İleri Programlama

Nesne Yönelimli Programlamanın Temelleri
(Objective Oriented Programming Basics)

Hüseyin Ahmetoğlu

Neden OOP?

- Kendi bilgisayarınızı bir araya getirmek istediğinizi varsayalım, bir hırdavatçıya gidip bir anakart, bir işlemci, biraz RAM, bir sabit disk, bir kasa, bir güç kaynağı alıp bir araya getirdiniz. Gücü açarsınız PC çalışır. CPU'nun 1 çekirdekli mi yoksa 6 çekirdekli mi olduğu konusunda endişelenmenize gerek yok; anakart 4 katmanlı veya 6 katlıdır; sabit disk, 3 inç veya 5 inç çapında 4 plaka veya 6 plakaya sahiptir; RAM Japonya veya Kore'de üretilir ve bu böyle devam eder. Donanım bileşenlerini bir araya getirir ve makinenin çalışmasını beklersiniz. Elbette, doğru arabirimlere sahip olduğunuzdan emin olmalısınız, yani, anakartınız yalnızca IDE'yi destekliyorsa, bir SCSI sabit disk yerine bir IDE sabit disk seçmelisiniz; doğru hız oranına sahip RAM'leri seçmeniz gerekir, vb. Bununla birlikte, bir makineyi donanım bileşenlerinden kurmak zor değildir.

Neden OOP?

- Benzer şekilde, bir araba şasi, kapılar, motor, tekerlekler, fren ve şanzıman gibi parça ve bileşenlerden monte edilir. Bileşenler yeniden kullanılabilir, örneğin bir tekerlek birçok otomobilde kullanılabilir (aynı özelliklere sahip).
- Bilgisayarlar ve arabalar gibi donanımlar, yeniden kullanılabilir donanım bileşenleri olan parçalardan bir araya getirilir.

Neden OOP?

- Peki Yazılım? Buradan bir program parçası, orada bir program parçası seçerek bir yazılım uygulamasını "birleştirebilir" ve programın çalışmasını bekleyebilir misiniz?
- Donanımın aksine, bir uygulamayı yazılım bileşenlerinden "birleştirmek" çok zordur. 70 yıl önce bilgisayarın ortaya çıkışından bu yana, milyonlarca program yazdık. Ancak her yeni uygulama için tekerlekleri yeniden icat etmemiz ve programı sıfırdan yazmamız gerekiyor!

Geleneksel Prosedür Odaklı diller

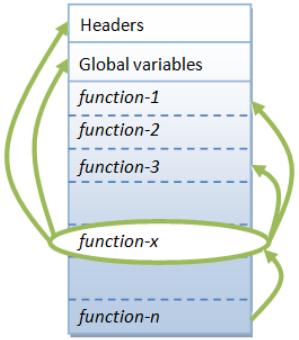
- Geleneksel prosedür odaklı programlama dilleri (C, Fortran, Cobol ve Pascal gibi), yeniden kullanılabilir yazılım bileşenleri oluşturmada bazı önemli dezavantajlardan muzdariptir:

  Prosedüre yönelik programlar, işlevlerden oluşur. İşlevler daha az yeniden kullanılabilir. Bir fonksiyonun bir programdan kopyalanması ve başka bir programda yeniden kullanılması çok zordur, çünkü fonksiyonun genel değişkenlere ve diğer fonksiyonlara gönderme yapması muhtemeldir. Diğer bir deyişle, işlevler kendi kendine yeten yeniden kullanılabilir bir birim olarak iyi bir şekilde kapsüllenmemiştir.

Geleneksel Prosedür Odaklı diller

- Prosedürel diller, gerçek hayattaki problemleri çözmek için yüksek düzeyde soyutlamaya uygun değildir. Örneğin, C programları, düşük seviyeli ve Müşteri İlişkileri Yönetimi (CRM) sistemi veya bir bilgisayar futbol oyunu gibi gerçek sorunları soyutlaması zor olan if-else, for-loop, array, method, pointer gibi yapıları kullanır.

