YMH 311-YAZILIM TASARIM VE MİMARİSİ

Dr. Öğr. Üyesi BİHTER DAŞ

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

3. HAFTA:

UML MODELLEME VE DİYAGRAMLAR

UML MODELLEME

Birleşik Modelleme Dili (Unified Modeling Language - UML), yazılım sistemlerini, özellikle nesne yönelimli bir yaklaşım ile oluşturulan yazılım sistemlerini tanımlamaya ve tasarlamaya yardımcı olan grafik notasyonlar ile modelleme imkânı sunan standart bir modelleme dilidir. UML model, gerçek sistemin daha küçük ve basit bir şeklidir. Böylelikle sistem daha kolay anlaşılır. Bir modelleme yaparken UML modelleme kullanılmaktadır.

Bir modelleme dili, sözde kod (pseudo-code), gerçek kod, diyagramlar ve resimlerden oluşabilir.

Fakat UML asla bir programlama dili değildir

UML MODELLEME

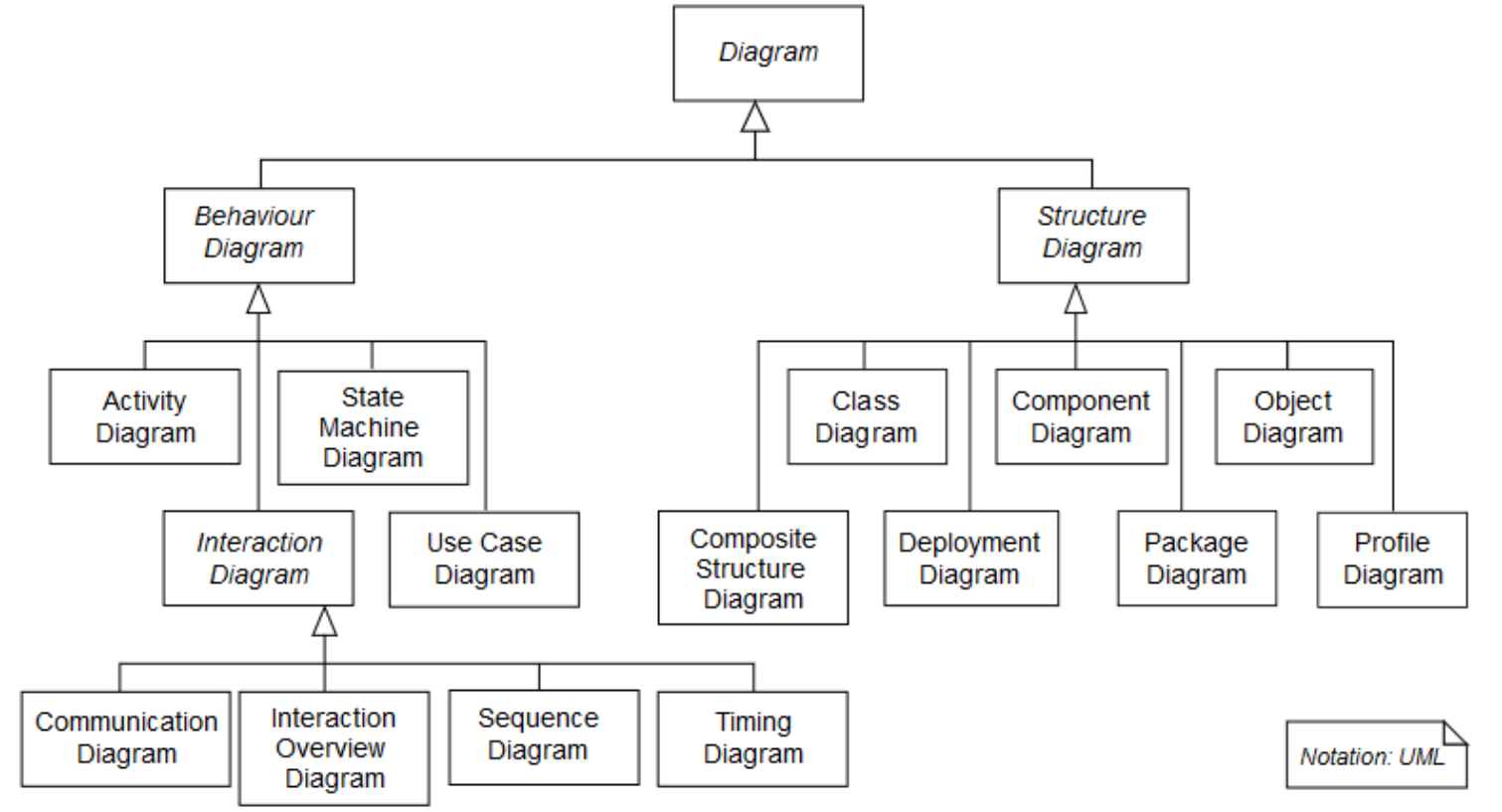
UML diyagramlarının avantajları:

- **Biçimsel bir dildir:** UML içindeki her bir elemanın anlamı vardır. Anlamlar kişiden kişiye değişmez ve farklı yorumlara sebep olmaz.
- **Kısa bir dildir:** Kısa ve basit bir notasyonu vardır.
- **Anlaşılır bir dildir:** Bir sistemin tüm yönlerini tanımlar.
- **Ölçeklenebilir bir dildir:** Küçük projelerden büyük projelere kadar, farklı büyüklükteki projelerde kullanılır.
- **Tecrübelerden oluşmuştur:** Nesneye Yönelik Programlama topluluklarının geçiş tecrübelerinden yola çıkılarak oluşturulmuştur.
- **Bir standarttır:** Object Management Group (OMG) tarafından kontrol edilen bir standarttır.

