POLYMORPHISM

Çok biçimlilik olarak bilinir. Teknik olarak; bir üst sınıf referansının tüm alt sınıf nesnelerini tutabilmesidir. Bilindiği gibi, alt sınıftaki nesneler üst sınıfın sahip olduğu metotlara (static ve final tanımlı metotlar hariç) da sahiptirler (örneğin, oluşturduğunuz bir nesne üzerinde Object sınıfına ait metotları görmeniz mümkündür). Bu özellik ile birlikte Polymorphism, bir üst sınıf referansı ile alt sınıftaki nesnelerin kullanılabilmesine olanak sağlar. Burdaki avantaj, bir işlemi gerçekleştirirken hangi sınıfa ait nesne ile işlem gerçekleştirdiğimizi bilmenize gerek kalmamasıdır. Öncelikle UpCasting olarak adlandırılan işlemi, yani üst sınıf nesnelerini nasıl tuttuğunu inceleyelim:

```
Public class Main {

    Public static void main (String[] args) {

  User user = new PremiumUser();
  // Ya da
  User user2;
  PremiumUser premiumUser = new PremiumUser();
  user2 = premiumUser;
 }

}
```

Gördüğünüz işlem UpCasting olarak adlandırılır. User sınıfı ve bu sınıftan türeyen PremiumUser sınıfı ile bu işlemi gerçekleştirdik. User sınıfından oluşturulan nesne bu referans ile tutuldu. UpCasting, Polymorphism olarak adlandırılan soyut yaklaşımın gerçekleştirilmesini sağlar.

Polymorphism, aynı zamanda nesnelerin birbirlerini tanımadan haberleşmeleri olarak da tanımlanabilir. Bu soyut kavram şeklinde kullanılır.

Static Polymorphism

Bir fonksiyon ile bir objenin, Compile-time da linklenmesine denir. Early bincling olarak da isimlendirilir. Dynamic Polymorphism de ise, linkleme işlemi run-time da yapılır.

Static Polymorphism 2'ye ayrılır:

1. Function Overloading
2. Operator Overloading

Function Overloading

Aynı etki alanı içerisinde, aynı isme sahip birden çok metod oluşturulabilir. Fakat isimleri aynı olsa da, herbirinin tanımlamaları birbirinden farklı olmalı. Bu farklılığı, parametre listesini değiştirerek yapabilirsiniz. Yani her bir metod, overload yapmalı. Aynı parametrelere sahip olup sadece dönüş tipini değiştirerek overload yapamazsınız.

Dynamic Polymorphism

C# abstract class oluşturmamıza imkan tanır. Abstract class'lar abstract metodlar içerebilir ve türetilmiş class'larda içeriği doldurulur. Abstract class konusu böyledir.

REGULAR EXPRESSION

Regular Expression (Regex veya Regexr / Düzenli / Kurallı ifadeler) modern programlama dillerinin neredeyse tamamında yer bulan, aynı söz dizimine (syntax) sahip olan, genellikle harflerden oluşan karakterler dizisinin (katar / string) belirliten kurallar çerçevesinde kısa yoldan ve esnek bir biçimde belirlenmesini sağlayan bir yapıdır.

Çalışma Yapısı

İşlem aşamasında aranan katara (string) bir kalıp tanımı yapılır. (atama). İşleminin ardından bu katarın eşlenikleriyle geri dönmesi beklenir. Bu yapı özel karakterler aracılığıyla sayesinde eşlenik bulma (search) veya yer değiştirme (replace) gibi olaylar hızlı ve etkili bir şekilde gerçekleşir. Eşlenik bulma ve yer değiştirme tanımlamaları için kullanılan özel karakterler

Öncelikle Visual Studio Code için bir pratik örnekle başlayalım. Herhangi bir dosya açalım ve Ctrl+F tuşlarına basarak "bul ve değiştir" alanını görüntüleyelim. Ardından Bul bölümüne şu ifadeyi yapıştıralım. ^.*$, Değiştir bölümünde ise kullanacağımız ifademiz şöyle "$0", Use Regular Expression (.*) seçeneğini seçip dosyamıza uyguladığımızda tüm satırların tırnak işareti içerisine alındığını görebiliriz.

(# Regex örnekler

Match

Verilen string değer içinde, tanımlanan regex ifadesine göre arama yap bulduğu ilk eşleşmeyi döndür.

```
String input = "Regular Expressions kullanımı";
Regex regex = new Regex (@"g*r");
Match m = regex.match (input);
```

Sonuc : m.Value = "gular"

Replace

Bir stringin içinde, diğer bir stringi ya da ifadeyi bulup, istenilen bir string ile ya da diğer regex ifade ile değiştirir.

```
String metin = "Bu metnin içinde " +
               "çok fazla boşluk var";
String Pattern = "\\s+";
string result = Regex.Replace (metin, Pattern, " ");
```

Sonuc: metin = "Bu metin içinde çok fazla boşluk var.