  Geleneksel prosedür dilleri, veri yapılarını (değişkenleri) ve algoritmaları (fonksiyonları) ayırır.



A function (in C) is not well-encapsulated

Geleneksel Prosedür Odaklı diller

- 1970'lerin başında, ABD Savunma Bakanlığı (DoD), BT bütçesinin neden kontrolden çıktığını araştırmak için bir ekip görevlendirdi;

  Bütçenin% 80'i yazılıma ayrılmış (kalan% 20'si donanıma).
  Yazılım bütçesinin% 80'inden fazlası bakıma ayrılmış (sadece kalan% 20 yeni yazılım geliştirme için).
  Donanım bileşenleri çeşitli ürünlere uygulanabilir ve bütünlükleri normalde diğer ürünleri etkilemez. (Donanım paylaşabilir ve yeniden kullanabilir! Donanım hataları izole edilmiştir!)
  Yazılım prosedürleri genellikle paylaşılamaz ve tekrar kullanılamaz. Yazılım hataları, bilgisayarlarda çalışan diğer programları etkileyebilir.

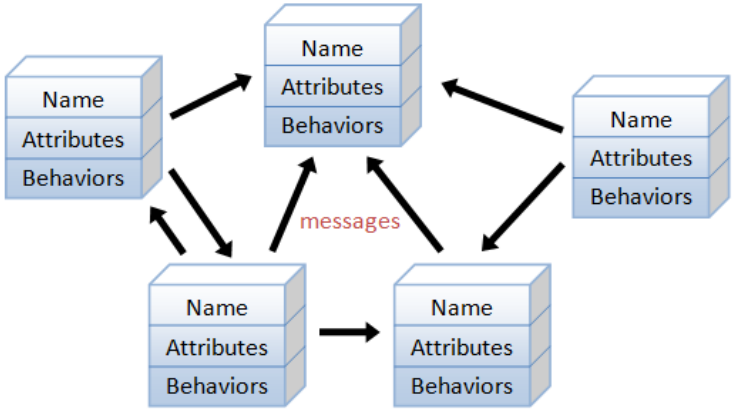
  Ekip, yazılımın donanım OBJECT gibi davranmasını önerdi. Daha sonra DoD, sistemleri oluşturmak için kullanılan 450'den fazla bilgisayar dilini Ada adı verilen nesne yönelimli bir dil ile değiştirdi.

Object-Oriented Programming Languages

- OOP'nin temel birimi, hem statik özellikleri hem de dinamik işlemleri bir "kutu" içinde kapsayan ve bu kutuları kullanmak için genel arabirimi belirleyen bir sınıftır. Sınıflar iyi bir şekilde kapsüllendiğinden, bu sınıfları yeniden kullanmak daha kolaydır. Başka bir deyişle, OOP, bir yazılım varlığının veri yapılarını ve algoritmalarını aynı kutu içinde birleştirir.

- OOP dilleri, gerçek hayattaki sorunları çözmek için daha yüksek düzeyde soyutlamaya izin verir. Geleneksel prosedür dili (C ve Pascal gibi), çözmeye çalıştığınız problem açısından düşünmek yerine, sizi bilgisayarın yapısı (örneğin, bellek bitleri ve baytlar, dizi, karar, döngü) açısından düşünmeye zorlar. OOP dilleri (Java, C ++ ve C # gibi) problem alanında düşünmenize ve problemi çözmek için problem alanının varlıklarını temsil etmek ve soyutlamak için yazılım nesnelerini kullanmanıza izin verir.