UML MODELLEME

- Tasarım ve analizi yapılmış olduğu için daha kolay kodlama yapılır.
- Hatalar minimuma indirilir
- Bellek kullanımı daha verimli olur
- Programın kararlılığı artar(Senaryoların yardımıyla)
- Takım çalışması için harika bir yardımcıdır

UML Modelleme



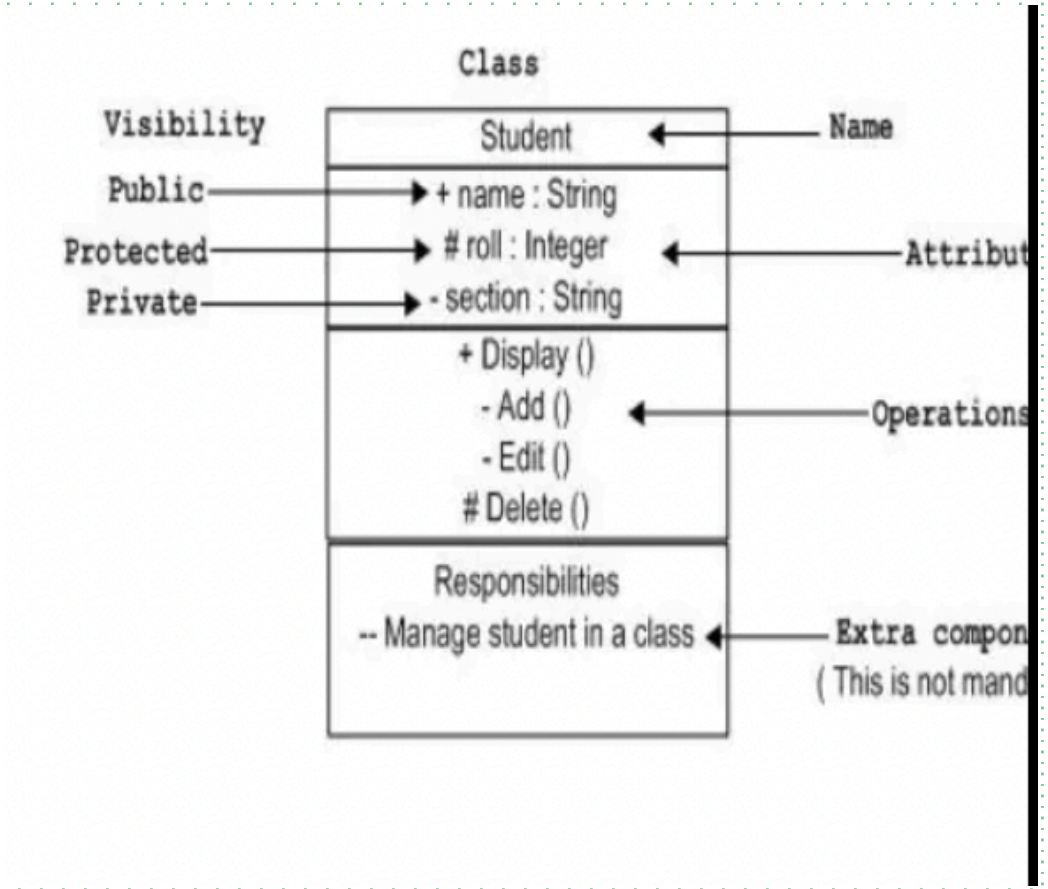
1. Sınıf (Class) Diyagramları

- Sınıf diyagramları, nesne yönelimli modellemede ana yapı taşıdır.
- Sistemin statik yapısını ifade eder.
- Bir sistemdeki farklı nesneleri (objects), özniteliklerini(attributes), işlemlerini (operations) ve nesneler arasındaki ilişkileri göstermek için kullanılırlar.
- Yapısal (structural) bir diyagramdır.

Sınıf (Class) diyagramları nasıl çizilir?

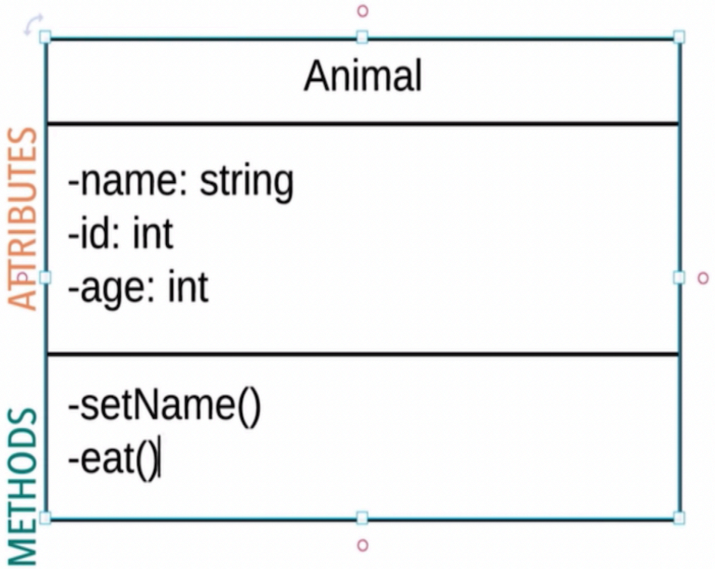
- Sınıflar dikdörtgen şeklinde çizilir.
- Sınıfların çizildiği dikdörtgenler dikey olarak 3 parçaya ayrılır. İlk parçaya sınıfın ismi yazılır. İkinci parçaya sınıfın özellikleri yani değişkenler yazılır. Üçüncü parçaya sınıfın yaptığı işlemler yani metotlar yazılır.
- Yazılan özellikler için erişim belirleyici + özellik ismi + :özellik tipi şeklinde belirtim yapılır.
- Yazılan metotlar için erişim belirleyici + metot ismi + (metot parametreleri) + dönüş tipi şeklinde belirtim yapılır.
- Erişim belirleyiciler için semboller kullanılır. Bu semboller public için +, private için -, protected için # şeklindedir.
- Bunlara ek olarak “notes” (sınıf hakkında ekstra bilgiler) ve “Constraints” adlı sınıfla ilgili çeşitli özel koşullara ait bilgilerde bulunabilir.

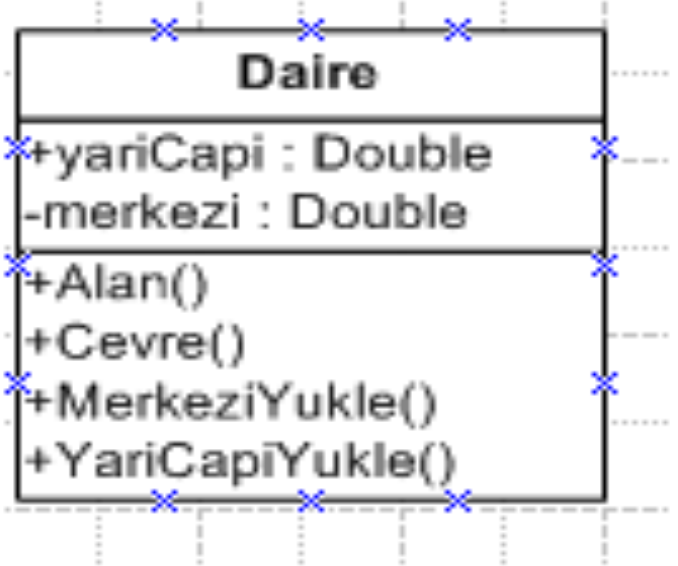
Sınıfın gösterimi



Sınıfın özniteliklerinin bir yazım biçimi vardır. Her bir öznitelik kutucuğun ortasındaki alanda bir satırlık yer kaplar. Bir özniteliğin genel formu şu şekildedir:

