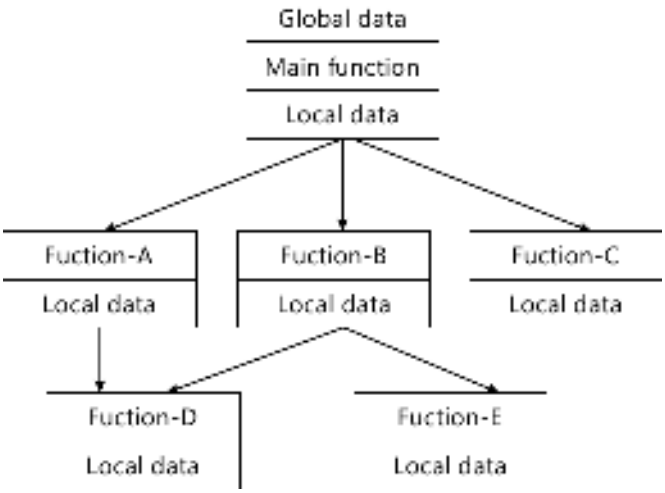
1. Nyp'ye Giriş

NYP temel olarak, ortaya çıktığı güne kadar süregelen programlama mantığını kökten değiştirmiştir. NYP'den önce kullanılan yazılım metodolojisi, **Prosedür Tabanlı Programlama** adı ile anılır. Bu metodoloji, belirli bir yönde ilerleyen kodlar ve iş yükünü hafifletmek için ortak işlerin yüklendiği fonksiyonların çağırılması esasına dayalıydı.

1960'lı yılların sonuna doğru ortaya çıkan bu yaklaşım, o dönemin yazılım dünyasında beliren bir bunalımın sonucudur. Yazılımların karmaşıklığı ve boyutları sürekli artıyor, ancak belli bir nitelik düzeyini korumak için gereken bakımın maliyeti zaman ve çaba olarak daha da hızlı artıyordu.



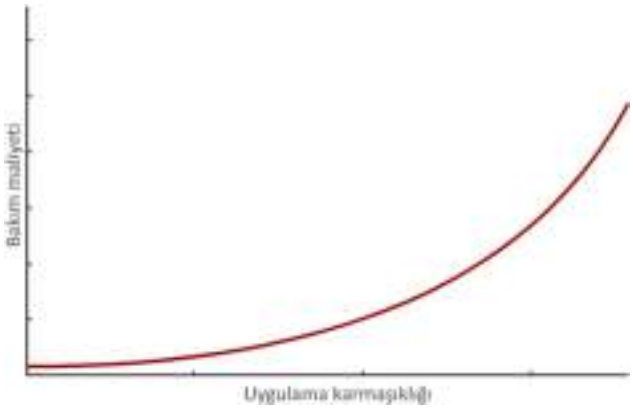
Şekil 1-1. Prosedülel Programlamanın Genel Yapısı

Prosedürel programlamada, geliştirilen uygulama parçalanamayan bir bütün halindeydi. Bu yüzden, uygulama üzerinde çalışan her geliştirici; uygulamanın hemen her yapısına hakim olmalıydı.

Bu durum nedeniyle, projelere yeni yazılımcıların katılması önemli bir adaptasyon süreci gerektiriyordu.

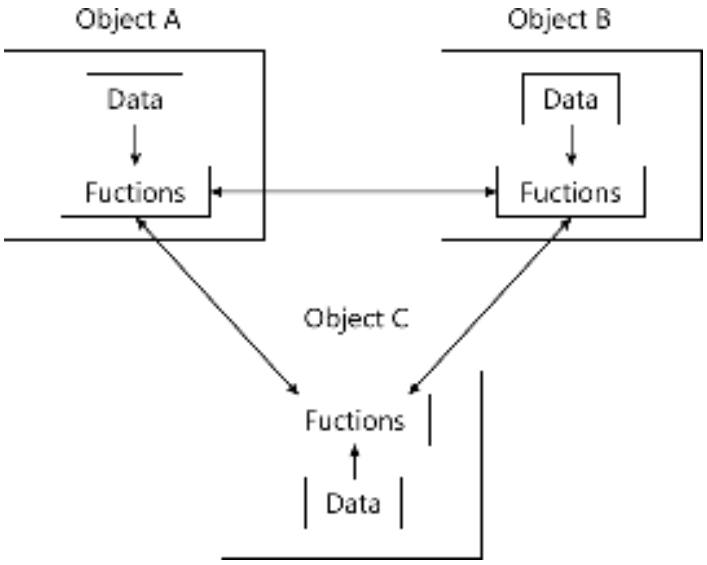
Uygulama tek bir bütün halinde olduğu için, ufak değişiklikler uygulamanın farklı noktalarında büyük sorunlara yol açabiliyordu.

