NESNEYE YÖNELİK PROGRAMLAMA 28.09.2017

Yrd. Doç.Dr. Pelin GÖRGEL

İstanbul Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Nesneye Yönelik Programlama
- Programlama Dilleri
- Java Teknolojisi
- Java Sanal Makinesi (JVM)
- Java Editörleri ve JVM Kurulumu
- Java'ya Giriş

Programlama Dilleri

- Programlama dilleri programlamaya yaklaşım açısından 4 gruba ayrılabilir:
  - Yordamsal diller
    - Fortran, Pascal, Visual Basic, C, Delphi, C++
  - Nesne yönelimli diller
    - C++, C#, Java
  - Mantık yönelimli diller
    - Prolog
  - Görev yönelimli diller
    - Matlab, SQL, PL/SQL, Oracle Forms/Reports

Yordamsal (Procedural) Programlama

- Yordamsal programlamada yazılımlar birbirini çağıran bir dizi yordam (procedure) ve işlevler topluluğu olarak geliştirilir.
- Programda yapılacak işler fonksiyonlara ayırılıp onlar tarafından yapılır. Program fonksiyon çağrıları ile ilerler bir fonksiyon biter sonra diğer bir fonksiyon icrası başlar.
- Her yordam ve işlev kendi yerel verisini, yine kendi yerel değişkenlerinde tutar.
- Paylaşılması gereken veriler yordam çağırma komutlarında parametre olarak yordamdan yordama geçirilir.
- Parametrelere sığmayacak büyük veriler ise genel (global) değişkenler içerisinde herkesin kullanımına açılır.

Yordamsal Programlamanın Zayıf Yanları

- Genel kullanıma açılan veriler tümüyle korumasız kalır.
- Verinin kullanım amacı veri üzerinde yapılabilecek işlemleri hiçbir biçimde sınırlamaz.
- Veriyi tutan değişken genel kullanıma açıldıktan sonra, o değişken türünün desteklediği her türlü işlem veriye uygulanabilir.
- Amaç dışı kullanımdan kaynaklanan yanlışlıklar ortaya çıktıktan sonra, yanlışlığa neden olan program kesiminin saptanması zordur. Bunun için sözkonusu veriye erişen tüm yordamların tek tek incelenmesi gerekir.

Yordamsal Programlamanın Zayıf Yanları

- Aynı verinin sayısal bir değişken yerine karakter türü bir değişkende tutulmasına karar verildiğinde, tek boyutlu dizi yerine iki boyutlu dizi kullanıldığında vs. Sözkonusu veriye erişen tüm yordamların elden geçirilmesi gerekir. Bu durumda yöntem değişikliğinden etkilenen tüm kod kesimleri eksiksiz saptanmalı ve gereken güncellemeler hatasız olarak yapılmalıdır
- Her aşamada sisteme yeni yanlışların sızması ya da dolaylı ilişkilerden ötürü ancak belli süre sonunda saptanabilecek ve bulunması daha zor hataların ortaya çıkması olasılığı yüksektir.

Yordamsal Programlamanın Zayıf Yanları

- Sonuç olarak yordamsal programlamadaki zorluklar şöyle özetlenebilir:
  - Hataların saptanma ve düzeltilmesindeki zorluk
  - Programın kullanıcı gereksinmelerini ve yenilik isteklerini tam karşılayacak biçimde hızla değiştirilebilir olmaması ve her yapılan ek ya da değişikliğin programın daha önce çalışan ve değiştirilmeyen kesimlerinde bile hataların oluşmasına yol açabilmesi
  - Eldeki sınanmış kod kesimlerinin ciddi değişiklikler yapılmaksızın yeni gereksinmelerin karşılanmasında kolayca kullanılamaması.

Nesneye Yönelik Programlama

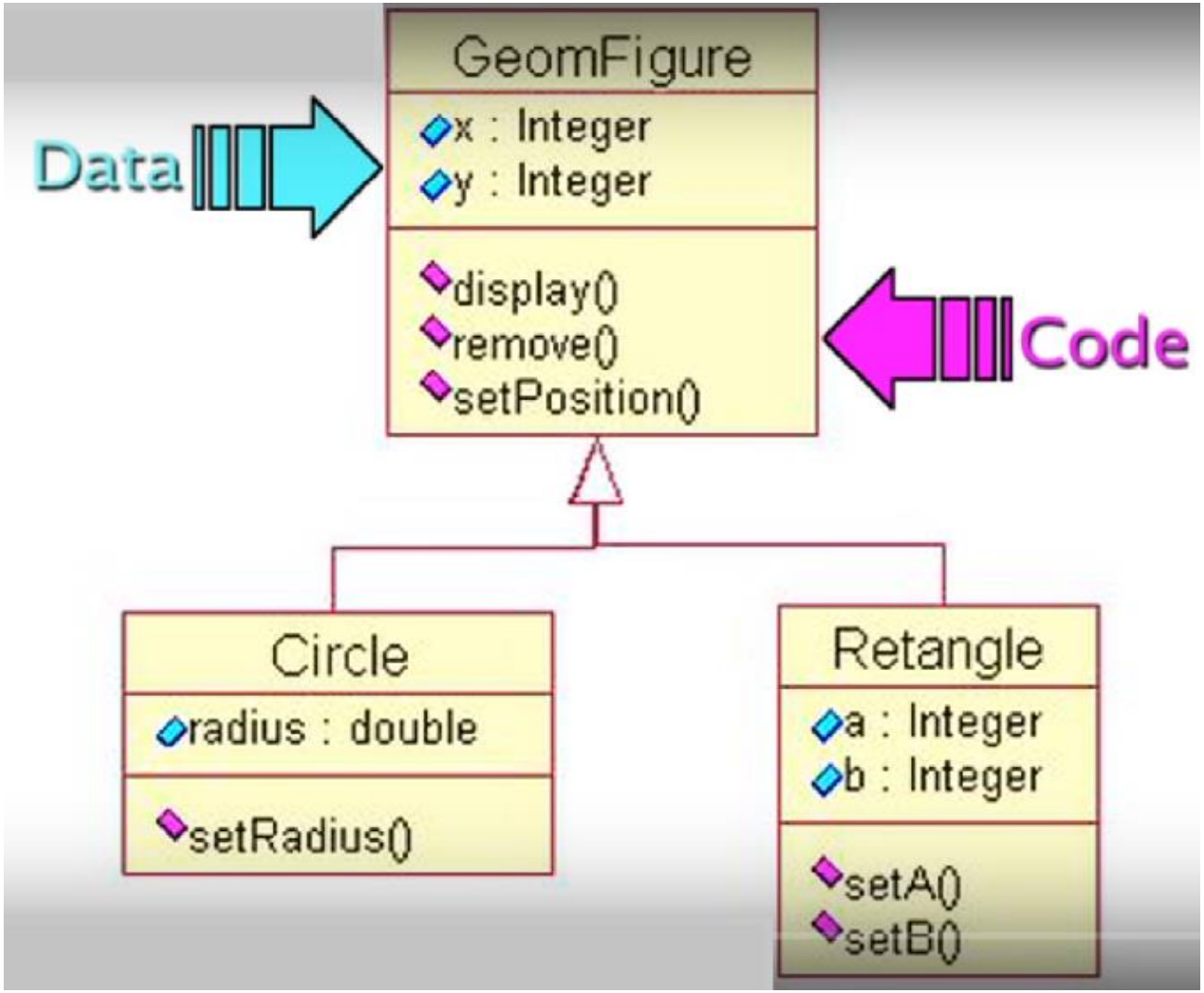
- Nesneye yönelik programlamada yazılımlar birbirleriyle iletişim kurabilen nesneler topluluğu olarak tasarlanır ve gerçekleştirilirler.
- Kod işletimi nesnelerin içinde yapılır ve her nesne bir diğer nesneye ileti göndererek ondan hizmet alabilir. Bu nedenle nesneye yönelik programlama "nesnelere ileti gönderme yoluyla programlama" olarak da isimlendirilir.
- Nesneler, metodlar (methods) ve nitelikler'den (attributes) oluşur. Nitelikler, nesnelerin sahip oldukları verilere, metodlar ise bunlar üzerinde yapılabilecek işlemlere karşılık gelir.
- Bir başka deyişle nesne, kendisini işleyecek kod kesimini kendisi ile birlikte tanımlayan ve taşıyan ve kendi tanımladığı biçimden daha farklı amaçlarla kullanılamayan veri türü olarak yorumlanabilir.

