İnsana konuşma yeteneği verildi ve konuşma, evrenin ölçüsü olan düşünceyi yarattı.

" İnsanlar, toplumları için ifade aracı haline gelmiş belirli bir dilin çokça merhametine bağlıdırlar.

Gerçekliğe esasen dil kullanmaksızın uyum sağladığımız ve dilin sadece belirli iletişim ve yansıma problemlerini çözmenin yanı sıra bir araç olduğu düşüncesi tamamen bir yanılsamadır.

Aslında, "gerçek dünya" büyük ölçüde grubun dil alışkanlıkları üzerine bilinçsizce inşa edilir.

Nesnelere Giriş

Doğayı parçalara ayırırız, kavramlar halinde organize ederiz ve belirli anlamlar atfederiz, büyük ölçüde konuşma topluluğumuz arasında var olan ve dilimizin kalıplarında kodlanmış bir anlaşmaya taraf olduğumuz için... anlaşmanın emrettiği veri organizasyonuna ve sınıflandırmasına abone olmadan hiç konuşamayız."

Nesne yönelimli programlama (OOP), bilgisayarı bir ifade ortamı olarak kullanma yönündeki bu hareketin bir parçasıdır.

Birçok kişi, büyük resmi öncelikle anlamadan nesne yönelimli programlamaya giriş yapmak konusunda rahat hissetmez. Bu nedenle, OOP'ye sağlam bir genel bakış sağlamak için burada birçok kavram tanıtılıyor.

- Assembly dili, temel makinenin küçük bir soyutlamasıdır. İzleyen birçok “zorunlu” dil (FORTRAN, BASIC ve C gibi) assembly dilinin soyutlamalarıydı. Bu diller, assembly diline göre büyük iyileştirmeler olmasına rağmen, temel soyutlamaları hala bilgisayarın yapısını düşünmenizi gerektiriyor, çözmeye çalıştığınız sorunun yapısından ziyade. Programcının, çözümün uygulandığı yer olan “çözüm alanındaki” makine modeli ile aslında çözülen sorunun modeli arasında bir ilişki kurması gerekir (örneğin, bir işletme gibi sorunun var olduğu yer olan “sorun alanı”). Bu eşleştirmeyi gerçekleştirmek için gereken çaba ve bunun programlama diline dışsal olması, yazılması zor ve bakımı pahalı programlar üretir ve yan etki olarak tüm “programlama yöntemleri” endüstrisini yaratır.

- Makineyi modellemenin alternatifi, çözmeye çalıştığınız sorunu modellemektir. LISP ve APL gibi erken diller, dünyanın belirli görüşlerini seçti (“Tüm sorunlar sonunda listelerdir” veya “Tüm sorunlar algoritmiktir”). Prolog, tüm sorunları karar zincirlerine döker. Kısıtlama tabanlı programlama için ve yalnızca grafik sembollerle manipüle ederek programlama için diller oluşturulmuştur. (İkincisi çok kısıtlayıcı olduğunu kanıtladı.) Bu yaklaşımların her biri, çözmeye tasarlandıkları belirli sorun sınıfı için iyi bir çözüm olabilir, ancak bu alanın dışına çıktığınızda garip hale gelirler.

- Nesne yönelimli yaklaşım, programcının sorun alanındaki öğeleri temsil etmek için araçlar sağlayarak bir adım daha ileri gider. Bu temsil, programcının herhangi bir belirli türdeki soruna kısıtlanmadığı kadar geneldir. Sorun alanındaki öğelere ve çözüm alanındaki temsillerine “nesneler” olarak atıfta bulunuyoruz. (Sorun alanı analogları olmayan diğer nesnelere de ihtiyacınız olacak.) Fikir şu ki, program, çözümü açıklayan kodu okuduğunuzda, sorunu da ifade eden kelimeleri okuyorsunuz, böylece yeni türde nesneler ekleyerek sorunun lingo'suna kendini uyarlamasına izin verilir. Bu, daha önce sahip olduğumuzdan daha esnek ve güçlü bir dil soyutlamasıdır.1 Böylece, OOP, sorunu çözümün çalıştırılacağı bilgisayarın şartları yerine, sorunun şartları açısından tanımlamanıza olanak tanır. Bilgisayara geri dönüş bağlantısı hala var: Her nesne küçük bir bilgisayara oldukça benziyor - bir durumu var ve gerçekleştirmesini isteyebileceğiniz işlemleri var. Ancak, bu, gerçek dünyadaki nesnelere o kadar da kötü bir benzetme gibi görünmüyor - hepsinin karakteristikleri ve davranışları var.