Yıllarla birlikte müşteri ihtiyaçları ve donanım kabiliyetleri arttı. Bunun getirisi olarak da, geliştirilen uygulamaların kapsamları ve boyutları büyüdü. Uygulama maliyetleri giderek artmaya başladı.



Şekil 1-2.Prosedürel programlama bakım maliyeti-uygulama karmaşıklığı

Yazılım dünyasında bu çıkmazın aşılması NYP ile sağlanmıştır.



Şekil 1-3. Nesne Yönelimli Programlamanın Genel Yapısı

NYP'yi bu soruna karşı bir çözüm haline getiren başlıca özelliği, yazılımda birimselliği (modularity) benimsemesidir. NYP ayrıca, bilgi gizleme (information hiding), veri soyutlama (data abstraction), çok biçimlilik (polymorphism) ve kalıtım (inheritance) gibi yazılımın bakımını ve aynı yazılım üzerinde birden fazla kişinin çalışmasını kolaylaştıran kavramları da yazılım literatürüne kazandırmıştır.

Sağladığı bu avantajlardan dolayı, NYP günümüzde geniş çaplı yazılım projelerinde yaygın olarak kullanılmaktadır.

C++, Java, ve C# NYP destekleyen programlama dilleridir. Ancak, bu dilleri prosedürel bir dil gibi kullanmak mümkündür. Bu noktada yazılım geliştiricisinin NYP tasarlama yetisi önem kazanır.

1.1 NYP dillerinin Ortak Özellikleri

- Modülerlik
- Veri ve kontrol soyutlamaları
- Otomatik bellek yönetimi
- Sınıflar
- Kalıtım (Inheritance)
- Çok şekillilik (Polymorphism)

1.1.1 Modülerlik

Yazılım, kendi üzerinde işlemler üzerinde izin veren basit ve tutarlı arayüzlere sahip ayrı modüller halinde geliştirilir.

Farklı görevlerin farklı metotlara ayrıştırılması hatta kimi zaman farklı nesnelere ayrıştırılması anlamına gelir. Her birimin kendi görevi vardır.

Kapsulleme: Kendi içlerinde ilişkili propertyler, metotlar ve diğer öğeler sadece bir nesne ya da birim olarak davranış gösterir.

1.1.2 Otomatik Bellek Yönetimi

Sistem özkaynaklarını kullanan nesneler işleri bitince onları otomatik olarak serbest bırakırlar.

Garbage Collector

Bilgisayar Biliminde garbage collection kaynak yönetiminin özel bir adımıdır. Bilgisayar hafızasının yönetilmesi işleminde kullanılmaktadır.

Gabrage Collector'dan önce bilgisayar programları çalışma zamanı sırasında bellek ihtiyacı duyarlar ve ihtiyaç duyulmayan hafıza alanları programlar tarafından işletim sistemine iade edilir. Bu işlem C dilinde malloc() ve free() fonksiyonları ile yapılmaktadır. Belirli durumlarda programların akışında bu iki fonksiyonun izini sürmek zorlaşır. İşletim sistemine iade edilmiş bir alanın program tarafından tekrar kullanılması güvenlik açıklarını beraberinde getirmektedir.

Programlama yaparken hafıza yönetimini otomatiğe bağlamak geliştiricilerin işini kolaylaştırmıştır. Böylelikle bir programcı program yazarken hafıza alanını garbage collector

sisteminden talep edip, iade işlemine karışılmamaktadır. Kalan tüm işlemler garbage collection mekanizması tarafından yapılmaktadır.