Nesneye Yönelik Programlama

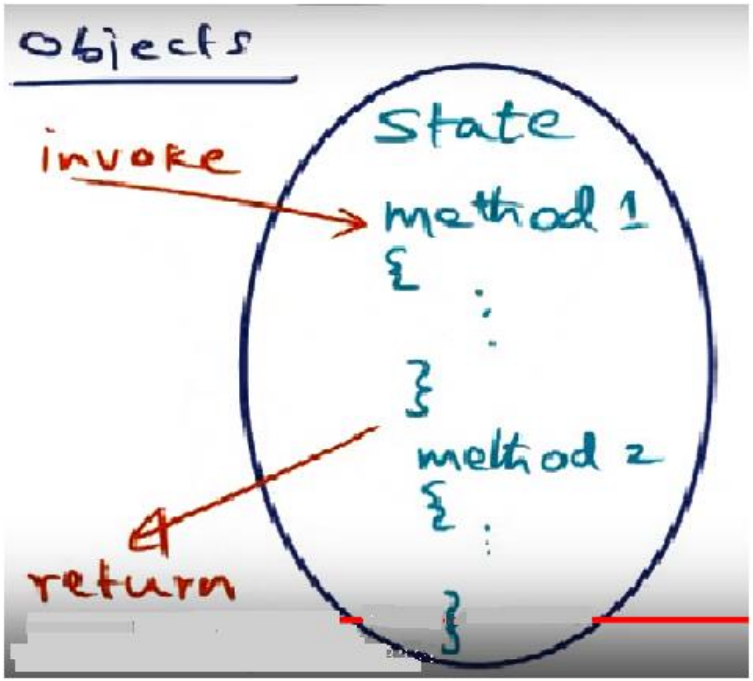
- Hata yönetimi yapılabilir.
- Veri güvenliği
- Karmaşıklık azaltılmıştır
- Kod ekleme, programı genişletme daha kolaydır

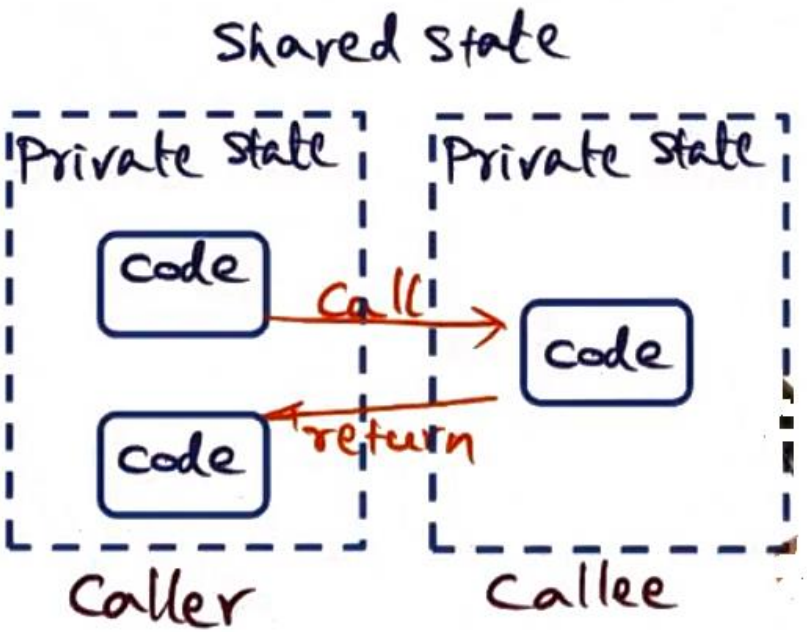
Nesneye Yönelik Programlama Java Örneği

```
Picture paper = new Picture (480, 640);
SimplePencil pencil = new SimplePencil (100, 100, paper);
```

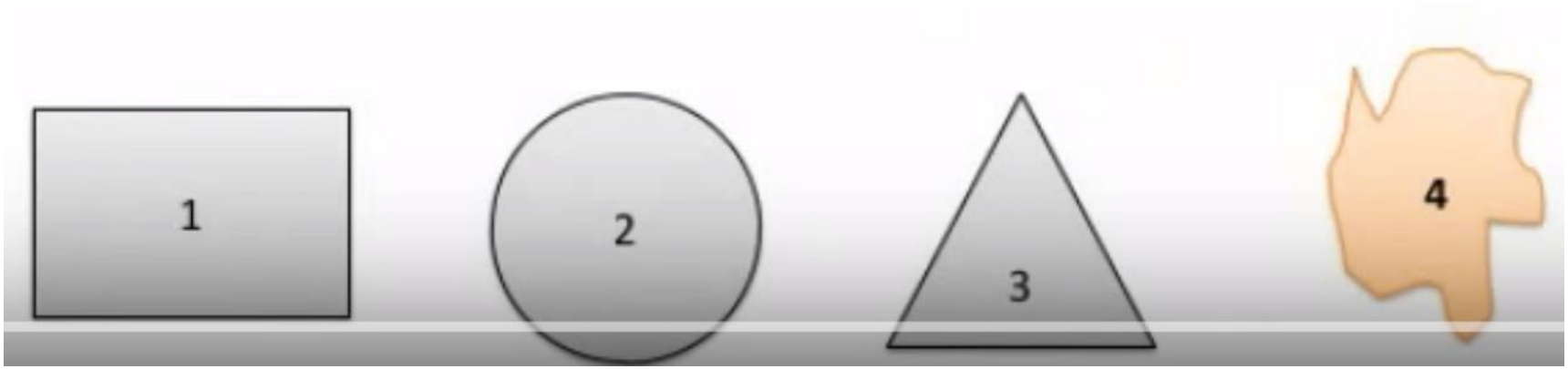

GeomFigure
x : Integer
y : Integer
display()
remove()
setPosition()
Data
Code
Circle
radius : double
setRadius()
Retangle
a : Integer
b : Integer
setA()
setB()

Yordamsal Programlama vs. Nesneye Yönelik Programlama





Senaryo: Sisteme bir Şekil Daha Eklenirse



Yordamsal Programlama vs. Nesneye Yönelik Programlama

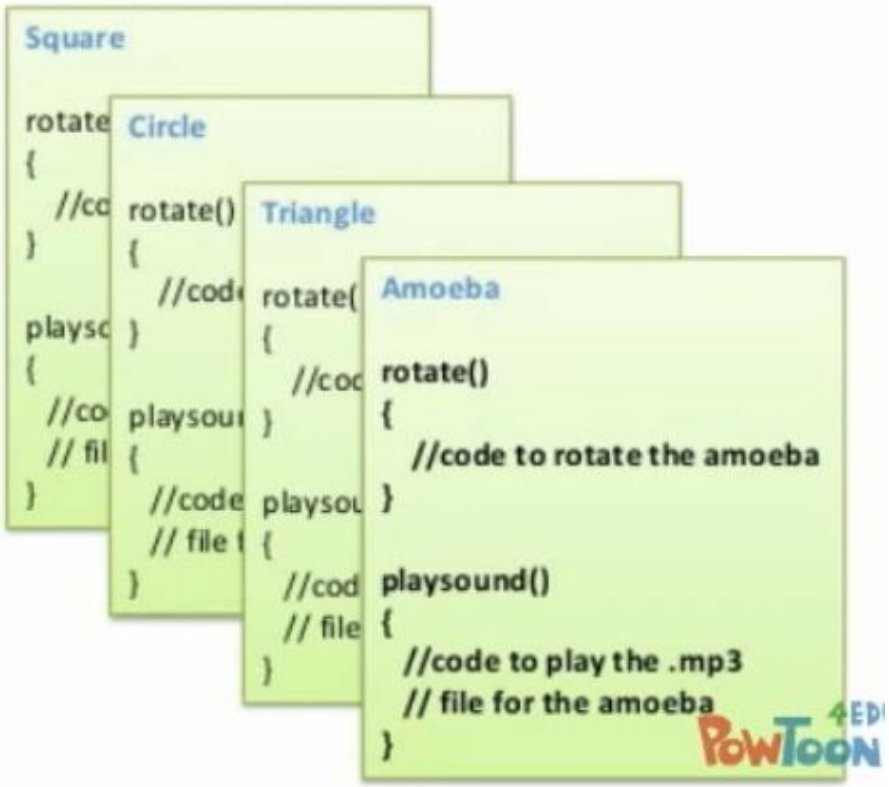
Procedural programming

```
rotate(shapeNum)
{
  //make the shape rotate 360 degrees
}

playsound(shapeNum)
{
  //if the shape is not an amoeba use
  //shapeNum to look up which .wav
  // file to play it
// else
  //play amoeba    .mp3 file
}
```

rotate() will still work but the **playsound()** will need to be changed

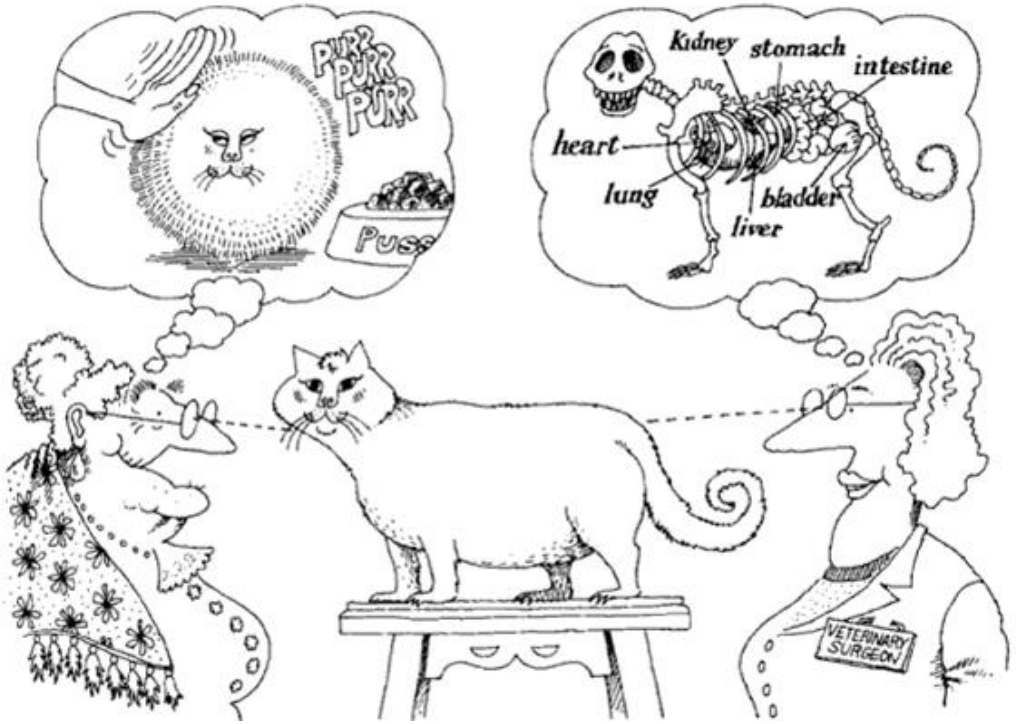
OO programming



Nesneye yönelik programlama ilkeleri

- Soyutlama

  Soyutlama" önemli özelliklere odaklanabilmek için ayrıntıları göz ardı etme sürecidir. Bu ayrıntılar yapılacak işe göre değişir.
- Saklama (Encapsulation) Bilgiyi gizlemedir.
- Kalıtım (Inheritance) Bilgiyi üst sınıflardan kullanmadır.

