is a Class ?

Class

A class is a collection of both attributes (Data) and Operations (member functions/behaviours) those applied to the same data.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

We have discussed a class should be responsible for its own data and operations.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

Let's say the class wants to restrict all of its object such that they can not set the stock limit below the -1 or greater than 100.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

But for this specific class it seems it is not applying the required constraints on stock.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

Based on our current knowledge how can we apply the required constraint?

"Let's say the class wants to restrict all of its object such that they can not set the stock limit below the -1 or greater than 100"

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

These restrictions on data can be applied where the object is created (main function).

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

These restrictions on data can be applied where the object is created (main function).

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

Could you identify any problem with this approach ?

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

However it is serving the purpose, but here the class is not responsible for the data consistency. It has to rely on some external code.

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

This approach comes up with same problems that we discussed earlier when data and operations were not packed in the class.

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

When the responsibility is not at a single place, it is possible the external code miss or skip to add validity check code.

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

Also, this may cause the validity check code for data is distributed and repeated at different locations of the project.

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Class

In that case, if we need to change the restriction on stock from 100 to 200 we have to find all places where these conditions are applied.

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
    public float calculateTax()
    {
        float tax;
        if (category == "Grocery")
        {
            tax = price * 10 / 100F;
        }
        else if (category == "Fruit")
        {
            tax = price * 5 / 100F;
        }
        else
        {
            tax = price * 15 / 100F;
        }
        return tax;
    }
}
```

Major Problems with the Approach

When class does not take the responsibility of its own data, the responsibility is shifted to other part of the code and it leads two major problems

1. Other parts may skip the validation check on data that lead to data inconsistency.
2. There is no way to find at how many places the validity code is distributed and if there is some change required, it has to made at multiple places. It raises the problems of maintainability.

Class is Responsible

In object oriented programming, class should act as the custodian of its own data and it should know what kind of restrictions are required to be added on the data.

Class is Responsible

To Do the task, class has to perform two tasks.

1. Restrict Access of Attributes to external world
2. Allow another way to change the value of the attribute that will check whether the new value is within the required constraint or not.

Encapsulation

When we want to secure data for direct access from external world (objects of the class, other classes), this is called Encapsulation.

How to Do Encapsulation: Access Modifiers

Access modifiers are an integral part of object oriented programming. Access modifiers are used to implement Encapsulation of OOP. Access modifiers allow you to define who can or can not access to certain features.

Modifier	Description
public	There are no restrictions on accessing public members.
private	Access is limited to within the class definition. This is the default access modifier type if none is formally specified

How to Do Encapsulation?

We use private keyword to make all the data members as private.

```
class Product
{
    private string name;
    private string category;
    private int price;
    private int stock;
    private int minimumStock;
}
```

How to Do Encapsulation?

It means that the data members can not be accessed by any instance of this class.

```
class Product
{
    private string name;
    private string category;
    private int price;
    private int stock;
    private int minimumStock;
}
```

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
    Console.Read();
}
```

How to Do Encapsulation?

It means that the data members can not be accessed by any instance of this class.

```
class Product
{
    private string name;
    private string category;
    private int price;
    private int stock;
    private int minimumStock;
}
```

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    if (stock <= -1 || stock > 100){
        Console.WriteLine("Invalid Input");
    }
    else{
        p.stock = stock;
    }
```

CS0122	'Product.name' is inaccessible due to its protection level	Week2	Program.cs
CS0122	'Product.stock' is inaccessible due to its protection level	Week2	Program.cs

Constraints on Data

Remember we wanted to add condition on the stock value, Now how to add constraint on the data update?

Constraints on Data

We can add condition inside the setter functions such as

```
class Product{
    private string name;
    private string category;
    private int price;
    private int stock;
    private int minimumStock;
    public bool setStock(int stock)     //Member Function
    {
       if (stock <= -1 || stock > 100){
     return false //Invalid Input
       }

       this.stock = stock;
       return true;
    }
}
```

How to Do Encapsulation?

Now, we update attribute through the setter function.

```
class Product
{
    private string name;
    private string category;
    private int price;
    private int stock;
    private int minimumStock;
}
```

```
static void Main(string[] args){
    Product p = new Product();
    Console.WriteLine("Enter Product Name");
    p.name = Console.ReadLine();
    Console.WriteLine("Enter Stock");
    int stock = int.Parse(Console.ReadLine());
    bool stockFlag = p.setStock(stock)
    if (!stockFlag){
        Console.WriteLine("Invalid Input");
    }
  }
    Console.Read();
```

CS0122	'Product.name' is inaccessible due to its protection level	Week2	Program.cs
CS0122	'Product.stock' is inaccessible due to its protection level	Week2	Program.cs

How to Do Access the Variable?

Now, the question is if we want to view the private data member, how can we do that?

How to Do Access the Variable?

If we want the object should be able to view the value of the private data member, we can provide the access through the public member function.

```
class Product
{
 private string name;
 private string category;
 private int price;
 private int stock;
 private int minimumStock;
 public int getStock()     //Member Function
    {
       return stock;
    }

}
```

How to Do Access the Variable?

This is called the Getter function of the the Class.

```
class Product
{
 private string name;
 private string category;
 private int price;
 private int stock;
 private int minimumStock;
 public int getStock()     //Member Function
    {
       return stock;
    }

}
```

How to Do Access the Variable?

In the same way we can define the getter functions for all the attributes of the class.

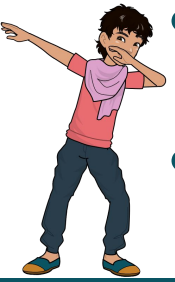
```
class Product
{
 private string name;
 private string category;
 private int price;
 private int stock;
 private int minimumStock;
 public string getProductName()     //Member Function
    {
       return name;
    }

}
```

ReadOnly Attributes

How to make attribute Read Only.

Conclusion

- When we want to secure data for direct access from external world (objects of the class, other classes), this is called Encapsulation.
- We can make data member private, so that it can not be accessed directly from outside of the class.
- Access modifiers are used to implement Encapsulation of OOP. Access modifiers allow you to define who does or who doesn't have access to certain features.
- We can provide access to the data through the public getter and setter functions
- We can add constraints on the data inside the setter functions



Learning Objective

Explain the need to WHY, WHAT and How of the Data Encapsulation.

Self Assessment:

1. Create a class named MyPoint. This contains.

- Two private instance variables x (int) and y (int).
- A default (or "no-argument") constructor that construct a point at the default location of (0, 0).
- A parameterized constructor that constructs a point with the given x and y coordinates.
- Getter and setter for the instance variables x and y.
- A method setXY() to set both x and y.
- A method called distance(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates

Self Assessment:

- The Class Diagram of MyPoint is as follows

MyPoint
- x: int - y: int
+ MyPoint() + MyPoint(int x, int y) + getX():int + setX(int x):bool + getY():int + setY(int y):bool + setXY(int x,int y):bool + distance(int x, int y):int

Self Assessment:

- Some times, we do not mention the getter and setter function within the CRC it is implied and understood.

MyPoint
- x: int - y: int
+ MyPoint() + MyPoint(int x1, int y1) + distance(int x2, int y2):int