<Görünürlük> <İsim>: <Tür> <Multiplicity> = <Varsayılan Değer> {<Property String>}





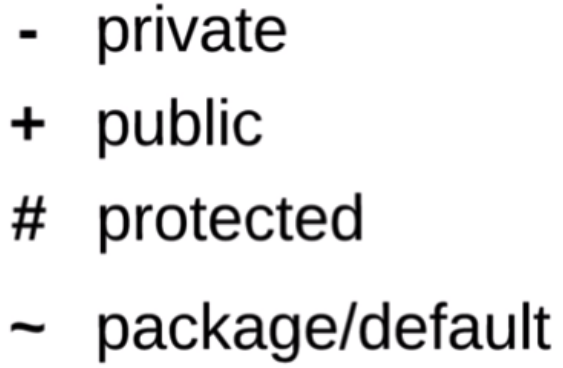
Görünürlük (Visibility): Bu özniteliğin dışarıdan erişim ilkesinin ne olduğunu belirtir. Eğer bir sınıfın üye değişkeni **public** ise buraya "**+**", **private** ise buraya "**–**", **protected** ise buraya "**#**" işareti konulur.

Private (-); Nitelik yada metota sınıf dışında erişim engellenmiştir.

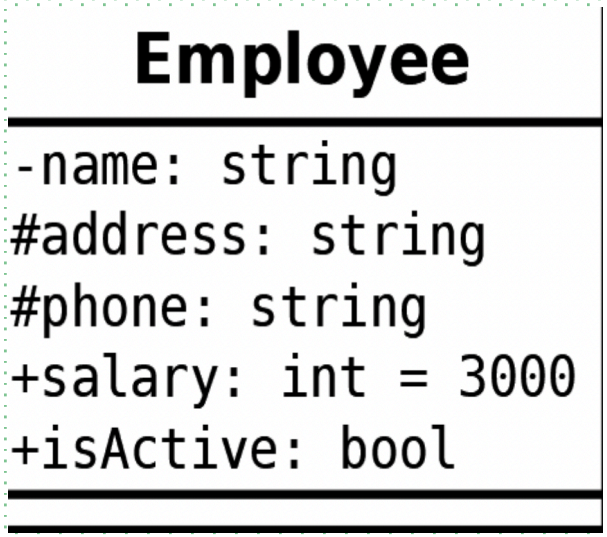
Protected (#); Nitelik yada metota erişim sınırlandırılmıştır.

Public (+); Nitelik yada metot genel kullanıma açıktır.

Package(~); Nitelik yada metot aynı paketteki (*package*) diğer sınıflar tarafından erişilebilir.

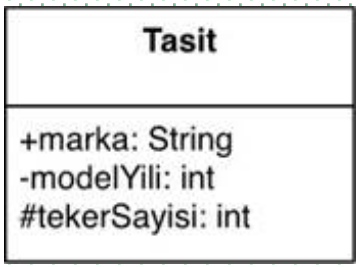


- **İsim (Name):** Genellikle buraya o özniteliğe programlamada karşılık gelen değişkenin ismi yazılır.
- **Tür (Type):** Bir özniteliğin türü, programlamada ona karşılık gelen değişkenin türünü belirtir. Bu da öznitelik ile ilgili kısıtlamaları (alabileceği minimum ve maksimum değerler gibi) daha iyi anlamamıza yardımcı olur.



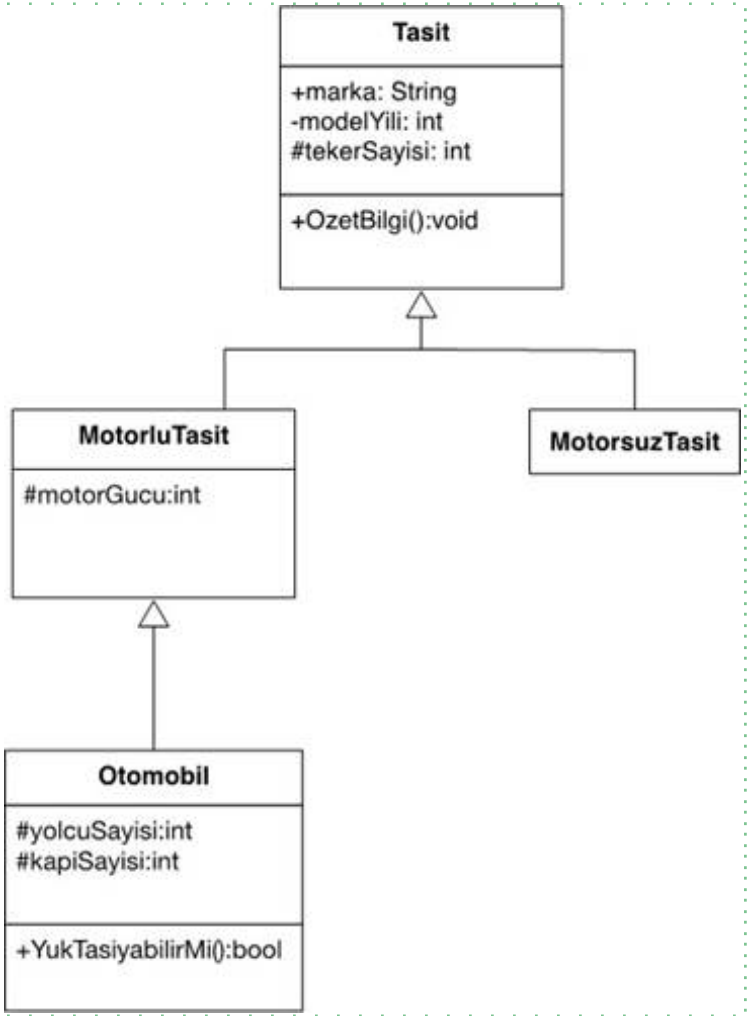
Bir sınıfın, sınıf niteliklerinin tanımlanması ve kodlanması


```
class Tasit {

}
```