- **1.** Her şey bir nesnedir. Bir nesneyi süslü bir değişken olarak düşünün; veri depolar, ancak bu nesneye "taleplerde bulunabilir", kendisi üzerinde işlemler gerçekleştirmesini isteyebilirsiniz. Teorik olarak, çözmeye çalıştığınız problemin herhangi bir kavramsal bileşenini (köpekler, binalar, hizmetler vb.) programınızda bir nesne olarak temsil edebilirsiniz.

- **2.** Bir program, nesnelerin birbirlerine ne yapacaklarını söyleyerek mesajlar gönderdiği bir sürü nesnedir. Bir nesneye talepte bulunmak için, o nesneye "bir mesaj gönderirsiniz". Daha somut bir şekilde, bir mesajı, belirli bir nesneye ait bir yöntemi çağırma isteği olarak düşünebilirsiniz.

- **3.** Her nesnenin, diğer nesnelerden oluşan kendi hafızası vardır. Başka bir deyişle, mevcut nesneleri içeren bir paket yaparak yeni bir tür nesne oluşturursunuz. Böylece, bir programın karmaşıklığını nesnelerin basitliği arkasına saklayarak inşa edebilirsiniz.

- **4.** Her nesnenin bir türü vardır. Deyimi kullanarak, her nesne bir sınıfın örneğidir, burada "sınıf" "tür" ile eşanlamlıdır. Bir sınıfın en önemli ayırt edici özelliği "Ona hangi mesajları gönderebilirsiniz?" dir.

- **5.** Belirli bir türdeki tüm nesneler aynı mesajları alabilir. Bu aslında daha sonra göreceğiniz gibi yüklü bir ifadedir. “daire” türünde bir nesne aynı zamanda “şekil” türünde bir nesne olduğu için, bir daire şekil mesajlarını kabul etmeyi garanti eder. Bu, şekillerle konuşan kod yazabileceğiniz ve otomatik olarak bir şeklin tanımına uyan her şeyi ele alabileceğiniz anlamına gelir. Bu yer değiştirme, OOP'deki güçlü kavramlardan biridir.

- Nesneye yönelik programlama (OOP) paradigması, yazılımın nasıl düşünüldüğü ve yapılandırıldığı konusunda devrim yaratan bir yaklaşım sunar. Bu bölümde belirtilen konseptler, OOP'nin temelini oluşturan unsurlardır ve yazılım tasarımı ve geliştirmesindeki kararları derinden etkiler.

- **1. **Nesnenin Arayüzü Var**:** Her nesnenin bir "arayüzü" vardır, yani dış dünya ile etkileşimde bulunmak için kullandığı yöntemler ve özellikler seti. Arayüz, nesnenin nasıl kullanılacağını belirtir ancak iç yapısını (kapsülleme yoluyla gizlenmiş) açığa çıkarmaz. Bu, nesneler arası etkileşimi düzenler ve belirli bir uygulama için gerekli işlevselliği sağlayan sınıfların tanımlanmasına olanak tanır.

-

- **2. **Sınıf ve Tür Kavramları**:** Aristotle'nin türlerin sınıflandırılması üzerine yaptığı erken çalışmalar, Simula-67 gibi ilk nesne yönelimli dillerde doğrudan kullanılmıştır. Bir "sınıf", benzer özellikler ve davranışlar paylaşan nesnelerin bir gruplandırmasıdır; bu nedenle, bir sınıf aslında bir veri türüdür. Yüzer noktalı bir sayı gibi, belirli özelliklere (veri) ve davranışlara (fonksiyonlar) sahiptir.