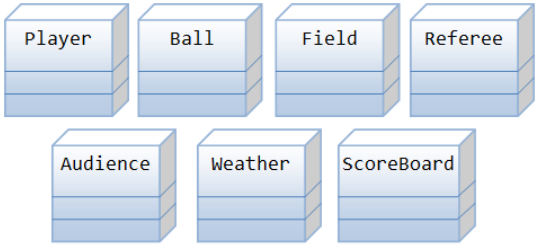
Object-Oriented Programming Languages



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Object-Oriented Programming Languages

- Örnek olarak, bir bilgisayar futbol oyunu yazmak istediğinizi varsayalım. Oyunu prosedürel dillerde modellemek oldukça zordur. Ancak OOP dillerini kullanarak, programı futbol oyunlarında görünen "gerçek şeylere" göre kolayca modelleyebilirsiniz.
- Oyuncu: özellikler arasında ad, numara, alandaki x ve y konumu vb. Bulunur; operasyonlar arasında koşma, zıplama, topa vurma vb.
- Top: nitelikler, alandaki x, y, z konumunu, yarıçapı, ağırlığı vb. İçerir.
- Hakem:
- Alan:
- İzleyici:
- Hava:
- En önemlisi, bu sınıflardan bazıları (Ball ve Audience gibi) başka bir uygulamada, örneğin bilgisayar basketbolu oyununda çok az değişiklik yapılarak veya hiç değişiklik yapılmadan yeniden kullanılabilir.



Classes (Entities) in a Computer Soccer Game

OOP'nin faydaları

- Prosedüre yönelik diller, temel birim işlevi ile prosedürlere odaklanır. Önce tüm işlevleri anlamanız ve ardından verilerin nasıl temsil edileceğini düşünmeniz gerekir.
- Nesne yönelimli diller, temel birim olarak nesnelerle kullanıcının algıladığı bileşenlere odaklanır. Kullanıcının verilerle etkileşimini tanımlayan tüm verileri ve işlemleri koyarak tüm nesneleri çözersiniz.
- Makinenin bitleri ve baytları yerine sorunlu alanda düşünebileceğiniz için yazılım tasarımında kolaylık sağlar. Üst düzey kavramlar ve soyutlamalarla uğraşırsınız. Tasarım kolaylığı, daha verimli yazılım geliştirmeye yol açar.
- Yazılım bakımında kolaylık: nesneye yönelik yazılımın anlaşılması daha kolaydır, bu nedenle test edilmesi, hata ayıklaması ve bakımı daha kolaydır.
- Yeniden kullanılabilir yazılım: tekerlekleri yeniden icat etmenize ve farklı durumlar için aynı işlevleri yeniden yazmanıza gerek yoktur. Yeni bir uygulama geliştirmenin en hızlı ve en güvenli yolu, mevcut kodları yeniden kullanmaktır - tamamen test edilmiş ve kanıtlanmış kodlar.

Class & Instances

- Java'da bir sınıf, aynı türden nesnelerin tanımıdır. Başka bir deyişle, bir sınıf, aynı türden tüm nesnelerde ortak olan statik nitelikleri ve dinamik davranışları tanımlayan bir plan, şablon veya prototiptir.

  Bir örnek, bir sınıfın belirli bir öğesinin gerçekleştirilmesidir. Başka bir deyişle, bir örnek, bir sınıfın somutlaştırılmasıdır. Bir sınıfın tüm örnekleri, sınıf tanımında açıklandığı gibi benzer özelliklere sahiptir. Örneğin, "Öğrenci" adlı bir sınıfı tanımlayabilir ve "Alice", "Beng" ve "Ali" için "Öğrenci" sınıfının üç örneğini oluşturabilirsiniz.

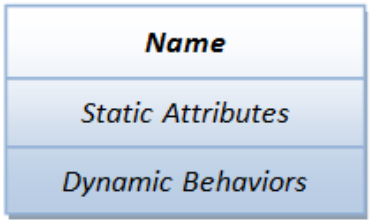
  "Nesne" terimi genellikle örneği ifade eder. Ancak genellikle gevşek bir şekilde kullanılır ve bir sınıfa veya bir örneğe atıfta bulunabilir.

