What is Encapsulation in C#?

The process of binding the data and functions together into a single unit (i.e. class) is called encapsulation in C#. Or you can say that the process of defining a class by hiding its internal data members from outside the class and accessing those internal data members only through publicly exposed methods (setter and getter methods) or properties with proper validations is called encapsulation.

How can we implement Encapsulation in C#?

In C# Encapsulation is implemented

1.By declaring the variables as private (to restrict its direct access from outside the class)

2.By defining one pair of public setter and getter methods or properties to access private variables.

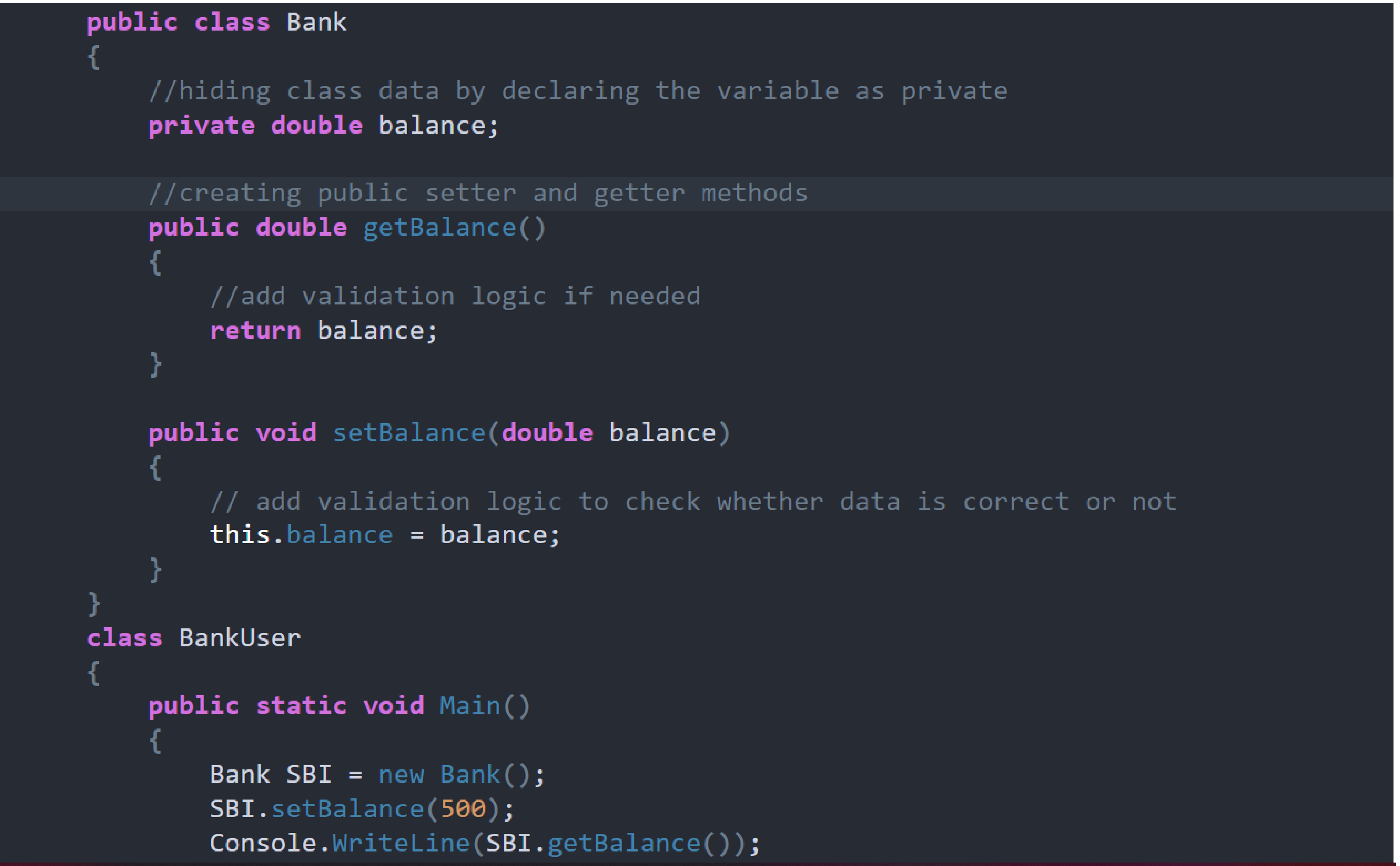
We declare variables as private to stop accessing them directly from outside the class. The public setter and getter methods or properties are used to access the private variables from outside the class with proper validations. If we provide direct access to variables then we cannot validate the data before storing it in the variable.