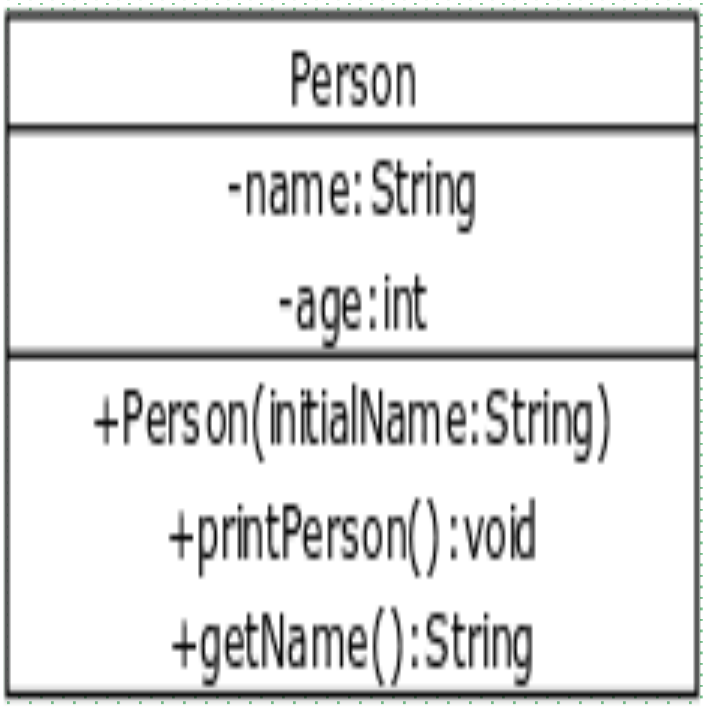



```
class Tasit {
   public String marka;
   private int modelYili;
   protected int tekerSayisi;
}
```


```
class Tasit
 {
    public String marka;
    private int modelYili;
    protected int tekerSayisi;


    public void OzetBilgi()
    {
    }
 }
```




```
class Tasit{
    public String marka;
    private int modelYili;
    protected int tekerSayisi;


    public void OzetBilgi()
    {
    }
}
class MotorsuzTasit: Tasit {
}


class MotorluTasit:Tasit {
    protected int motorGucu;
}

class Otomobil : MotorluTasit {
    protected int yolcuSayisi;
    protected int kapiSayisi;
    public bool YukTasiyabilirMi(){

    }
}
```



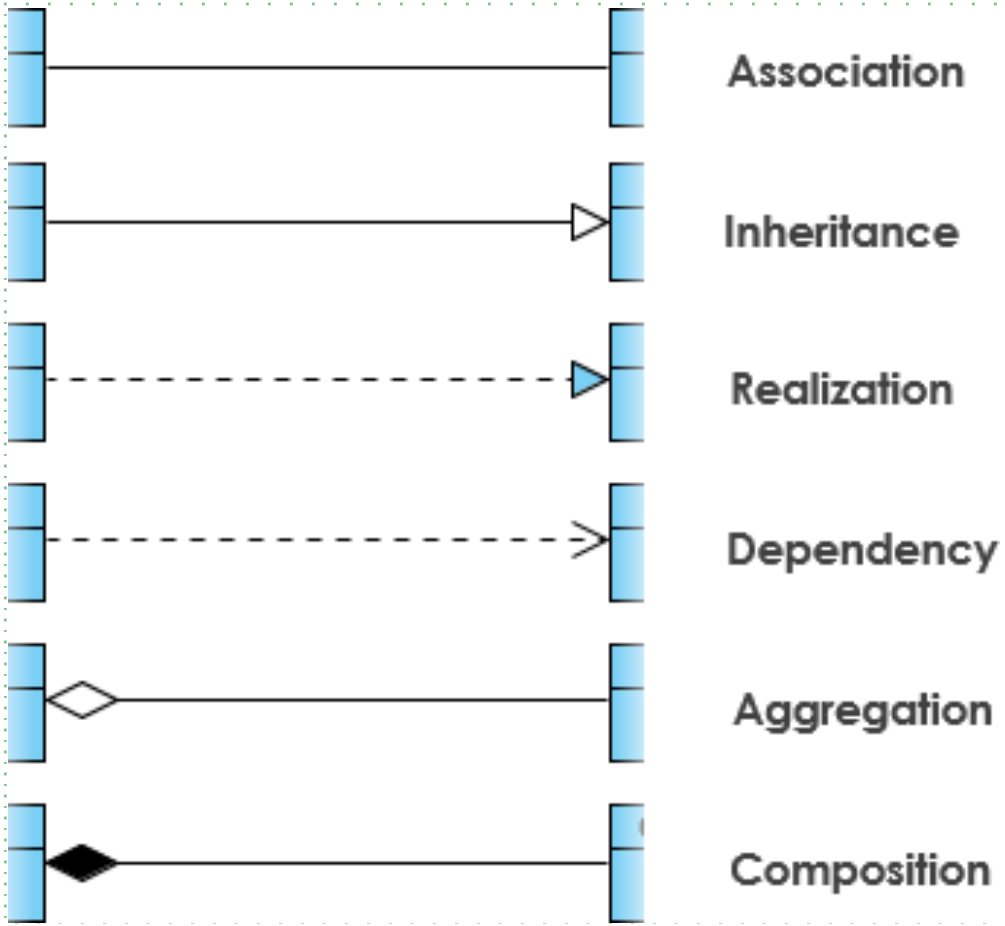

```java
public class Person {
    private String name;
    private int age;

    public Person(String initialName) {
        this.name = initialName;
        this.age = 0;
    }

    public void printPerson() {
        System.out.println(this.name + ", age " +   this.age + " years");
    }

    public String getName() {
        return this.name;
    }
}
```

Sınıflar arasındaki bağlantıların gösterimi

Sınıflar arası ilişkilerin listesi şu şekildedir: UML diyagramlarında sınıflar arasındaki ilişkiler çizgiler ve oklar ile temsil edilir. Çizginin ve okun tipi, onun ne tür bir ilişki olduğu ile ilgili ipucu verir. UML'de temel olarak 6 tür ilişki (*relationship*) bulunur.