Garbage collection kullanılan hafıza alanlarının izini sürerek ihtiyaç duyulmayan alanları işletim sistemine iade etmektedir. Yeni bir talep geldiği zamanda yer bulup programa göndermektedir.

Garbage collection çalışma zamanında hafıza alanlarını yönetmektedir. Bu da bu sistemin işlemci kullanmasına sebep olmaktadır. İşlemci kullanımı ile alakalı iki farklı yönetim yaklaşımı bulunmaktadır.

- Birincisi stop-the-world yaklaşımıdır. Herhangi bir t zamanında garbage collector devreye girdiği zaman ilgili programdaki tüm işlemler durdurulur, hafıza yönetimi yapılır, daha sonra işlemler başlatılır. Bu çalışma zamanı kritik uygulamalar için bir dezavantaj oluşturmaktadır.
- Bu nedenle daha sonra concurrent algoritmalar geliştirilmiştir. Concurrent yaklaşımda garbage collection ile ilgili uygulama eşzamanlı olarak çalışmaktadır.

1.1.3 Soyutlama

Soyutlama denilince, nesneyi bazı karakteristikleri olan ve bazı eylemleri gerçekleştirebilen bir veri tipi olarak genelleştirmek anlaşılmalıdır. Yeryüzü üzerinde bulunan her şey bir nesnedir. Bilgisayar yazılımı açısından nesne, bir varlığın temsil biçimidir. Örneğin; bina bir nesnedir. Binayı bilgisayar yazılımı içinde temsil etmek istersek, öncelikle bu nesnenin yani binanın ayırt edici özelliklerini belirlemeliyiz.

Örneğin;

Bina yüksekliği
Dış yüzey rengi
Kat sayısı
Zemin alanı
Zemin boyutları

gibi özellikler, binayı temsil etmek için ilk akla gelen birkaç özelliktir. O halde nesnenin bilgisayarda temsilinde, özellikleri kilit rol oynamaktadır.

“Soyutlama” önemli özelliklere daha etkin kullanabilmek için ayrıntıları göz ardı etme mantığına dayanır.

Bir nesnenin karakteristikleri “özellik”, eylemleri ise metot olarak ifade edilir. Bu anlamda soyutlaştırma sonucunda bir nesne; özellikleri (properties) ve metotları (methods) ile temsil edilir.

1.1.4 Sınıf

Nesne Yönelimli Programlama Dilleri, soyutlamayı sınıf (Class) yapısı ile gerçekleştirirler. Sınıf yapısı içinde o nesneye ait özellikler ve metotlar tanımlanabilir.

Ancak sınıf soyut bir yapıdır ve doğrudan kullanılamaz. O sınıftan üretilen örnekler sınıfa ait tüm özelliklere ve metotlara sahip olurlar. Bunlar program içinde doğrudan kullanılabilirler. Sınıftan üretilen her örnek, aynı özellik ve metotlara sahip olacak ancak özelliklerin değerleri doğal olarak farklı olabilecektir.

Örneğin, insanı bir nesne olarak düşünürsek ve insan sınıfı olarak soyutlarsak, ilk akla gelen özellikleri; boy, kilo, IQ ve EQ değerleri, eğitim düzeyi vb. gibi özellikler olur. Bu özelliklerin değerleri doğal olarak her insan için farklıdır. İnsan için ilk akla gelen eylemler (metotlar) ise; yürüme, okuma-yazma, konuşma, araç kullanma vb. gibi metotlardır.

1.1.5 Veri ve Kontrol Soyutlamaları

Nesne tanımlanırken verinin kullanılması veya veriye erişilmesi için gerekli detayların azaltılması işlemidir.

Soyutlama kabaca, iki şekilde yapılır.

- Veri (data) Soyutlaması: veri yapıları üzerinde yapılan soyutlamadır.
- Kontrol (control) Soyutlaması: yapısal programlama ile gelen altprogram (subprogram), işlev (function) gibi kavramlar üzerinde yapılan soyutlamadır.