Sınıf, Veri ve İşlemleri Kapsayan 3 Bölmeli bir Kutudur

- Bir sınıf, üç bölmeli bir kutu olarak görselleştirilebilir:

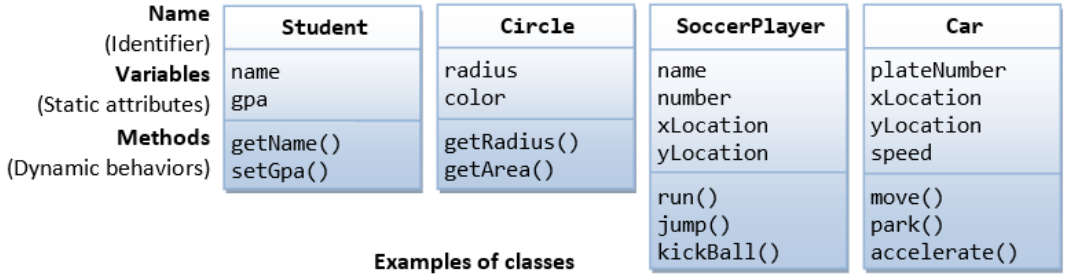
İsim (veya kimlik): sınıfı tanımlar.
Değişkenler (veya nitelik, durum, alan): sınıfın statik niteliklerini içerir.
Yöntemler (veya davranışlar, işlev, işlem): sınıfın dinamik davranışlarını içerir.

Başka bir deyişle, bir sınıf, statik öznitelikleri (veriler) ve dinamik davranışları (veriler üzerinde çalışan işlemler) bir kutuda kapsüller.

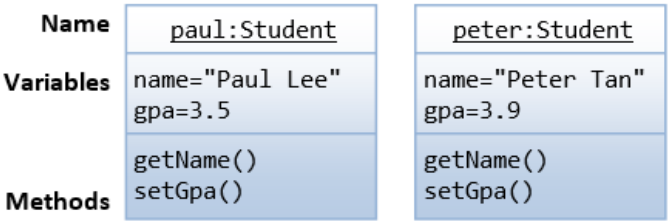


A class is a 3-compartment box

Sınıf, Veri ve İşlemleri Kapsayan 3 Bölmeli bir Kutudur



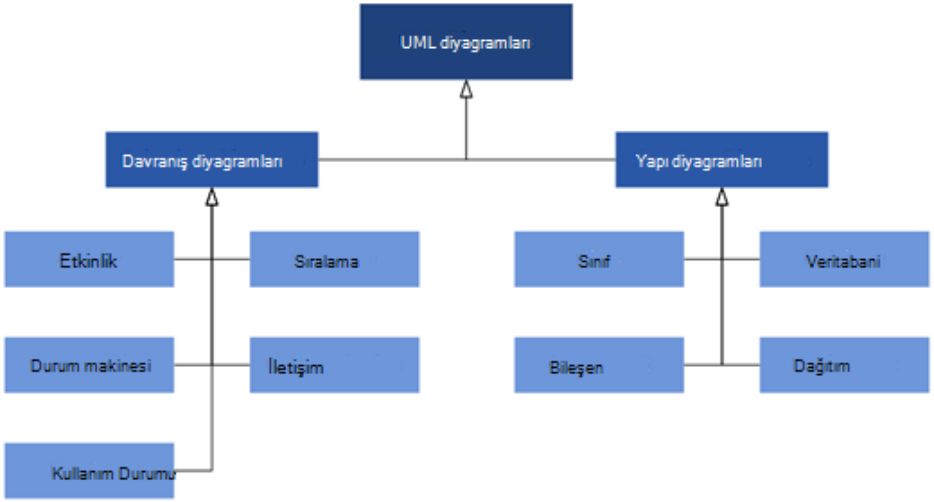
Examples of classes



Two instances - paul and peter - of the class Student

Birleşik Modelleme Dili (UML) Sınıfı ve Örnek Diyagramları:

- Sınıf diyagramları UML notasyonlarına göre çizilir. Bir sınıf, sırasıyla adı, değişkenleri ve yöntemleri içeren 3 bölmeli bir kutu olarak temsil edilir. Sınıf adı kalın ve ortalanmış olarak gösterilir. Bir örnek, örnek adı örnek adı: Sınıf adı olarak gösterilen ve altı çizili 3 bölmeli bir kutu olarak da temsil edilir.