- Association (Ortaklık)
- Dependency (Bağımlılık)
- Aggregation (Birleşme)
- Composition (Kompozisyon)
- Generalization (Genelleme) veya Inheritance (Kalıtım)
- Realization (Gerçekleme)

UML’de bu ilişki türlerinin gösterimi:



Association(Ortaklık)

Association'lar iki sınıf arasında düz çizgi olarak gösterilir. Association'ların uç kısımlarında herhangi bir ok işareti yer almayabilir. Bu durumda Association ile birbirine bağlanan sınıfların, birbiri ile çift yönlü (*bidirectional*) bir ilişkiye sahip olduğu anlaşılır. Ancak Association'larda tek yönlü bir ilişki de mevcuttur. İlişki türüne göre Association'lar 3 tiptir:

Unidirectional Association (Tek Yönlü Ortaklık)

Bidirectional Association (Çift Yönlü Ortaklık)

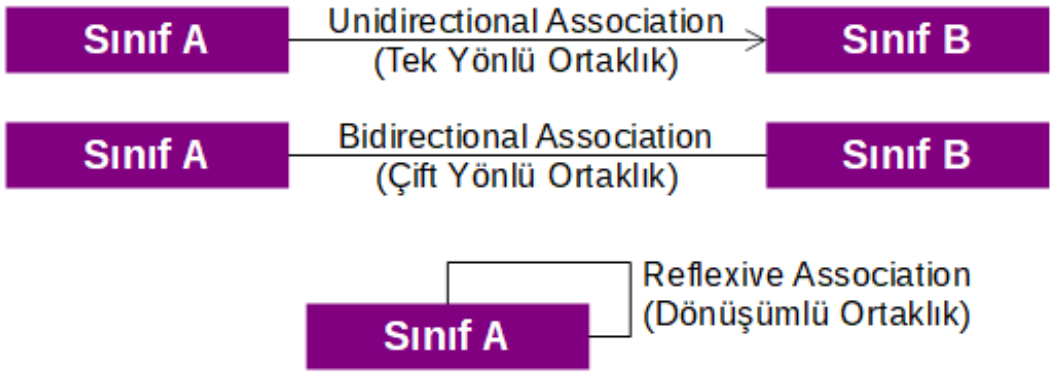
Self/Reflexive Association (Öz/Dönüşümlü Ortaklık)

Association(Ortaklık)

Unidirectional Association (Tek Yönlü Ortaklık) sadece bir sınıfın diğer sınıf türünden bir değişkeni barındırdığı durumu anlatır.

Bidirectional Association (Çift Yönlü Ortaklık) ise her iki sınıfın da birbirleri türünden değişkenleri barındırdığı durumu anlatır.

Self/Reflexive Association (Öz/Dönüşümlü Ortaklık) ise bir sınıfın yine kendi türünden bir değişkeni içerisinde barındırdığı durumu anlatır. Association türlerinin UML'de gösterimi şu şekilde özetlenebilir:



Bağıntı ilişkileri için tanımlanmış bilgiler aşağıdaki gibidir; Sınıflar arasında “bire-sıfır”, “bire-bir”, “bire-çok” türünde ilişkiler kurulabilir.

Bağıntının adı

Sınıfın bağıntıdaki rolü

Bağıntının çokluğu

Çokluk Bezemesi	Açıklama
1	Yalnızca 1 (herhangi bir sayma sayısı da kullanılabilir)
0..1	Yalnızca 0 ya da 1
M..N	En az M, en çok N
*	0 ya da daha çok
0..*	0 ya da daha çok
1..*	1 ya da daha çok

Sınıflar arasında HAS-A ilişkisi kurulur

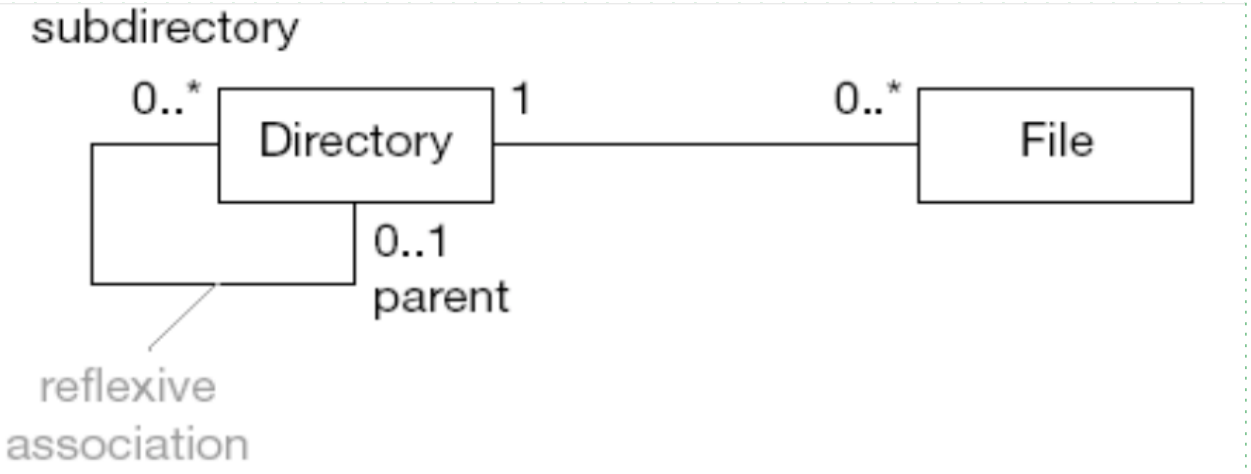
Unidirectional Association (Tek Yönlü Ortaklık)

"Person HAS AN Address"
"İnsan bir adrese sahiptir"

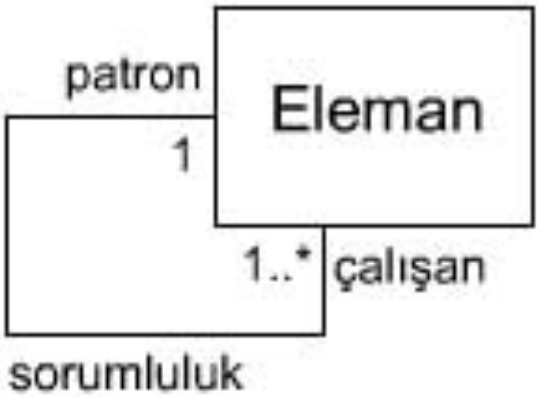
Bidirectional Association (Çift Yönlü Ortaklık)

Student sıfır ya da sonsuz tane kursa kayıt olmuş olabilir, aynı şekilde kursa hiç öğrenci kayıt olmamış olabilir ya da sonsuz tane öğrenci kayıt olabilir.