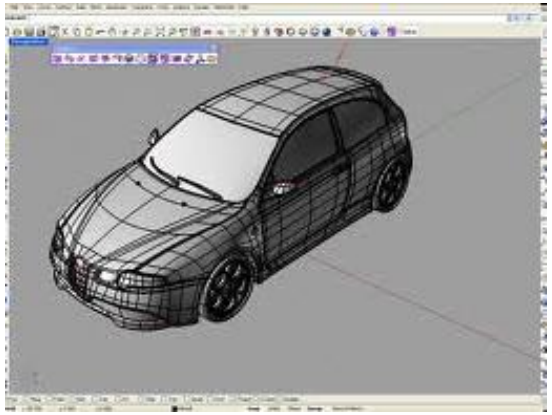


Model Nedir?

- Modelleme bir mühendislik tekniğidir.
- Model sayesinde anlaşılması güç yazılımları basit bir dille ifade edebiliriz.
- Bu da yazılımın anlaşılmasını kolaylaştırır ve hataları kolaylıkla görüp en düşük seviyeye indirgememizi sağlayacaktır.

MODELLEME NEDİR?

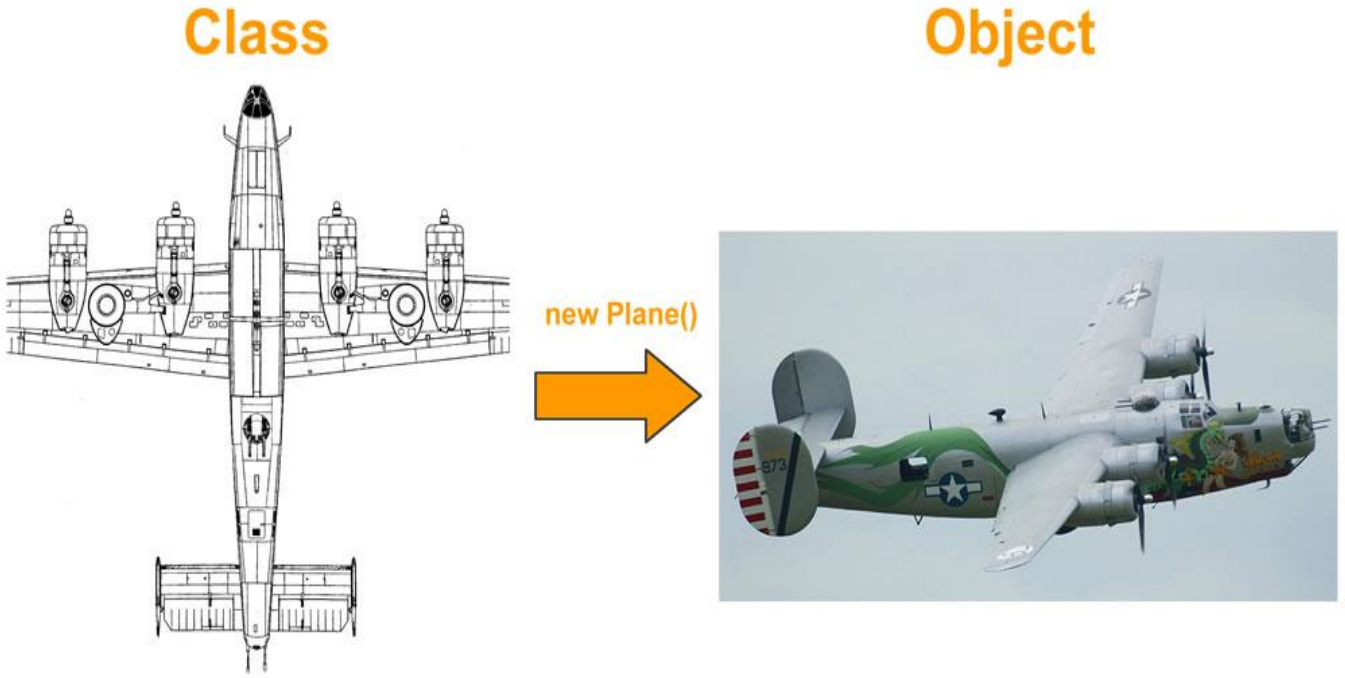
- Modelleme bir sistemi incelemek üzere o sistemin basit bir örneği yapılması anlamına gelir. Bu örnek gerçek sistemin yardımcısı ve basitleştirilmiş bir şeklidir.
- Modelleme sistemlerin karmaşıklığını çözümlemede kullanılan en eski ve en etkin yöntemdir.
- Modeller gerçek dünyadaki örneklerinin yerini alamazlar, ancak gerçek olay veya sistemin karmaşık yapısının anlaşılabilir parçalara indirgenmesinde yararlı olurlar.



MODELLEME NEDİR?

- Bir sistem modellenirken farklı bakış açılarıyla tekrar tekrar incelenir.
- Bu inceleme sırasında modellemeyi yapan kimse sistemin özelliklerinden o anda ilgilendiklerini öne çıkarırken diğerlerini göz ardı edebilir.
- Sonuçta oluşan bu soyut yapı sistemin ilgilenilen özelliklerinin bir modeli olur.
- Hiçbir model gerçek sistemin özelliklerini tümüyle içermez.

Class-Object



Yazılımda sistemin modellenmesi

- Yazılım projelerinde yer alan proje yöneticileri, müşteriler, çözümleyiciler, tasarımcılar, programcılar, testçiler ve teknik yazarlardan her birinin eğitim düzeyleri ve alt yapıları farklıdır.
- Bir yazılım sistemin modellenmesi süreci aşamaları:
  - ✓ Sistem analizi
  - ✓ Sistem tasarımı
  - ✓ Kodlama
- Eğer bir sistem, tüm proje ekibinin anlayabileceği ortak bir dille modellenirse, çok karmaşık anlatımlar basitleşebilir ve aralarındaki iletişim çeşitli diyagramlarla maksimum düzeyde tutulabilir.

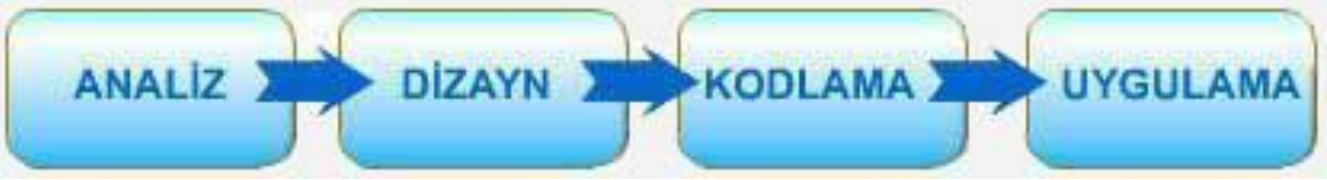
UML

Unified Modelling Language (Bütünleşik Modelleme Dili)

- Nesneye yönelik sistemlerin analiz ve tasarımında standart olarak kullanılan modelleme dili UML'dir.
- UML bir programlama dili değildir. Bir diyagram çizme ve ilişkisel modelleme dilidir.
- Yazılım mühendisliğinde nesne tabanlı modellemede kullanılan standart olmuş görsel modelleme dilidir.
- UML 1.0, taslak olarak 1997 de tanıtıldı.
- Yazılım geliştirmenin çözümlemeden bakıma kadar tüm aşamalarında ekipler ve bireyler arasındaki iletişimin düzgün yürütülmesi için kullanılmaktadır.

UML’in Rolü

- Bir sistemin geliştirilmesi kabaca aşağıdaki



Programın analiz ve dizayn aşamasında UML'e büyük ölçüde ihtiyaç duyulmaktadır.

UML'in Rolü

- Yazılım yaşam döngüsü içerisinde farklı görev tanımlamaları bulunmaktadır.
  - Analistler,
  - tasarımcılar,
  - programcılar,
  - testçiler,
  - kalite sorumluları,
  - müşteriler / kullanıcılar,
- Her birinin sisteme yada projeye bakış açısı birbirinden farklıdır.

UML'in Rolü

- Müşteri açısından projeye baktığımızda müşteriyi işlerin sıralandırılması, sisteme artıları ve eksileri , işler arasındaki ilişkiler ilgilendirirken bir fonsiyonun detayları ilgilendirmemektedir.
- Analist açısından baktığımızda nesne özellikleri, fonksiyonlar ve alacakları parametreler yeterli iken,
- tasarımcı açısından parametrelerin veri tipleri, fonsiyonun performansı, yaşam süresi gibi bilgiler de önemli olmaktadır.

Bu nedenle UML bu ekip için gerekli farklı diyagramlar içermektedir. Yazılım geliştirme işinde yer alacak farklı ekiplerin farklı bakış açılarına uygun farklı UML diyagramları bulunmaktadır. UML, yazılım geliştirmede analiz ve dizayn aşamalarında büyük rol oynamaktadır.