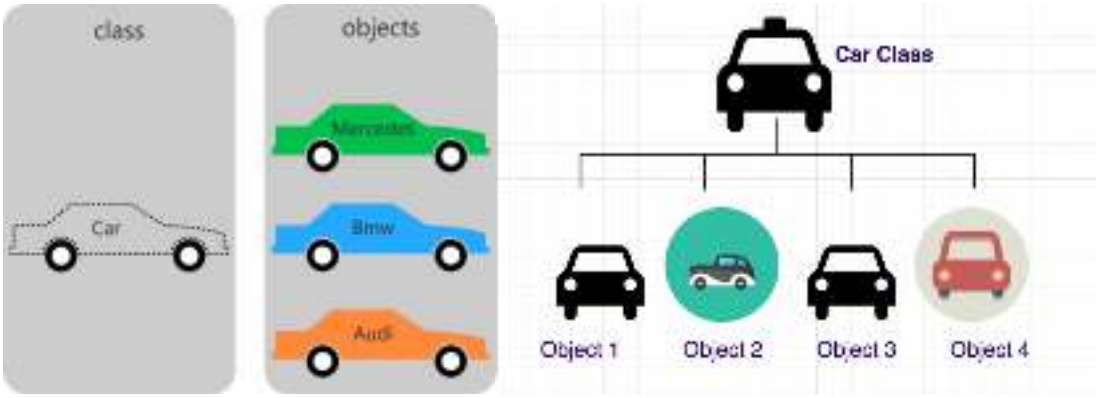
Örnek olarak bir "kişi" nesnesinde, kişinin yaşını tutan bir tamsayı değişkeni olan "yas" değişkenini ele alalım. Bu "yas" değişkenine, programın çalıştırma anında, "-10" gibi bir değer atanmasını kimse engelleyemez.

İşte burada Nesne Yönelimli Programlamanın getirdiği görünürlük (visibility), ve özellik (property) tanımlama gibi yetenekler kullanarak "yas" değerine gerçek bir değerin girilmesi garanti edilir. İşte buna “Veri Soyutlaması” adı verilir.

Elimizde yaş kontrolü diye bir fonksiyon olsun. Bu kontrol içinde negatif sayı kontrolü yapabiliriz. Bu işleme ise “kontrol soyutlama” işlemi denir.

1.1.6 Nesne (Object)

Sınıfları (class) kullanarak değişken yaratma işlemidir.



1.1.7 Kalıtım - Miras Alma (Inheritance)

Nesneye yönelik programlamada, bir nesne, genellikle bir nesne sınıfına ait bir örnektir (instance).

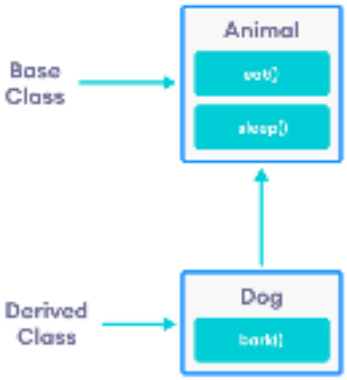
Örneğin, Albert Einstein, insan sınıfının bir örneğidir. Bir nesne sınıfından alt sınıflar (subclasses) oluşturulabiliyorsa, türetme özelliği (derivation) var demektir. Örneğin insan sınıfı, canlı sınıfının bir alt sınıfıdır.