Self/Reflexive Association (Öz/Dönüşümlü Ortaklık)



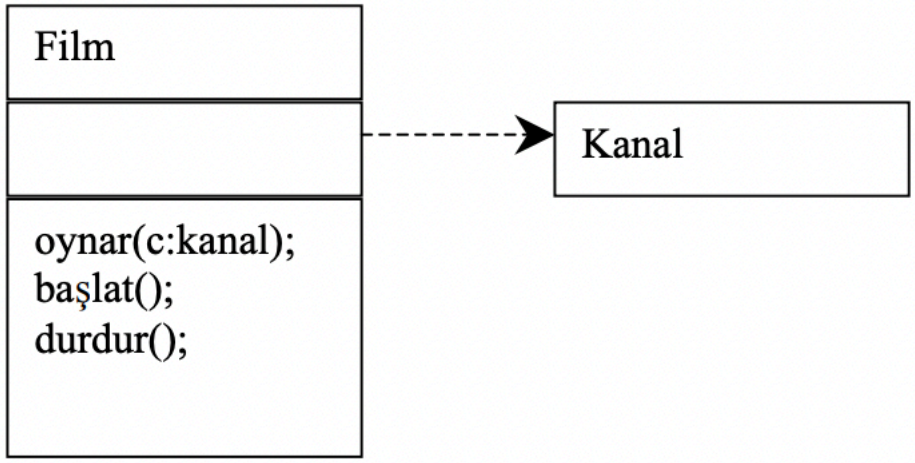
Directory(Dosya dizini) hiç ya da sonsuz sayıda alt dosya dizinine sahip olabilir. Aynı şekilde hiç ya da 1 tane üst dizine sahip olabilir.



Bir patron bir eleman olmasına rağmen kendisi gibi eleman olan birden çok çalışandan sorumludur.

Dependency(Bağımlılık)

Eğer bir sınıf, diğer sınıf metotlarından en az birinde parametre olarak kullanılıyorsa iki sınıf arasında bağımlılık ilişkisi vardır. Bağımlılık ilişkisi, bağımlı sınıftan bağımsız sınıfa doğru kesikli çizgi ile çizilen bir ok ile gösterilmektedir.



Film sınıfı kanal sınıfına bağımlıdır. Kanal sınıfında yapılacak bir değişiklik Film sınıfında da bir değişiklik gerektirir. Tersi söz konusu değildir.

Aggregation & Composition

Bu ilişki türü çift yönlü association olarak düşünülebilir. Aggregation ve Composition'da sınıflar arasında parçası olma anlamı vardır. Bir sınıf diğer sınıfın bir parçasıdır. Bu ilişki Aggregation'da biraz daha zayıfken, Composition'da daha güçlüdür. Bu tür ilişkiler HAS-A ya da IS-PART-OF şeklinde okunabilir.

Aggregation

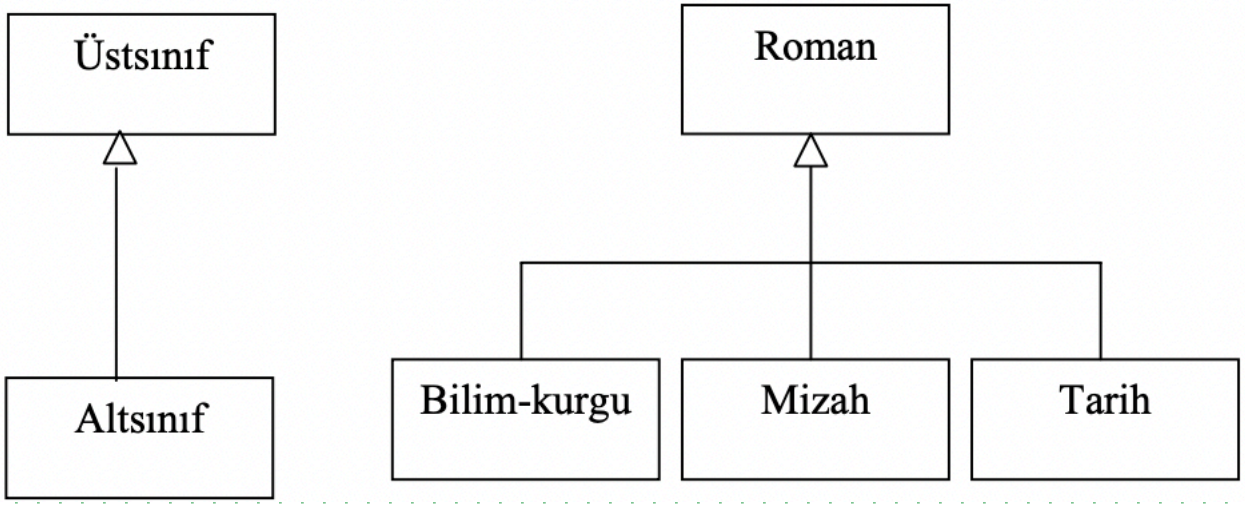
Tekerlek araba sınıfının bir parçasıdır. Ancak araba sınıfı yok olduğunda tekerlek yok olmak zorunda değildir. Aralarında zayıf bir parça ilişkisi vardır.

Composition

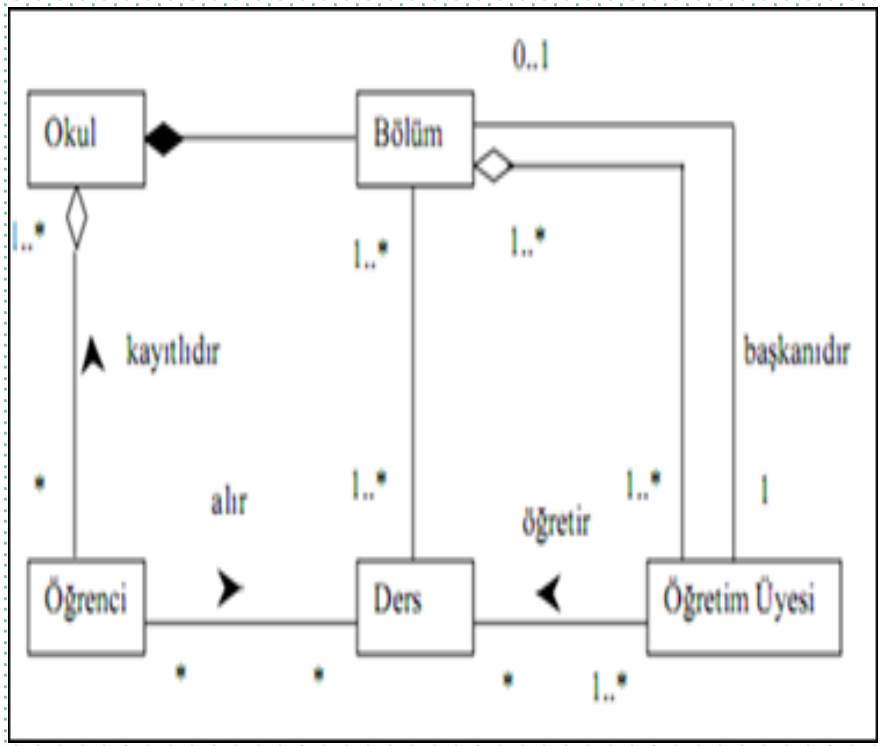
Kalp, insan sınıfının bir parçasıdır. İnsan sınıfı yok olduğunda kalpte yok olacaktır. İki sınıf arasında güçlü bir parça ilişkisi vardır.