What is the problem if we don’t follow encapsulation in C# while designing a class?

If we don’t the encapsulation principle while designing the class, then we cannot validate the user-given data according to our business requirement as well as it is very difficult to handle future changes.

Let us understand this with an example. Assume in the initial project requirement, the client did not mention that the application should not allow the negative number to store in that variable. So, we give direct access to the variable

Let us see an example to understand this concept. In the following example, we declare the balance variable as private in the Bank class, and hence it can not be accessed directly outside of the Bank class. In order to access this balance variable, we have exposed two public methods i.e. getBalance and setBalance. The getBalance method (Accessors) is used to fetch the value store in the balance variable whereas the setBalance method (Mutator) is used to set the value in the balance variable.



```
    public class Bank
    {
        //hiding class data by declaring the variable as private
        private double balance;

        //creating public setter and getter methods
        public double getBalance()
        {
            //add validation logic if needed
            return balance;
        }

        public void setBalance(double balance)
        {
            // add validation logic to check whether data is correct or not
            this.balance = balance;
        }
    }
    class BankUser
    {
        public static void Main()
        {
            Bank SBI = new Bank();
            SBI.setBalance(500);
            Console.WriteLine(SBI.getBalance());
```

What is Abstraction in C#?

The process of representing the essential features without including the background details is called Abstraction. In simple words, we can say that it is a process of defining a class by providing necessary details to call the object operations (i.e. methods) by hiding or removing its implementation details is called abstraction in C#. It means we need to expose what is necessary and compulsory and we need to hide the unnecessary things from the outside world. In C# we can hide the member of a class by using private access modifiers.

As we know a car is made of many things, such as the name of the car, the color of the car, gear, breaks, steering, silencer, the battery of the car, engine of the car, etc. Now you want to ride a car. So to ride a car what are the things you should know. The things a car driver should know are as follows.

1.Name of the Car
2.The color of the Car
3.Gear
4.Break
5.Steering

So these are the things that should be exposed and know by the car driver before riding the car. The things which should be hidden to a Car rider as are follows

1.The engine of the car
2.Diesel Engine
3.Silencer

What is inheritance in C#?

The process of creating a new class from an existing class such that the new class acquires all the properties and behaviors of the existing class is called inheritance. The properties (or behaviors) are transferred from which class is called the superclass or parent class or base class whereas the class which derives the properties or behaviors from the superclass is known as a subclass or child class or derived class. In simple words, inheritance means to take something that is already made (or available).

C#.NET classified the inheritance into two categories, such as

1.Implementation inheritance.

2.Interface inheritance

Implementation inheritance: This is the commonly used inheritance. Whenever a class is derived from another class then it is known as implementation inheritance.

Interface inheritance: This type of inheritance is taken from Java. Whenever a class is derived from an interface then it is known as interface inheritance.

1.Single Inheritance: When a class is derived from a single base class then the inheritance is called single inheritance.

2.Multilevel Inheritance: When a derived class is created from another derived class, then that type of inheritance is called multilevel inheritance.

3.Hierarchical Inheritance: When more than one derived class is created from a single base class then it is called Hierarchical inheritance.

4.Hybrid Inheritance: Hybrid Inheritance is the inheritance that is the combination of any single, hierarchical, and multilevel inheritances.

5.Multiple Inheritance: When a derived class is created from more than one base class then such type of inheritance is called multiple inheritances. But multiple inheritances are not supported by .net using classes and can be done using interfaces.