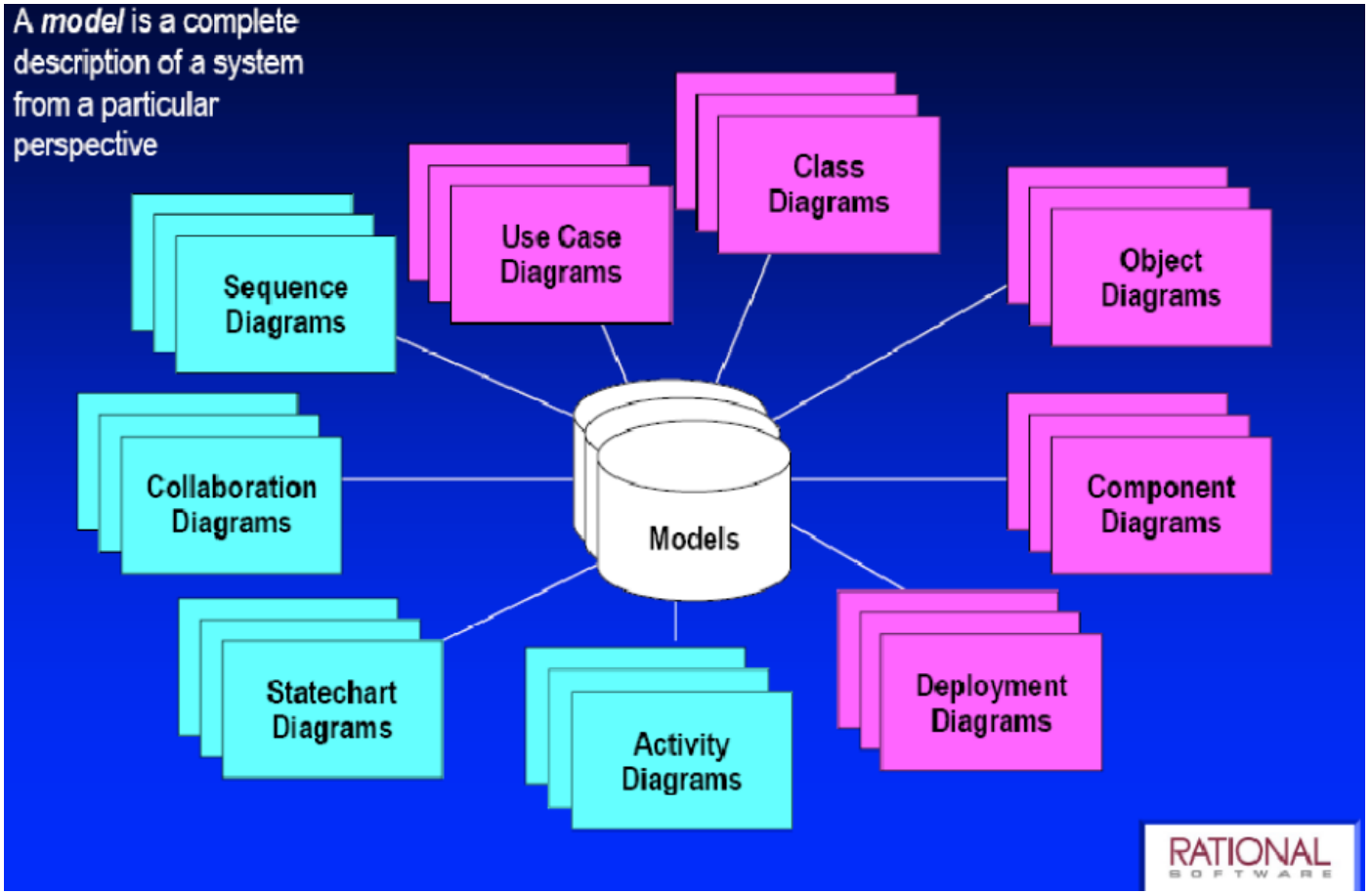
UML'YE NEDEN GEREK VAR?

- Hataların kolaylıkla fark edilip en düşük seviyeye indirgenmesi.(Risk, zaman, maliyet)
- Yazılım üretiminde başarı oranını yükseltme.
- Yazılımda paylaşım önemlidir. Tüm ekibin aynı dili konuşabilmesi gerekmektedir.
- Sistemin tamamını basit bir dille ve görsellikle görebilmek ve tasarlayabilmek gerekli.
- Modellenmiş ve dokümante edilmiş bir yazılımın tanıtımının kolay olması.
- Yazılım kalitesini arttırma.

UML'NİN AVANTAJLARI

- Kodlama kolaylığı sağlar. UML ile uygulamanızın tasarımı analiz aşamasında yapıldığı için, modellemeniz bittikten hemen sonra kod yazmaya başlayabilirsiniz.
- Kullanılan tekrar kod sayısı ayırt edilebilir bu sayede verim sağlanır.
- Mantıksal hataların minimum seviyeye düşürülmesini sağlar. Bütün sistem tasarlandığı için oluşabilecek hataların düzeltilmesi de daha kolaydır.
- Geliştirme maliyetinin düşmesini sağlar.
- UML diyagramları ile yazılım tamamını görebileceğimiz için verimli bellek kullanımı sağlanabilir.
- Karmaşık sistemlerde değişiklik yapmayı kolaylaştırır.
- UML diyagramlarını kullanan yazılımcılar aynı dili konuşacaklarından kolay iletişim sağlanır.

UML DİYAGRAMLARI



UML DİYAGRAMLARI

- UML, modelleme için değişik diyagramlar kullanır. Diyagramlar, bir sistem modelini kısmen tarif eden grafiklerdir.
- UML 2.0, 3 bölümde incelenen 13 farklı diyagram içerir.
  - Yapısal diyagramlarda modellenen sistemde nelerin var olması gerektiği vurgulanır.
  - Davranış diyagramlarında modellenen sistemde nelerin meydana gelmesi gerektiğini belirtir.
  - Davranış diyagramlarının bir alt kümesi olan Etkileşim diyagramlarında ise modellenen sistemdeki elemanlar arasındaki veri ve komut akışı gösterilir.

Davranış Diyagramları

- Kullanıcı Senaryosu (Use-Case) diyagramı
- Durum (Statechart) diyagramı
- Etkinlik (Activity) diyagramı

Davranış Diyagramları

- **Use-case diagram**
  - Programımızın davranışının bir kullanıcı gözüyle incelenmesi Use Case diyagramlarıyla yapılır.
  - Gerçek dünyada insanların kullanacağı bir sistemde bu diyagramlar büyük önem taşırlar.

- **State diagram**

  Gerçek nesnelerin herhangi bir zaman içindeki durumunu gösteren diyagramlardır.Mesela, Ali nesnesi insan sınıfının gerçek bir örneği olsun. Ali 'nin doğması, büyümesi, gençliği ve ölmesi State Diagram 'larıyla gösterilir.

- **Activity diagram**

  Bir nesnesinin durumu zamanla kullanıcı tarafından ya da nesnenin kendi içsel işlevleri tarafından değişebilir.Bu değişim sırasını activity diyagramlarıyla gösteririz.

Yapısal Diyagramlar

- Sınıf (Class) diyagramı
- Nesne (Object) diyagramı
- Bileşen (Component) diyagramı
- Paket (Package) diyagramı
- Dağılım (Deployment) diyagramı
- Birleşik Yapı (Composite Structure) diyagramı

Yapısal Diyagramlar

- **Class diagram**

  Gerçek dünyada eşyaları nasıl araba, masa, bilgisayar şeklinde sınıflandırıyorsak yazılımda da birtakım benzer özelliklere ve fiillere sahip gruplar oluştururuz. Bunlara "Class"(sınıf) denir.

- **Object diagram**

  Bir nesne(object) sınıfın (class) bir örneğidir. Bu tür diyagramlarda sınıfın yerine gerçek nesneler kullanılır.

- **Component diagram**

  Özellikle birden çok geliştiricinin yürüttüğü projelerde sistemi component dediğimiz parçalara ayırmak, geliştirmeyi kolaylaştırır.Sistemi öyle modellememiz gerekir ki her geliştirici ötekinden bağımsız olarak çalışabilsin.Bu tür modellemeler Component Diyagramlarıyla yapılır.

- **Deployment diagram**

  Bu tür diyagramlarla sistemin fiziksel incelenmesi yapılır. Mesela bilgisayarlar arasındaki baglantılar, programın kurulacağı makinalar ve sistemimizdeki bütün aletler Deployment Diyagramında gösterilir.

Etkileşim Diyagramları

- **Collaboration diagram**

  Bir sistemin amacının yerine gelmesi için sistemin bütün parçaları işlerini yerine getirmesi gerekir. Bu işler genellikle birkaç parçanın beraber çalışmasıyla mümkün olabilir. Bu tür ilişkileri göstermek için Collaboration Diyagramları gösterilir.

- **Sequence diagram**

  Class ve Object diyagramları statik bilgiyi modeller.Halbuki gerçek zamanlı sistemlerde zaman içinde değişen interaktiviteler bu diyagramlarla gösterilemez. Bu tür zamanla değişen durumları belirtmek için sequence diyagramları kullanılır.