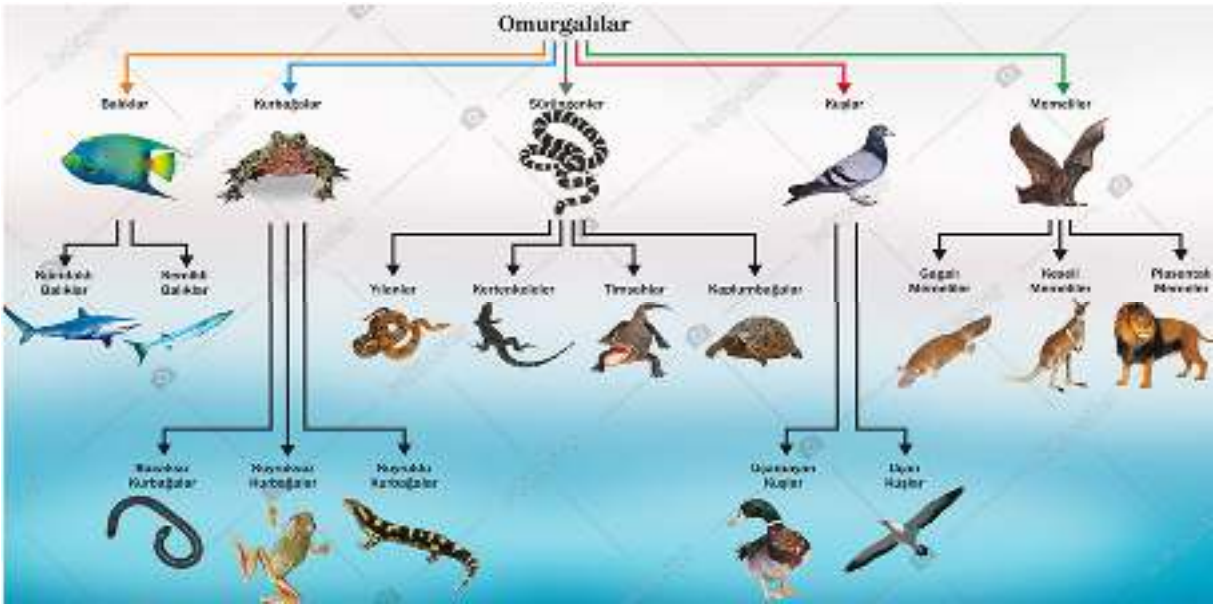
Kendisinden alt sınıf üretilen sınıfa, temel sınıf (base class) veya süper sınıf (super class) veya ana sınıf (parent class) adı verilir.

Subclass yerine child class terimi de kullanılmaktadır.

Alt sınıfın nesneleri, türetildikleri temel sınıfa ait özellikleri alıyorsa, burada miras alma (inheritance) özelliği vardır denir.

Bu anlamda, miras alma özellikli bir nesne yönelimli programlama dilinde, bir nesne sınıfından türetilen alt nesne sınıfına ait nesneler, üst sınıfın özelliklerini (properties) ve metodlarını (methods) aynen alırlar.





1.1.8 Çok Şekillilik (Polymorphism)

Farklı nesnelerin, aynı mesaja (olaya ya da uyarıma) farklı şekillerde cevap verebilme yeteneğidir.

Her nesne sınıfı, kendi metotlarını paketlediği için ve bu metotlar programın kalan kısmı için gizli olduğundan, farklı sınıflar aynı isimde bazı metotlara sahip olabilirler.

Örneğin ŞEKİL adlı bir süper sınıfımız (super class) olsun. Bu sınıftan, DAİRE, KARE ve ÜÇGEN adlı 3 alt sınıf türettiğimizi varsayalım. Bu alt sınıfların her biri kendi örneklerini çizmek için ÇİZ adlı bir metoda sahip olabilir. Fakat her bir alt sınıf için bu metot aynı olmasına karşın, bu metodu devreye sokmak için gönderilecek mesajdan sonra her alt sınıfa ait nesne farklı bir şekil (yani kendini, yani bir üçgen, daire veya kare) çizecektir.

Nesne Yönelimli Programlamanın sınıf türetebilme özelliği sayesinde, bir nesne hiyerarşisi oluşturma imkânı bulunmaktadır.

Örnek:

```
Sınıf: Dikdörtgen
     Veri1: En
     Veri2: Boy
     Fonksiyon: Alan_Hesapla = En*Boy
     Kontrol: En verisi
     Kontrol: Boy verisi
Uygulama Main Class
     dikdörtgen t = new dikdörtgen(5, 3);
     t.Alan_Hesapla;
     dikdörtgen t2 = new dikdörtgen(7, 8);
     t2.Alan_Hesapla;
```