Java'da Sınıf Tanımı

- Sınıf Adlandırma Sözleşmesi: Sınıf adı, birkaç kelimeden oluşan bir isim veya isim cümlesi olacaktır. Tüm sözcükler büyük harfle başlamalıdır. Sınıf adı için tekil bir isim kullanın. Anlamlı ve kendini tanımlayan bir sınıf adı seçin. Örnekler için SoccerPlayer, HttpProxyServer, FileInputStream, PrintStream ve SocketFactory.

```
[AccessControlModifier] class ClassName {
   // Class body contains members (variables and methods)
   ......
}
```

```
public class Circle {          // class name
   double radius;              // variables
   String color;

   double getRadius() { ..... }  // methods
   double getArea() { ..... }
}
```

```
public class SoccerPlayer {  // class name
   int number;               // variables
   String name;
   int x, y;

   void run() { ..... }      // methods
   void kickBall() { ..... }
}
```

Bir Sınıfın Örneklerini Oluşturma

- Belirli bir sınıfın örnek tanımlayıcısını (örnek adı) bildirin.
- ‘new’ operatörünü kullanarak örneği oluşturun (yani, örnek için depolama alanı ayırın ve örneği başlatın).
- Bir örnek bildirildiğinde ancak oluşturulmadığında, null adında özel bir değer tutar.

```
// Declare 3 instances of the class Circle, c1, c2, and c3
Circle c1, c2, c3;  // They hold a special value called null
// Construct the instances via new operator
c1 = new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");

// You can Declare and Construct in the same statement
Circle c4 = new Circle();
```

Nokta (.) Operatörü

- Bir sınıfa ait değişkenler ve yöntemler, resmi olarak üye değişkenler ve üye yöntemleri olarak adlandırılır. Bir üye değişken veya yönteme başvurmak için şunları yapmanız gerekir:

  Öncelikle ilgilendiğiniz durumu tanımlayın ve ardından,
  İstenen üye değişkenine veya yönteme başvurmak için nokta operatörünü (.) Kullanın.

  Örneğin, iki üye değişkeni (radius ve color) ve iki üye yöntemi (getRadius () ve getArea ()) olan Circle adında bir sınıfımız olduğunu varsayalım. Circle sınıfının c1, c2 ve c3 olmak üzere üç örneğini oluşturduk. GetArea () yöntemini çağırmak için, önce ilgilenilen örneği, örneğin c2'yi tanımlamanız, ardından c2.getArea () biçiminde nokta operatörünü kullanmanız gerekir.

Nokta (.) Operatörü

- Örnek tanımlanmadan getArea () çağrısı yapmak anlamsızdır, çünkü yarıçap bilinmemektedir (her biri kendi yarıçapını koruyan birçok Circle örneği olabilir). Ayrıca, c1.getArea () ve c2.getArea () farklı sonuçlar üretebilir.

- Genel olarak, aVariable adlı bir üye değişkeni ve aMethod () adlı bir üye yöntemi olan AClass adında bir sınıf olduğunu varsayalım. AClass için anInstance adında bir örnek oluşturulur. AnInstance.aVariable ve anInstance.aMethod () kullanırsınız.

```
// Suppose that the class Circle has variables radius and color,
//  and methods getArea() and getRadius().
// Declare and construct instances c1 and c2 of the class Circle
Circle c1 = new Circle ();
Circle c2 = new Circle ();
// Invoke member methods for the instance c1 via dot operator
System.out.println(c1.getArea());
System.out.println(c1.getRadius());
// Reference member variables for instance c2 via dot operator
c2.radius = 5.0;
c2.color = "blue";
```

Üye Değişkenleri

```
[AccessControlModifier] type variableName [= initialValue];
[AccessControlModifier] type variableName-1 [= initialValue-1] [, type variableName-2 [= initialValue-2]] ... ;
```

```
private double radius;
public int length = 1, width = 1;
```