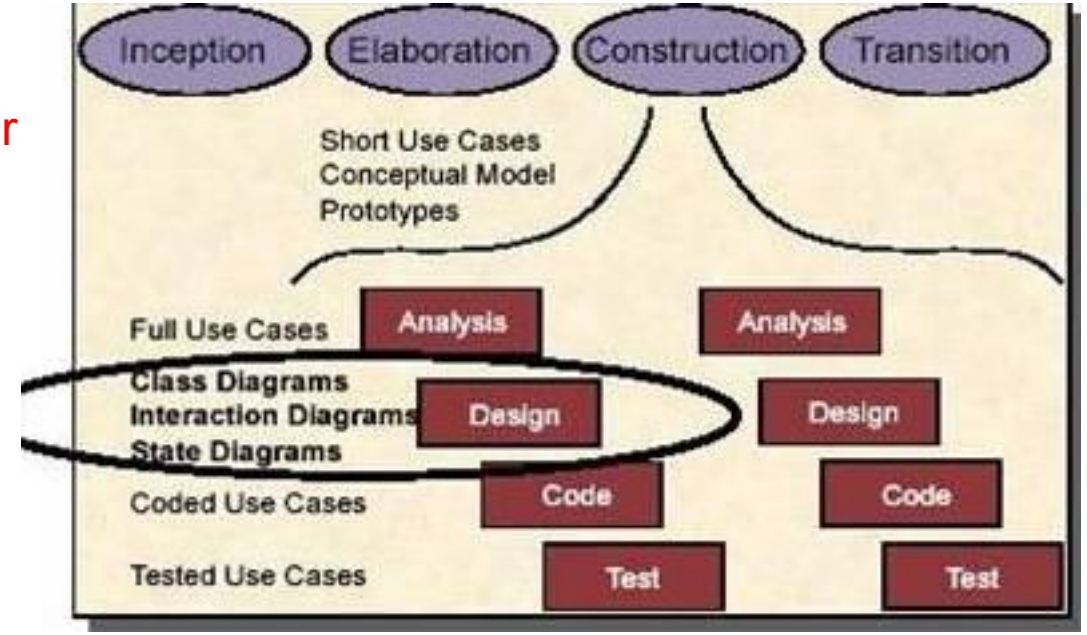
❖UML Diyagramları

- Analiz aşamasında ;

1. Use Case Diyagramları kullanılır.

- Tasarım aşamasında ise;

1. Sınıf Diyagramları
2. Durum Diyagramları
3. Etkileşim Diyagramlar



Use-case diyagramları

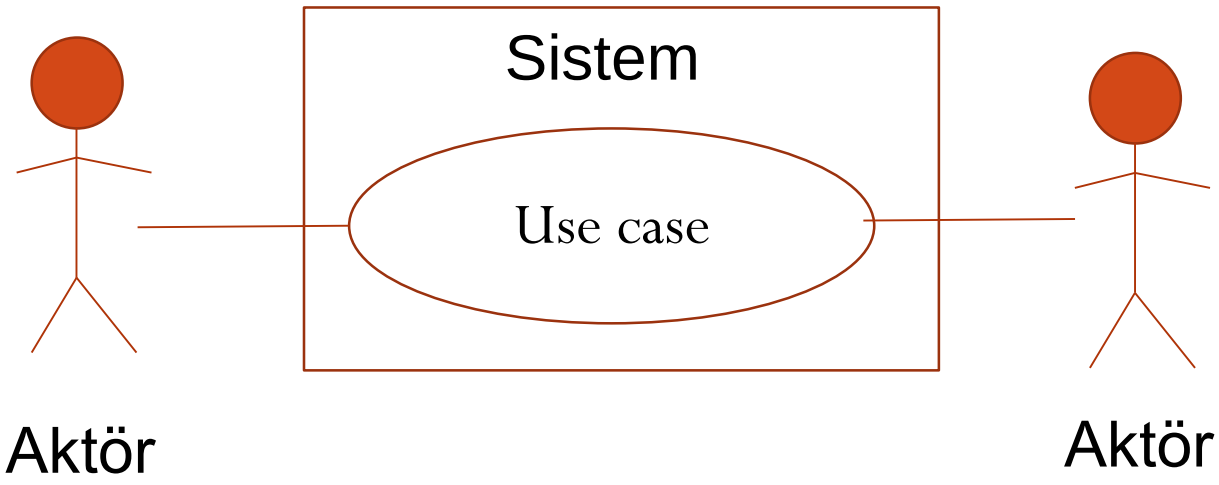
- Analistler ve uzmanlar tarafından geliştirilir.
- Sistemin çok basit bir şekilde modellenmesini ve işlerin detayının (senaryonun) metin olarak anlatılmasını içerir.

- Aktörden gelen bazı isteklere karşı sistemin yaptığı aktiviteleri gösterir.

- Amaç
  - ✓ Sistemin içeriğini belirtmek.
  - ✓ Sistemin gereksinimlerini elde etmek.

Use-case diyagramları

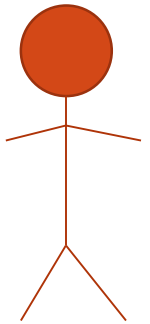
✓Bir kullanıcı ve bir sistem arasındaki etkileşimi anlatan senaryo topluluğudur. Use case diyagramlarda use case'ler ve aktörler adını verdiğimiz iki ana bileşen bulunmaktadır.



Use-case diyagramları bileşenleri

- Aktör

- Sistemin kullanıcılarıdır.
- Aktörler genelde belirli bir rol ifade ederler.



- Diğer aktörlerle bağlantılı olabilirler bu bağlantı bir ok ile gösterilir.
- Sistem sınırları dışında gösterilir.

Use-case diyagramları bileşenleri

- Use case

  - Sistemin destekleyeceği işlerdir
  - Sistem fonksiyonelliğinin büyük bir parçasını gösterir.
  - Diğer bir use case ile genişletilebilir.
  - Diğer bir use case içerebilir.
  - Sistem sınırları içinde gösterilir.

Use-case diyagramları bileşenleri

Sistem sınırı

- İçerisinde sistemin ismi yazılıdır.
- Sistemin kapsamını gösterir.

Bağıntı ilişkisi

- Aktör ve use case'ler arasındaki bağıntıyı gösteren çizgidir.

Use-case diyagramları bağıntıları

Inclusion (içerme) ilişkisi

- Bu metotla bir use case içindeki adımlardan birini başka bir use case içinde kullanabiliriz.
- Bir “use-case”in diğerinin davranışını içermesi.
- Kullanmak istediğimiz use case 'ler arasına çektiğimiz noktalı çizginin üzerine <<include>> yazısını yazarız.

<<include>>

<----------

Use-case diyagramları bağıntıları

Extension (eklenti) ilişkisi

- Bu metodla varolan bir Use Case'e yeni adımlar ekleyerek yeni use case'ler yaratılır.

- Inclusion'da olduğu gibi extension'ları göstermek için yine use case'ler arasına noktalı çizgiler konur ve üzerine <<extend>> ibaresi yazılır.

<<extend>>

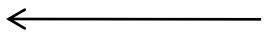
<----------

Use-case diyagramları bağıntıları

Genelleme ilişkisi:

- İki “use-case” veya iki aktör arasındaki kalıtım ilişkisidir.

  Yani özelleşmiş use case ile daha genel use case arasındaki ilişkidir.



- Özelleşmiş use case’den temel use case’e doğru bir ok ile gösterilir.

Use-case Diyagramı Oluşturmada Yöntem

Amaç: Sisteminin aktörlerini ve “use-case”lerini belirlemek ve üst seviye “use-case” modelini oluşturmak.

- Aktörler belirlenir
- “Use-case”ler belirlenir
- Her aktör ve “use case” kısaca tanımlanır
- Üst seviye “use-case” modeli tanımlanır

“Use-case”leri detaylandır.

Amaç: Belirlenen tüm “use-case”lerin is akışlarını detaylı olarak tanımlamak.

- Ana akıs tanımlanır
- Alternatif akıslar tanımlanır

Use-case Diyagramı Oluşturmada Yöntem

3. "Use-case" modelini yapılandır

Amaç: Oluşturulan use case modelini ortak noktaları en aza indirecek şekilde yapılandırmak.

- Gereken yerlerde "extend" ve "include" iliskileri kullanılabilir
- Yapılandırılan "use-case" modeli, is süreçlerini referans alınarak değerlendirilir.

4. Kullanıcı arayüzlerini tanımla

Amaç: Use case tanımları esas alınarak kullanıcı arayüzlerini üst seviyeli olarak tanımlamak.

- Kağıt üzerinde çizim yapılabilir
- Arayüz prototipleme aracı kullanılabilir

Bir örnek: ATM uygulaması

Bir bankanın ATM cihazı için yazılım geliştirilecektir. ATM, banka kartı olan müşterilerin hesaplarından para çekmelerine, hesaplarına para yatırmalarına ve hesapları arasında para transferi yapmalarına olanak sağlayacaktır. ATM, banka müşterisi ve hesapları ile ilgili bilgileri, gerektiğinde merkezi banka sisteminden alacaktır.

Bir örnek: ATM uygulaması

ATM uygulama yazılımının kullanıcıları:

- Banka müşterisi
- Merkezi Banka Sistemi

} Aktörler

Bir örnek: ATM uygulaması

Belirlenen aktörler ATM'den ne istiyorlar ?