Generalization/Inheritance

Bu ilişkide nesneler genelden özele doğru sıradüzensel bir dizilim içinde yerleştirilir. Sınıflar arasında üst sınıf ve alt sınıf ayırımının yapılması gerekmektedir. Alt sınıflar içi boş olan bir ok ile üst sınıfa bağlanırlar.



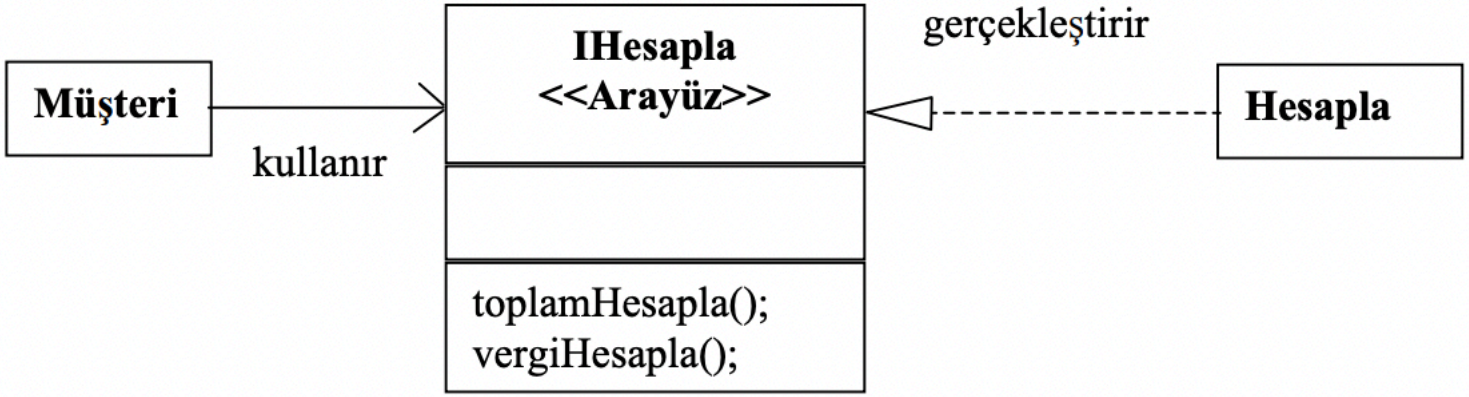
Bu tür kalıtım ilişkisinde nesneler arasında “IS-A” ilişkisi bulunmaktadır.
Bilim-kurgu is a roman
Bilim-kurgu bir roman türüdür.



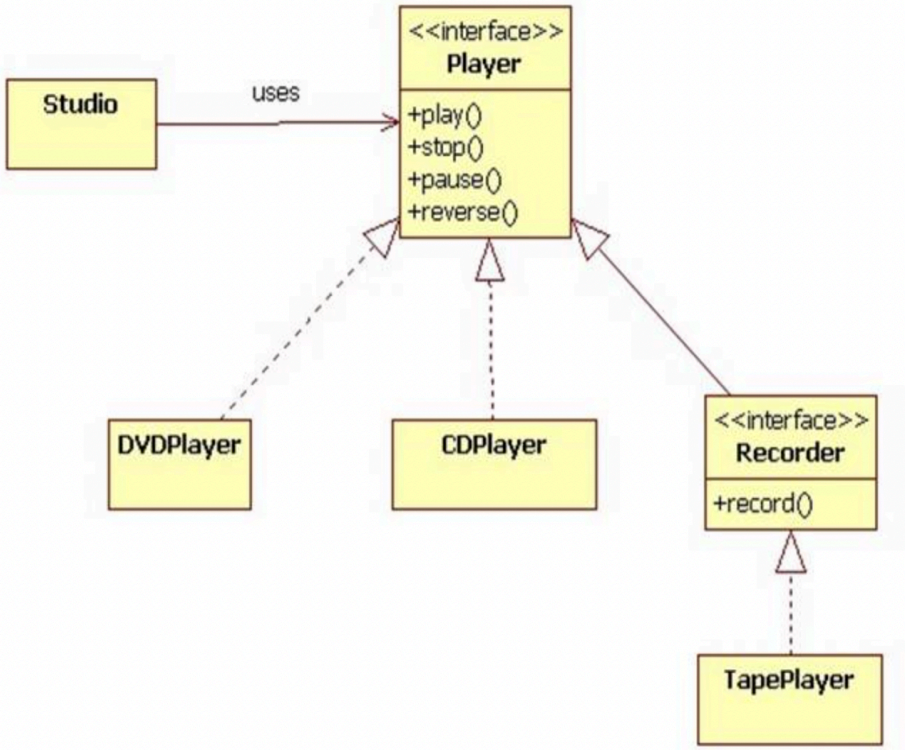
•Her okulun sıfır ya da daha fazla öğrencisi yer almaktadır.
•Her okulda bir ya da daha fazla bölüm vardır.
•Her bölüm tek bir okula bağlıdır.
•Her bölüm bir ya da daha fazla sayıda ders açmaktadır.
•Her bölüm bir ya da daha fazla öğretim üyesine sahiptir.
•Her bölümün öğretim üyelerinden seçilmiş bir başkanı vardır.
•Her öğretim üyesi bir ya da daha fazla bir bölümde görev yapmaktadır.
•Her öğretim üyesi sıfır ya da daha fazla sayıda ders vermektedir.
•Her ders bir ya da daha çok bölüm tarafından açılır.
•Her dersi sıfır ya da daha fazla sayıda öğrenci almaktadır.
•Her derse en az bir tane öğretim üyesi atanmıştır.
•Her öğrenci bir ya da daha fazla sayıda okula kayıtlıdır.
•Her öğrenci sıfır ya da daha fazla sayıda ders alabilir.
•Her öğretim üyesi ya tekbir bölümün başkanıdır ya da hiçbir bölümün başkanı değildir.

Realization (Gerçekleme)

Bu ilişki kullanıcı arayüzler ile sınıflar arasındaki ilişkiyi modellemek için kullanılır. Kesikli çizgi ile ifade edilir. Kalıtımdaki çizginin kesik kesik olan halidir. Arayüz sadece metot adlarını ve bunların parametrelerini içerir.