- **3. **Abstract Data Types ve Sınıflar**:** OOP'de, yeni veri türleri yaratırız, bu genellikle "abstract data types" veya sınıflar yoluyla yapılır. Bu sınıflar, tıpkı yerleşik türler gibi çalışır; nesneler (veya örnekler) oluşturabilir ve bu nesneler üzerinde işlemler gerçekleştirebilirsiniz. Her sınıfın üyeleri ortak özellikleri paylaşır, ancak her birinin kendi benzersiz durumu vardır.
-
- **4. **Sınıfların Dil Uzantısı Olarak Kullanımı**:** Programcılar, bir problemi çözmek için bir sınıfı tanımlarlar, bu da onların mevcut bir veri türünü kullanmaya zorlanmaları yerine, programlama dilini kendi ihtiyaçlarına özgü yeni veri türleri ekleyerek genişletmelerine olanak tanır. Programlama sistemi, yeni sınıfları memnuniyetle karşılar ve yerleşik türlere verdiği özeni ve tip kontrolünü sağlar.

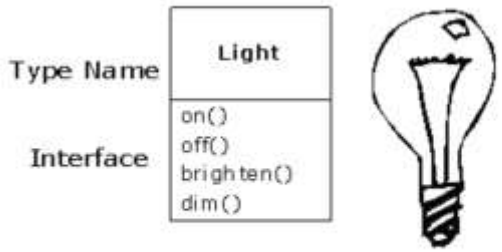
- **5. **Nesnelerin Çok Yönlülüğü ve Kimlikleri**:** Nesneler sadece bir bellek adresi ile tanımlanmaz. Farklı makinelerde, farklı adres alanlarında var olabilirler veya diske kaydedilebilirler. Bu durumlarda, bir nesnenin kimliği, bellek adresinden başka bir şey tarafından belirlenmelidir.

- **6. **Sınıf ve Tür Arasındaki Ayrım**:** Bazıları, 'tür'ün arayüzü belirlediğini, 'sınıf'ın ise o arayüzün belirli bir uygulaması olduğunu söyleyerek bir ayrım yapar. Bu, farklı sınıfların aynı arayüzü uygulayabileceği ve dolayısıyla farklı nesnelerin aynı mesajlara (metot çağrılarına) yanıt verebileceği anlamına gelir.

- OOP'nin güzelliği, her türlü uygulama için kullanılabilir olmasıdır; sadece simülasyonlarla sınırlı değildir. OOP tekniklerini kullanarak, karmaşık problemler setini basit, yönetilebilir çözümlere indirgeyebiliriz. Bu, yazılım geliştirme sürecini daha esnek ve genişletilebilir hale getirir, çünkü yeni sınıflar ve nesneler eklemek, mevcut kod üzerinde önemli değişiklikler yapmadan yapılabilir.

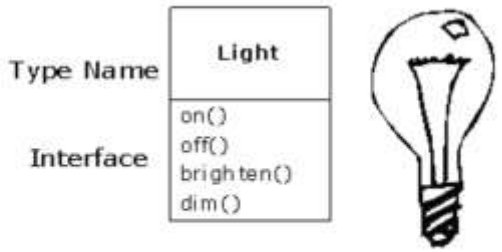
- Bir sınıf oluşturulduğunda, istediğiniz kadar çok nesne yapabilir ve sonra bu nesneleri çözmeye çalıştığınız problemde var olan öğelermiş gibi manipüle edebilirsiniz. Aslında, nesne yönelimli programlamanın zorluklarından biri, problem alanındaki öğeler ile çözüm alanındaki nesneler arasında bire bir eşleme oluşturmaktır.

- Ancak, bir nesneyi sizin için faydalı bir iş yapmaya nasıl ikna edersiniz? Nesnenin bir şey yapması, örneğin bir işlemi tamamlaması, ekranda bir şey çizmesi veya bir anahtarı açması için bir istekte bulunmanın bir yolu olmalı. Ve her nesne sadece belirli istekleri karşılayabilir. Bir nesneye yapabileceğiniz istekler, onun arayüzü tarafından tanımlanır ve arayüzü belirleyen şey türdür. Basit bir örnek, bir ampulün temsili olabilir.
- Arayüz, belirli bir nesne için yapabileceğiniz istekleri belirler. Ancak, bu isteği karşılamak için bir yerde kod olmalıdır. Bu, gizli verilerle birlikte uygulamayı içerir. Prosedürel programlama açısından, o kadar da karmaşık değil. Bir türün her olası istekle ilişkili bir yöntemi vardır ve bir nesneye belirli bir istek yaptığınızda bu yöntem çağrılır. Bu süreç genellikle bir nesneye "mesaj gönderdiğiniz" (bir istek yaptığınız) ve nesnenin o mesajla ne yapacağını bulduğu (kod yürütür) denilerek özetlenir.




```
Light lt = new Light();
lt.on();
```