- Aktör: Banka müşterisi
  - Para çekme
  - Para yatırma
  - Para transferi
- Aktör: Merkezi Banka Sistemi
  - Günlük özet alma

Bir örnek: ATM uygulaması

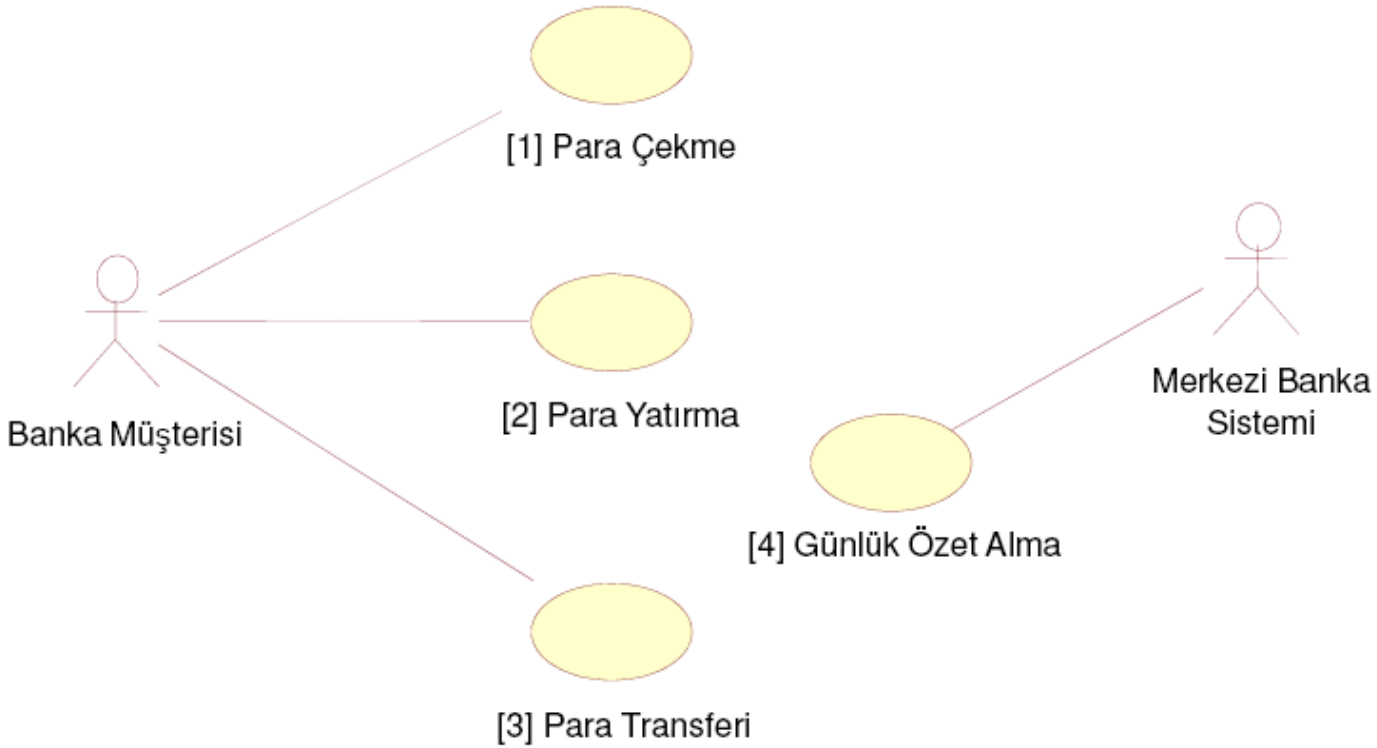
- Aktör: Banka müşterisi

  Bankada hesabı ve banka kartı olan, ATM'den işlem yapma hakkı olan kişidir.

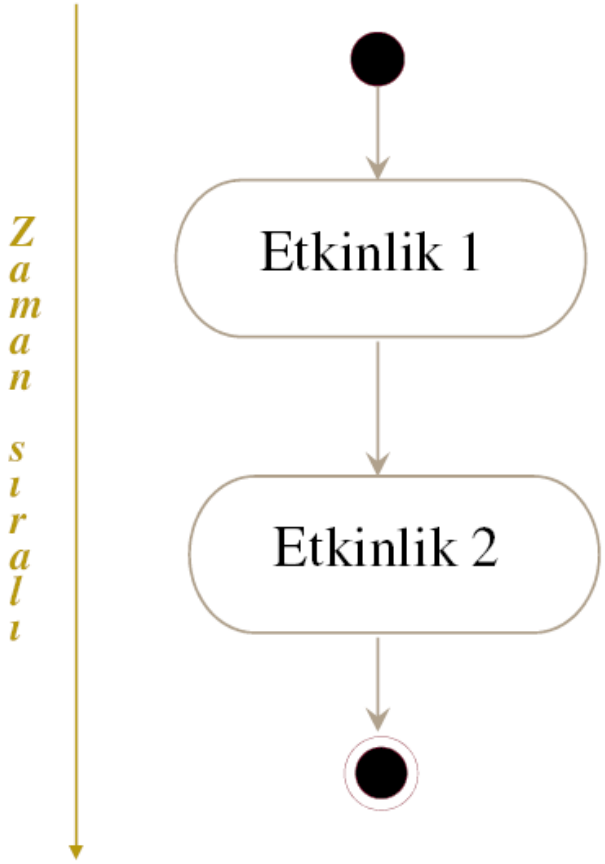
- Use case: Para çekme

  Banka müşterisinin nasıl para çekeceğini tanımlar. Para çekme işlemi sırasında banka müşterisinin istediği tutarı belirtmesi ve hesabında bu tutarın mevcut olması gerekir.

Bir örnek: ATM uygulaması (Use-case)

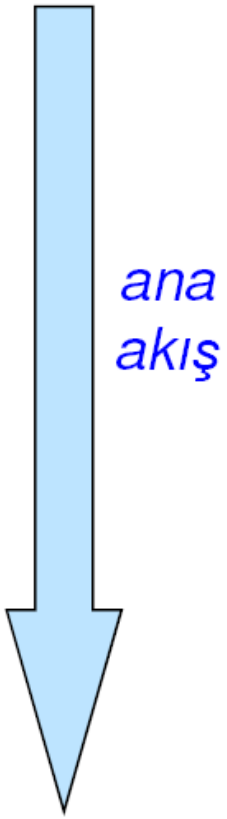


Bir örnek: ATM uygulaması (Sequence diagram)