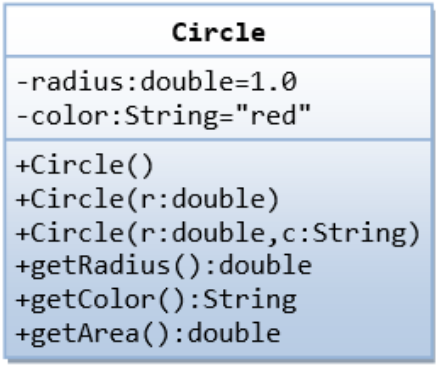
Üye Metotları

```
[AccessControlModifier] returnType methodName ([parameterList]) {
   // method body or implementation
   ......
}
```

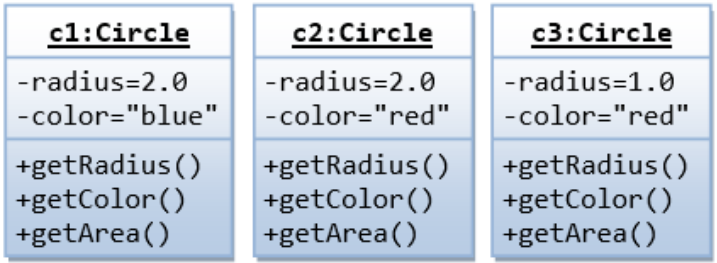
```
// Return the area of this Circle instance
public double getArea() {
   return radius * radius * Math.PI;
}
```

Örnek

Class Definition



Instances



```java
/**
 * The Circle class models a circle with a radius and color.
 */
public class Circle {    // Save as "Circle.java"
   // Private instance variables
   private double radius;
   private String color;

   // Constructors (overloaded)
   /** Constructs a Circle instance with default radius and color */
   public Circle() {                   // 1st Constructor (default constructor)
      radius = 1.0;
      color = "red";
   }
   /** Constructs a Circle instance with the given radius and default color*/
   public Circle(double r) {           // 2nd Constructor
      radius = r;
      color = "red";
   }
   /** Constructs a Circle instance with the given radius and color */
   public Circle(double r, String c) { // 3rd Constructor
      radius = r;
      color = c;
   }

   // Public methods
   /** Returns the radius */
   public double getRadius() {  // getter for radius
      return radius;
   }
   /** Returns the color */
   public String getColor() {   // getter for color
      return color;
   }
   /** Returns the area of this circle */
   public double getArea() {
      return radius * radius * Math.PI;
   }
}
```

```java
/**
 * A Test Driver for the "Circle" class
 */
public class TestCircle {    // Save as "TestCircle.java"
   public static void main(String[] args) {   // Program entry point
      // Declare and Construct an instance of the Circle class called c1
      Circle c1 = new Circle(2.0, "blue");  // Use 3rd constructor
      System.out.println("The radius is: " + c1.getRadius());  // use dot operator to invoke member methods
      //The radius is: 2.0
      System.out.println("The color is: " + c1.getColor());
      //The color is: blue
      System.out.printf("The area is: %.2f%n", c1.getArea());
      //The area is: 12.57

      // Declare and Construct another instance of the Circle class called c2
      Circle c2 = new Circle(2.0);  // Use 2nd constructor
      System.out.println("The radius is: " + c2.getRadius());
      //The radius is: 2.0
      System.out.println("The color is: " + c2.getColor());
      //The color is: red
      System.out.printf("The area is: %.2f%n", c2.getArea());
      //The area is: 12.57

      // Declare and Construct yet another instance of the Circle class called c3
      Circle c3 = new Circle();  // Use 1st constructor
      System.out.println("The radius is: " + c3.getRadius());
      //The radius is: 1.0
      System.out.println("The color is: " + c3.getColor());
      //The color is: red
      System.out.printf("The area is: %.2f%n", c3.getArea());
      //The area is: 3.14
   }
}
```

KAYNAKLAR

- Programming Notes. (2021, March 11). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming/index.html
- Çobanoğlu B. (2020) Java ile Programlama ve Veri Yapıları. İstanbul Pusula Yayıncılık. 978-605-2359-84-6