- **Burada, türün/sınıfın adı Light'tır, bu belirli Light nesnesinin adı lt'dir ve bir Light nesnesinden yapabileceğiniz istekler onu açmak, kapatmak, daha parlak hale getirmek veya yapmaktır. karartıcı. Bu nesne için bir "referans" (lt) tanımlayarak ve o türden yeni bir nesne istemek için new öğesini çağırarak bir Light nesnesi yaratırsınız. Nesneye mesaj göndermek için nesnenin adını belirtir ve onu nokta (nokta) ile mesaj isteğine bağlarsınız. Önceden tanımlanmış bir sınıfın kullanıcısı açısından, nesnelerle programlamanın hemen hemen hepsi bu kadar. Yukarıdaki şema, Birleşik Modelleme Dilinin (UML) biçimini takip eder. Her sınıf, kutunun üst kısmında tür adı, kutunun orta kısmında açıklamayı düşündüğünüz veri üyeleri ve yöntemler (bu nesneye ait işlevler, o nesneye gönderdiğiniz tüm mesajlar) kutunun alt kısmında. Genellikle, UML tasarım diyagramlarında yalnızca sınıfın adı ve genel yöntemler gösterilir, bu nedenle bu durumda olduğu gibi orta kısım gösterilmez. Yalnızca sınıf adıyla ilgileniyorsanız, alt bölümün de gösterilmesine gerek yoktur.**




```
Light lt = new Light();
lt.on();
```

- **Sınıflar**

- Nesne tabanlı programlamanın temelinde, yukarıdaki giriş bölümünde de adını andığımız ‘sınıf’ (class) adlı bir kavram bulunur. sınıflar, nesne üretmemizi sağlayan veri tipleridir. İşte nesne tabanlı programlama, adından da anlaşılacağı gibi, nesneler (ve dolayısıyla sınıflar) temel alınarak gerçekleştirilen bir programlama faaliyetidir.

- Bir kişi işe alındığında, o kişiye dair belli birtakım bilgileri bu veritabanına işliyorsunuz. Mesela işe alınan kişinin adı, soyadı, unvanı, maaşı ve buna benzer başka bilgiler.

```
class Çalışan:

    pass
```

- Yukarıdaki, boş bir sınıf tanımıdır. Hatırlarsanız fonksiyonları tanımlamak için def adlı bir ifadeden yararlanıyorduk. İşte sınıfları tanımlamak için de class adlı bir ifadeden yararlanıyoruz. Bu ifadenin ardından gelen Çalışan kelimesi ise bu sınıfın adıdır.
-
- Eğer arzu ederseniz, yukarıdaki sınıfı şu şekilde de tanımlayabilirsiniz:
-

```
class Çalışan():
    pass
```

- fonksiyonlarda olduğu gibi, bir sınıfı kullanabilmek için de öncelikle o sınıfı tanımlamamız gerekiyor

- **Sınıf Nitelikleri**
- class Çalışan():
  - kabiliyetleri = []
  - unvanı = 'işçi'
-
- Burada unvanı ve kabiliyetleri adlı iki değişken tanımladık. Teknik dilde bu değişkenlere ‘sınıf niteliği’ (class attribute) adı verilir.
-

- class Çalışan():
  - kabiliyetleri = []
  - unvanı = 'işçi'
  - maaşı = 1500
  - memleketi = ''
  - doğum_tarihi = ''
- ‘Çalışan’ adlı bir grubun ortak niteliklerini belirledik. Elbette her çalışanın memleketi ve doğum tarihi farklı olacağı için sınıf içinde bu değişkenlere belli bir değer atamadık. Bunların birer karakter dizisi olacağını belirten bir işaret olması için yalnızca memleketi ve doğum_tarihi adlı birer boş karakter dizisi tanımladık.
- Sınıf niteliklerine, doğrudan sınıf adını kullanarak erişebilmek mümkündür.

```
print(Çalışan.maaşı)

print(Çalışan.memleketi)

print(Çalışan.doğum_tarihi)
```

bu sınıfa yeni sınıf nitelikleri de ekleyebilirsiniz

```
Çalışan.isim = 'Ahmet'

Çalışan.yaş = 40
```