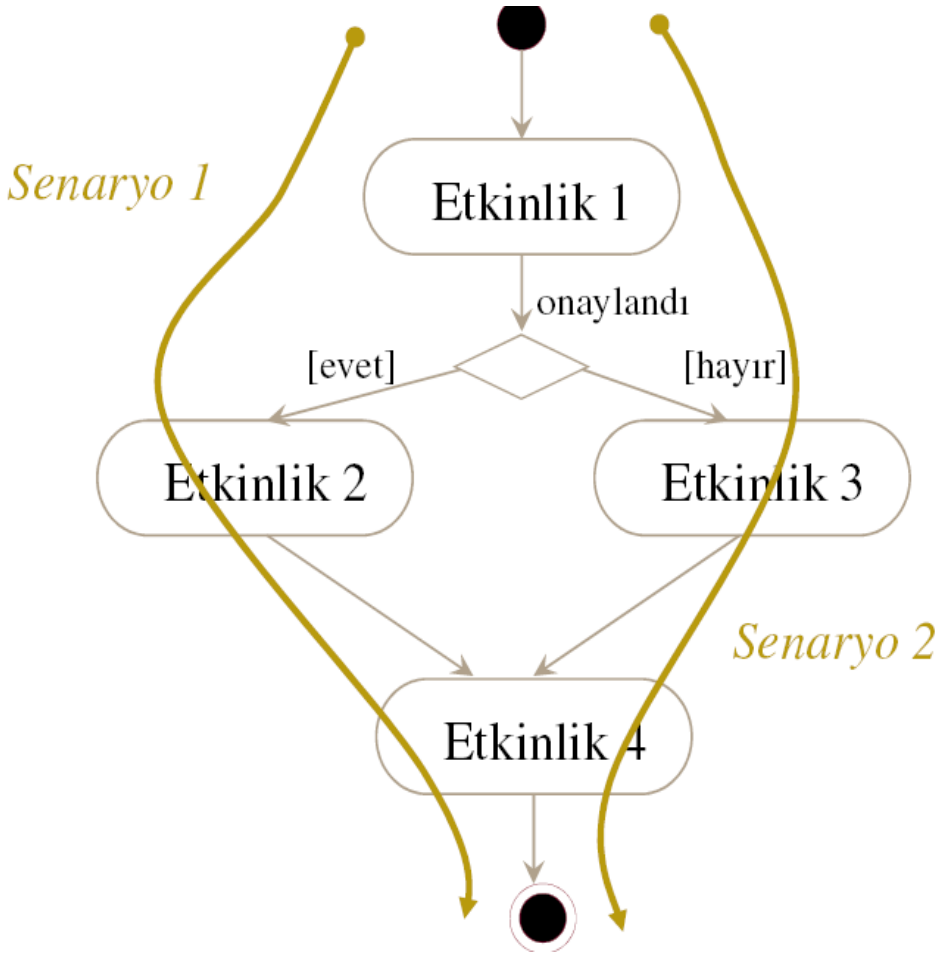


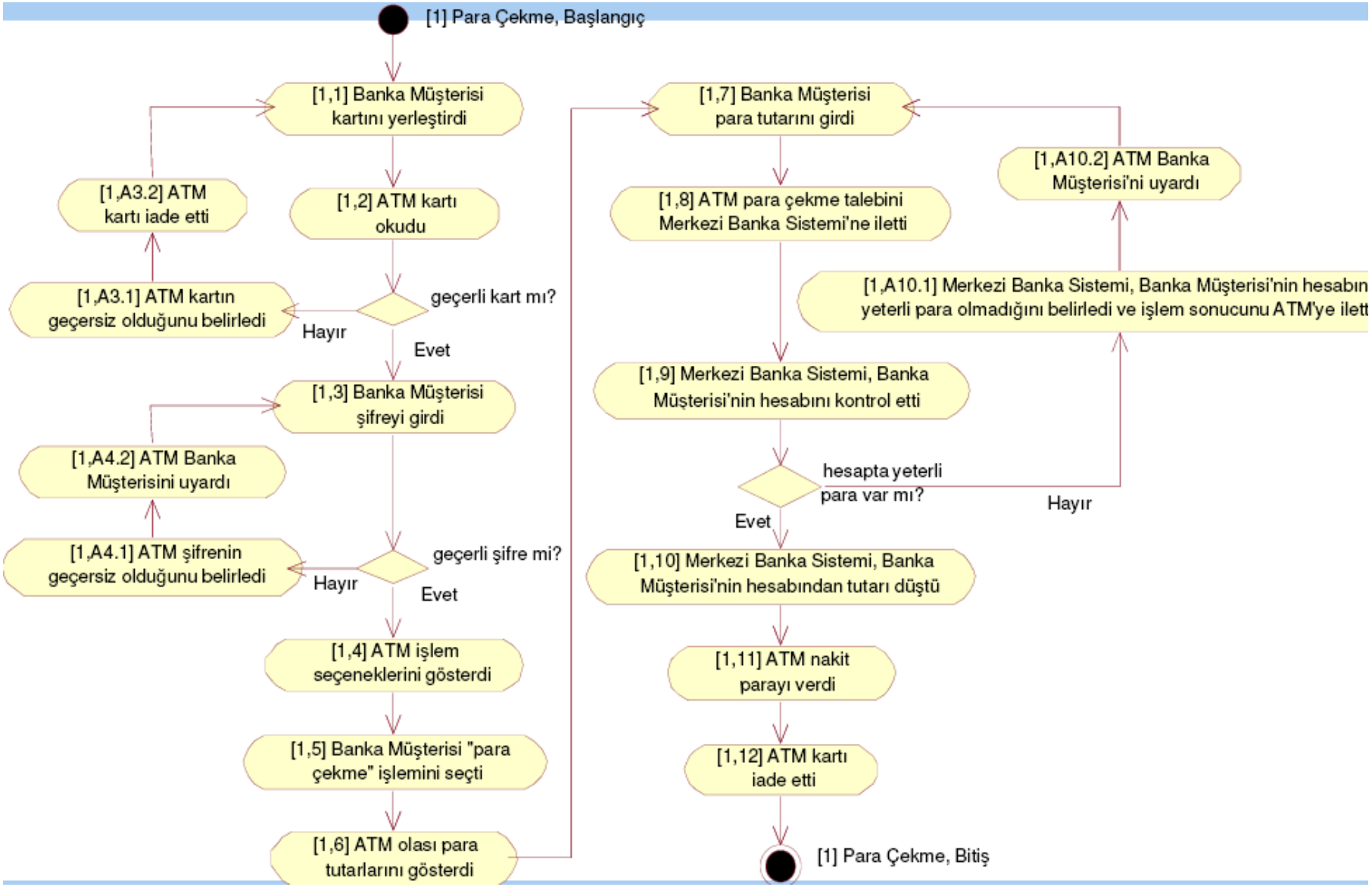
Bir örnek: ATM uygulaması (Sequence diagram)

1. Banka Müşterisi kartını yerleştirir
2. ATM kartı okur
3. Banka Müşterisi şifreyi girer
4. ATM işlem seçeneklerini gösterir
5. Banka Müşterisi “para çekme” işlemini seçer
6. ATM olası para tutarlarını gösterir
7. Banka Müşterisi para tutarını girer
8. ATM para çekme talebini Merkezi Banka Sistemi’ne iletir
9. Merkezi Banka Sistemi, Banka Müşterisi’nin hesabını kontrol eder
10. Merkezi Banka Sistemi, Banka Müşterisi’nin hesabından tutarı düştü ve işlem sonucunu ATM’ye iletir
11. ATM nakit parayı verir
12. ATM kartı iade eder

