Kapsülleme, nesne yönelimli programlamanın (OOP) temel ilkelerinden biridir. Kapsülleme, verilerin ve bu verilere erişimi sağlayan fonksiyonların bir arada gruplandığı bir yapı oluşturulmasına olanak tanır. Bu sayede dış dünyadan veri gizlenir ve veriye yalnızca belirli metotlar üzerinden erişilebilir hale gelir.

**Kapsülleme Nedir?**

Kapsülleme, bir sınıfın (class) içindeki verileri (fields) ve bu verilere erişim sağlayan metodları (methods) bir arada tutarak, dışarıdan erişimi kısıtlamaktır. Bu sayede veriler üzerinde kontrollü bir erişim sağlanır ve veri güvenliği artırılır. Kapsülleme, veri gizleme (data hiding) ve arabirim (interface) sağlamada oldukça kullanışlıdır.

C# dilinde kapsülleme, genellikle sınıf üyelerinin erişim belirleyicileriyle (access modifiers) kontrol edilir. Bu belirleyiciler şunlardır:

* **public**: Herkes erişebilir.
* **private**: Sadece sınıf içi erişim izni verir.
* **protected**: Sadece sınıf ve sınıfı miras alan sınıflar erişebilir.
* **internal**: Aynı assembly içindeki sınıflar erişebilir.
* **protected internal**: Aynı assembly içindeki sınıflar ve miras alınmış sınıflar erişebilir.

**Kapsülleme Avantajları:**

1. **Veri güvenliği**: Verilere doğrudan erişim yerine kontrollü bir erişim sağlanır.
2. **Kod bakımının kolaylaşması**: Verilerin nasıl değiştirildiği, dışarıdan bağımsız hale getirilir.
3. **Modülerlik**: Sınıfın iç yapısı dışarıdan soyutlanır, böylece daha düzenli bir yapıya ulaşılır.

**Basitten Zora 4 Uygulamalı Örnek**

**Örnek 1: Temel Kapsülleme**

Bu örnekte, bir sınıfın özel (private) veri üyelerine erişim için **getter** ve **setter** metodları kullanacağız.

using System;

public class Person

{

// Private field

private string name;

// Public property for name (Getter/Setter)

public string Name

{

get { return name; }

set { name = value; }

}

public void DisplayInfo()

{

Console.WriteLine("Name: " + Name);

}

}

class Program

{

static void Main()

{

Person person = new Person();

person.Name = "Ahmet"; // Set the name using the setter

person.DisplayInfo(); // Get the name using the getter

}

}

**Açıklama**: Burada name adlı özel (private) bir alanımız var. Bu alana dışarıdan erişmek için Name adlı bir public property oluşturduk. Name property'si, name alanına erişimi kontrol eder.

**Örnek 2: Yalnızca Okunabilir Özellik (Readonly)**

Bu örnekte, bir sınıfa yalnızca okunabilir (readonly) özellik ekleyerek, dışarıdan yalnızca okuma izni veriyoruz.

using System;

public class Car

{

// Read-only property

public readonly string Model;

public Car(string model)

{

Model = model;

}

public void DisplayModel()

{

Console.WriteLine("Car Model: " + Model);

}

}

class Program

{

static void Main()

{

Car car = new Car("Tesla Model 3");

car.DisplayModel(); // "Tesla Model 3"

// car.Model = "BMW"; // Error: Cannot assign to 'Model' because it is readonly

}

}

**Açıklama**: Model özelliği yalnızca sınıfın yapıcısında (constructor) atanabilir ve dışarıdan değiştirilmesi engellenmiştir.

**Örnek 3: Veri Doğrulama ile Kapsülleme**

Burada, veri doğrulama ile kapsülleme kullanacağız. Kullanıcı ya da dışarıdan gelen veriler doğrulanacak, geçersiz veriler engellenecek.

using System;

public class Account

{

private double balance;

public double Balance

{

get { return balance; }

set

{

if (value >= 0)

{

balance = value;

}

else

{

Console.WriteLine("Balance cannot be negative.");

}

}

}

public void Deposit(double amount)

{

if (amount > 0)

{

Balance += amount;

Console.WriteLine($"Deposited: {amount}, New Balance: {Balance}");

}

else

{

Console.WriteLine("Deposit amount must be positive.");

}

}

public void Withdraw(double amount)

{

if (amount > 0 && amount <= Balance)

{

Balance -= amount;

Console.WriteLine($"Withdrew: {amount}, New Balance: {Balance}");

}

else

{

Console.WriteLine("Invalid withdrawal amount.");

}

}

}

class Program

{

static void Main()

{

Account myAccount = new Account();

myAccount.Deposit(500);

myAccount.Withdraw(200);

myAccount.Balance = -50; // Invalid operation

}

}

**Açıklama**: Balance özelliği, yalnızca sıfır ya da daha büyük bir değere ayarlanabilir. Bu, yanlış veri girişlerinin engellenmesine yardımcı olur. Ayrıca, Deposit ve Withdraw metotları da uygun koşulları kontrol eder.

**Örnek 4: Kapsülleme ile Şifre Yönetimi (Private ve Public Erişim)**

Bu örnekte, bir kullanıcının şifresine erişim sağlarken veri gizliliği sağlanacaktır.

using System;

public class User

{

private string password;

// Constructor to set the password

public User(string password)

{

this.password = password;

}

// Public method to change password

public void ChangePassword(string oldPassword, string newPassword)

{

if (oldPassword == password)

{

password = newPassword;

Console.WriteLine("Password successfully changed.");

}

else

{

Console.WriteLine("Old password is incorrect.");

}

}

// Public method to validate password

public bool ValidatePassword(string inputPassword)

{

return inputPassword == password;

}

}

class Program

{

static void Main()

{

User user = new User("mySecurePass123");

user.ChangePassword("mySecurePass123", "newPass456"); // Correct old password

Console.WriteLine(user.ValidatePassword("newPass456")); // True

user.ChangePassword("wrongOldPass", "anotherNewPass"); // Incorrect old password

}

}

**Açıklama**: Şifre, private olarak saklanır ve dışarıdan doğrudan erişilemez. Şifre değiştirme ve doğrulama işlemleri, kontrollü bir şekilde yapılır. Bu, güvenliği artıran bir yaklaşımdır.

**Sonuç:**

Kapsülleme, nesne yönelimli programlamada önemli bir kavramdır. Bu ilke sayesinde verilerin dış dünyadan korunması ve kontrollü bir şekilde erişilmesi sağlanır. C# dilinde kapsülleme, private, protected, public gibi erişim belirleyicilerle gerçekleştirilir ve veri güvenliğini artırmak için yaygın olarak kullanılır.