Sınıfların Örneklenmesi

```
class Çalışan():
        kabiliyetleri = []
        unvanı = 'işçi'
        maaşı = 1500
        memleketi = ''
        doğum_tarihi = ''

ahmet = Çalışan()  // sınıfımızı ahmet adlı bir değişkene atadık
```

Başka bir örnek

```
class Sipariş():
        firma = ''
        miktar = 0
        sipariş_tarihi = ''
        teslim_tarihi = ''
        stok_adedi = 0
jilet = Sipariş()
```

Burada class, sınıfı tanımlamamıza yarayan bir öğedir. Tıpkı fonksiyonlardaki def gibi, sınıfları tanımlamak için de class adlı bir parçacığı kullanıyoruz. Sipariş ise, sınıfımızın adı oluyor. Biz sınıfımızın adını parantezli veya parantezsiz olarak kullanma imkanına sahibiz.

Sınıfın gövdesinde tanımladığımız şu değişkenler birer sınıf niteliğidir (class attribute):

```
firma = ''
miktar = 0
sipariş_tarihi = ''
teslim_tarihi = ''
stok_adedi = 0
```

jilet = Sipariş() komutunu verdiğimizde ise, biraz önce tanımladığımız sınıfı örnekleyip (instantiation), bunu jilet adlı bir örneğe (instance) atamış oluyoruz. Yani jilet, Sipariş() adlı sınıfın bir örneği olmuş oluyor. Bir sınıftan istediğimiz sayıda örnek çıkarabiliriz:

kalem = Sipariş()
pergel = Sipariş()
çikolata = Sipariş()
Bu şekilde Sipariş() sınıfını üç kez örneklemiş, yani bu sınıfın bütün özelliklerini taşıyan üç farklı üye meydana getirmiş oluyoruz.

Bu sınıf örneklerini kullanarak, ilgili sınıfın niteliklerine (attribute) erişebiliriz:

```
kalem = Sipariş()
kalem.firma
kalem.miktar
kalem.sipariş_tarihi
kalem.teslim_tarihi
kalem.stok_adedi
```

eriştiğimiz bu nitelikler birer sınıf niteliği olduğu için, sınıfı hiç örneklemeden, bu niteliklere doğrudan sınıf adı üzerinden de erişebilirdik:

Sipariş.firma
Sipariş.miktar
Sipariş.sipariş_tarihi
Sipariş.teslim_tarihi
Sipariş.stok_adedi

```
class Çalışan():
        kabiliyetleri = []
        unvanı = 'işçi'
        maaşı = 1500
        memleketi = ''
        doğum_tarihi = ''
```

Burada kabiliyetleri, unvanı, maaşı, memleketi ve doğum_tarihi adlı beş adet değişken tanımladık. Teknik dilde bu değişkenlere ‘sınıf niteliği’ (class attribute) adı verildiğini biliyorsunuz.
Çalışan() sınıfı içindeki niteliklere erişmek için birkaç tane örnek çıkaralım:

```
ahmet = Çalışan()
mehmet = Çalışan()
ayşe = Çalışan()
```

Bu şekilde Çalışan() sınıfının üç farklı örneğini oluşturmuş olduk. Bu sınıfın niteliklerine, oluşturduğumuz bu örnekler üzerinden erişebiliriz:

```
print(ahmet.kabiliyetleri)
print(ahmet.unvanı)
print(mehmet.maaşı)
print(mehmet.memleketi)
print(ayşe.kabiliyetleri)
print(ayşe.doğum_tarihi)
```

Çıkardığımız örnekler aracılığıyla sınıf nitelikleri üzerinde değişiklik de yapabiliyoruz:

```
ahmet.kabiliyetleri.append('prezantabl')
```