Ihesapla adlı diğer sınıfların kullanımına açılmış bir arayüz sınıfı tanımlanmıştır. Bu sınıfta yer alan metotların gerçekleştirimi Hesapla sınıfı tarafından üstlenilmiş durumdadır. İyi bir tasarımda tüm sınıflar yalnızca arayüzde yer alan metotları kullanmalı, doğrudan Hesapla sınıfının metotlarını çağırmamalıdır. Böylece Hesapla sınıfının gerçekleştirimi uygulamanın kalanını etkilemeksizin istendiği gibi değiştirilebilir.




```
interface Player {
   void play();
   void stop();
   void pause();
   void reverse();
}

interface Recorder extends Player {
   void record();
}

class TapePlayer implements Recorder {
   public void play() { ... }
   public void stop() { ... }
   public void pause() { ... }
   public void reverse() { ... }
   public void record() { ... }
}
```

Extends ile bir sınıf başka sınıfı miras alabilir, yada bir arayüz başka bir arayüzü miras alabilir. İmplements ile bir sınıf bir arayüzü implemente edebilir.

```java
interface A {
   void funcA();
}
interface B extends A {
   void funcB();
}
class C implements B {
   public void funcA() {
      System.out.println("This is funcA");
   }
   public void funcB() {
      System.out.println("This is funcB");
   }
}
```

Yukarıdaki kod parçasına uygun Class diyagramını çiziniz.

Örnek: Kütüphane yönetim sistemi

Classes of Library Management System :

Library Management System class –
It manages all operations of Library Management System. It is central part of organization for which software is being designed.

User Class –
It manages all operations of user.

Librarian Class – It manages all operations of Librarian.

Book Class –
It manages all operations of books. It is basic building block of system.

Account Class –
It manages all operations of account.

Library database Class –
It manages all operations of library database.

Staff Class –
It manages all operations of staff.

Student Class –
It manages all operations of student.

Attributes of Library Management System :

Library Management System Attributes –
UserType, Username, Password

User Attributes –
Name, Id

Librarian Attributes –
Name, Id, Password, SearchString

Book Attributes –
Title, Author, ISBN, Publication

Account Attributes –
no_borrowed_books, no_reserved_books, no_returned_books, no_lost_books
fine_amount

Library database Attributes –
List_of_books

Staff Class Attributes –
Dept

Student Class Attributes –
Class

Methods of Library Management System :

Library Management System Methods –
Login(), Register(), Logout()

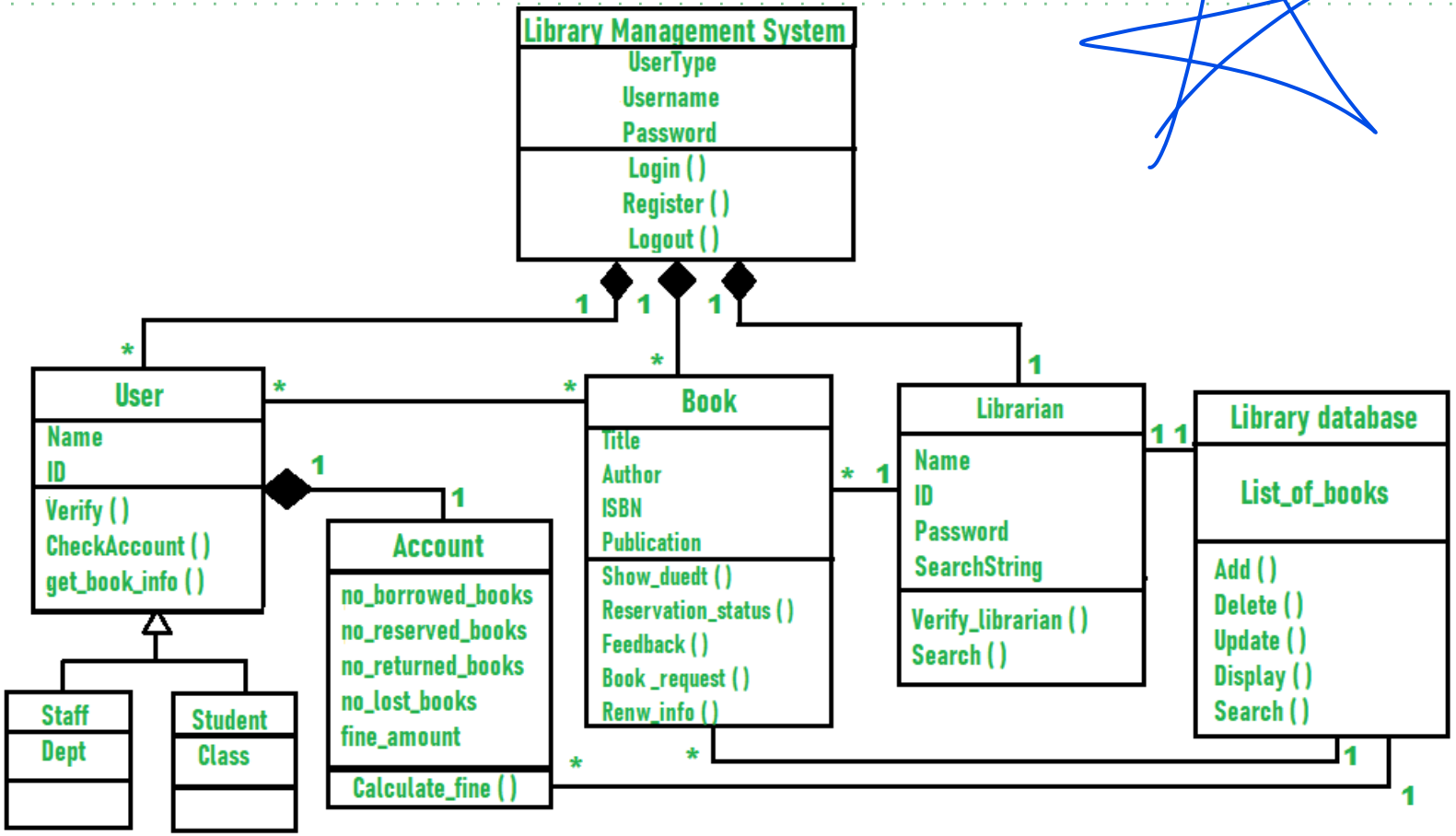
User Methods –
Verify(), CheckAccount(), get_book_info()

Librarian Methods –
Verify_librarian(), Search()

Book Methods –
Show_duedt(), Reservation_status(), Feedback(), Book_request(), Renew_info()

Account Methods –
Calculate_fine()

Library database Methods –
Add(), Delete(), Update(), Display(), Search()



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsiz, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (http://www.buzluca.info/dersler.html)

https://www.mustafayemural.com/

https://tugrulbayrak.medium.com/

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Sorularınız