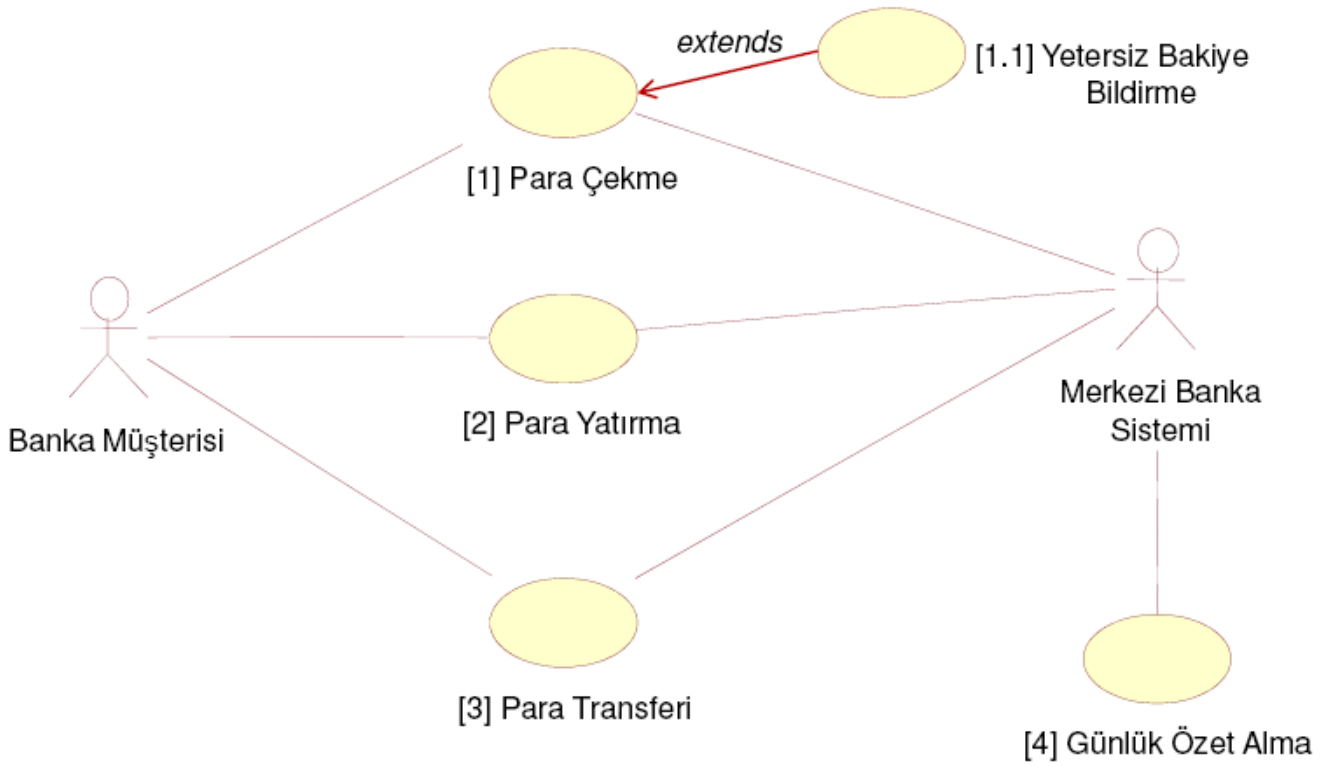


Bir örnek: ATM uygulaması

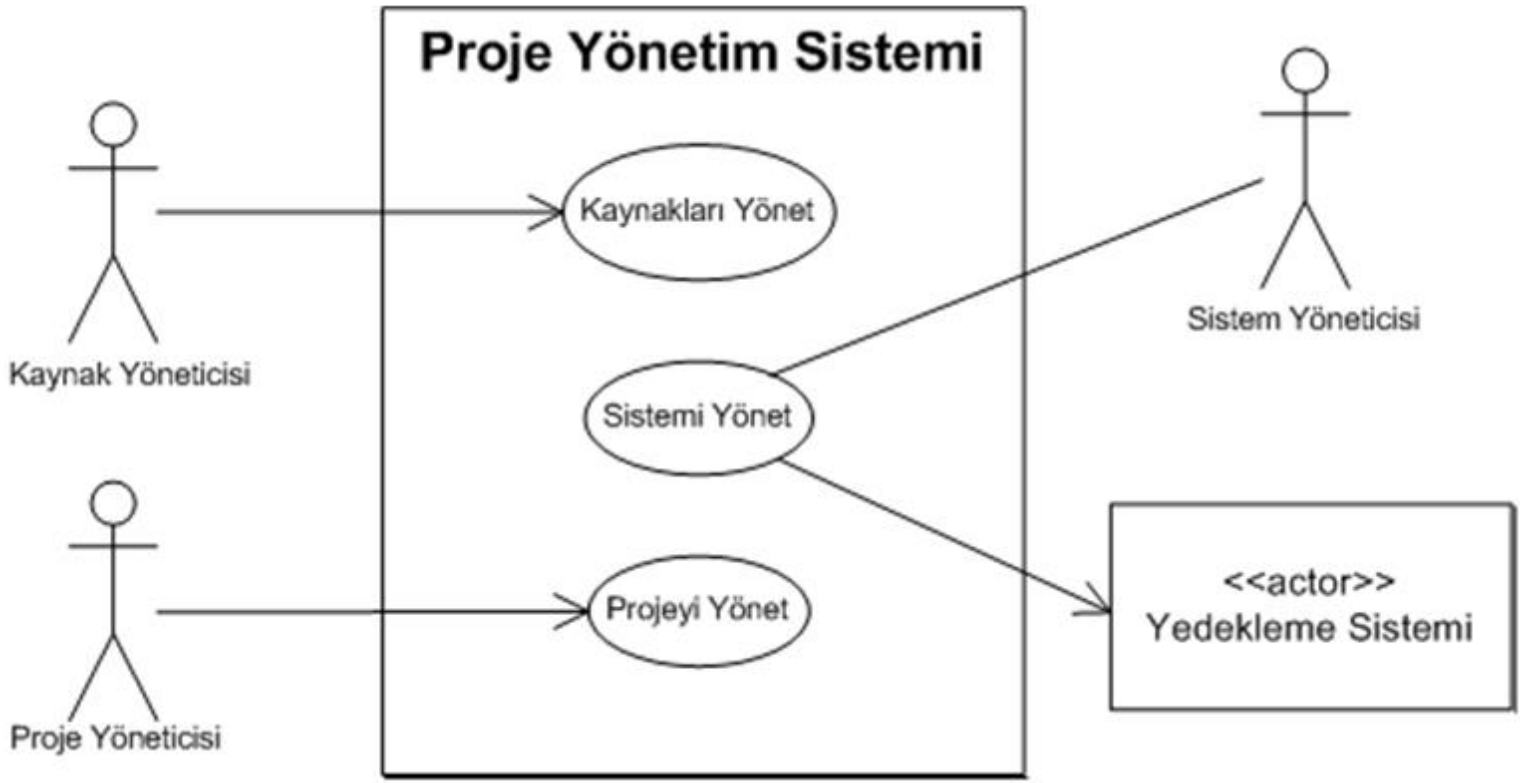



[1] Para Çekme, Başlangıç
[1,1] Banka Müşterisi kartını yerleştirdi
[1,2] ATM kartı okudu
geçerli kart mı?
Hayır
[1,A3.1] ATM kartın geçersiz olduğunu belirledi
[1,A3.2] ATM kartı iade etti
Evet
[1,3] Banka Müşterisi şifreyi girdi
geçerli şifre mi?
Hayır
[1,A4.1] ATM şifrenin geçersiz olduğunu belirledi
[1,A4.2] ATM Banka Müşterisini uyardı
Evet
[1,4] ATM işlem seçeneklerini gösterdi
[1,5] Banka Müşterisi "para çekme" işlemini seçti
[1,6] ATM olası para tutarlarını gösterdi
[1,7] Banka Müşterisi para tutarını girdi
[1,8] ATM para çekme talebini Merkezi Banka Sistemi'ne iletti
[1,9] Merkezi Banka Sistemi, Banka Müşterisi'nin hesabını kontrol etti
hesapta yeterli para var mı?
Hayır
[1,A10.1] Merkezi Banka Sistemi, Banka Müşterisi'nin hesabın yeterli para olmadığını belirledi ve işlem sonucunu ATM'ye ilett
[1,A10.2] ATM Banka Müşterisi'ni uyardı
Evet
[1,10] Merkezi Banka Sistemi, Banka Müşterisi'nin hesabından tutarı düştü
[1,11] ATM nakit parayı verdi
[1,12] ATM kartı iade etti
[1] Para Çekme, Bitiş

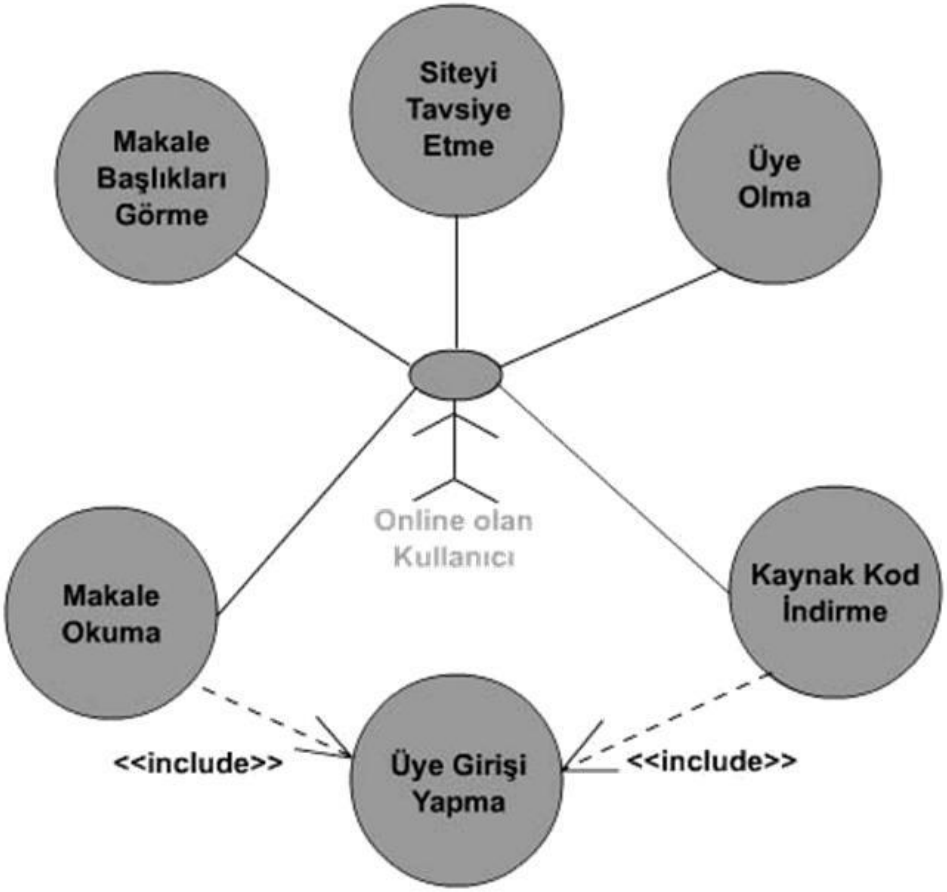
Bir örnek: ATM uygulaması (Use-case diagram)



Bir use-case diyagramı örneği



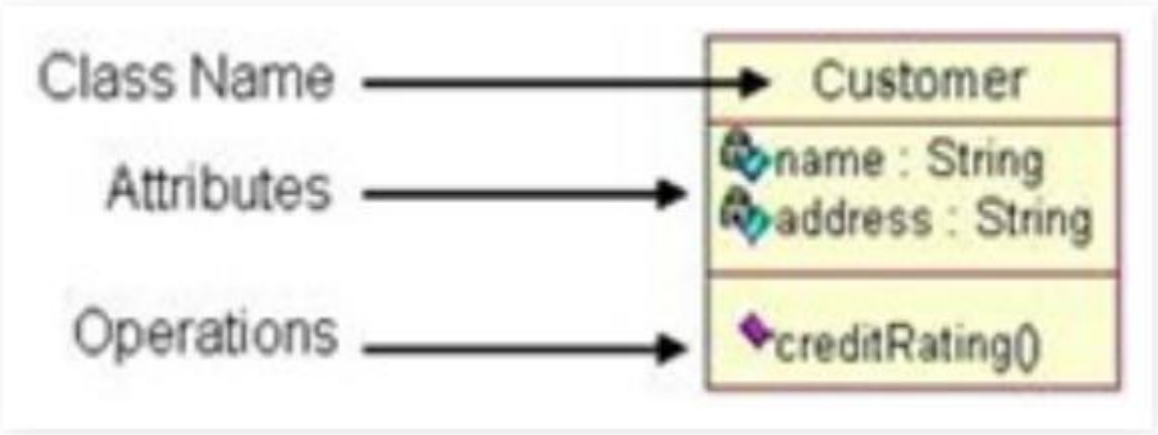
Bir use-case diyagramı örneği



Web sayfasına gelen bir kullanıcının neler yapabileceğini use case diyagramlarıyla göstermeye çalışalım. Siteye gelen bir kullanıcı kayıtsız şartsız makale başlıklarını görebilmektedir. Online olan kullanıcı Siteyi tavsiye edebilir, siteye üye olabilir,kitapları inceleyebilir. Ancak makale okuması ve kaynak kod indirebilmesi için siteye üye girişi yapmalıdır. Makale okuması ve kaynak kod indirebilmesi için gereken şart siteye üye olmaktır.Siteye bağlanan bir kullanıcının site üzerindeki hareketlerini belirtir diyagram bu şekilde oluşturulabilir.

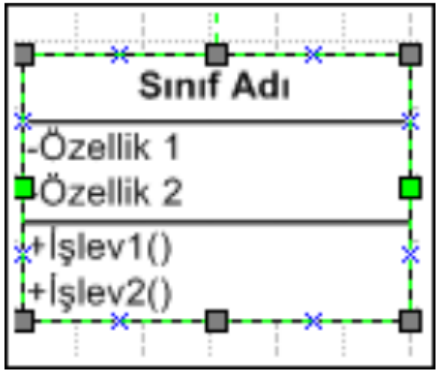
UML- Sınıf Diyagramları

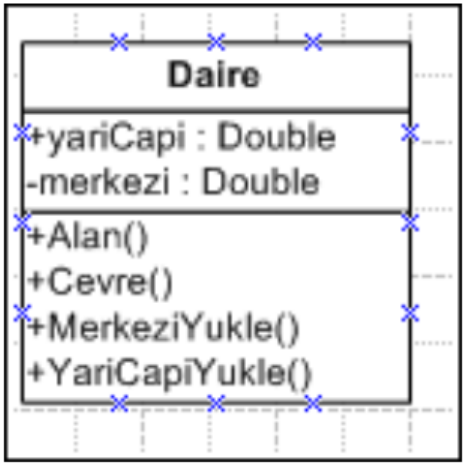
- Sınıf Diyagramları nesne tabanlı programlamada kullanılan sınıflar ve sınıfların arasındaki ilişkilerin modellenebildiği diyagramlardır.
- UML'de sınıflar, nesne tabanlı programlama mantığı ile tasarlanmıştır. Sınıf diyagramının amacı bir model içerisinde sınıfların tasvir edilmesidir.
- Nesne tabanlı uygulamada, sınıfların kendi özellikleri (üye değişkenler), işlevleri (üye fonksiyonlar) ve diğer sınıflarla ilişkileri bulunmaktadır. UML'de sınıf diyagramlarının genel gösterimi aşağıdaki gibidir.



UML- Sınıf Diyagramları

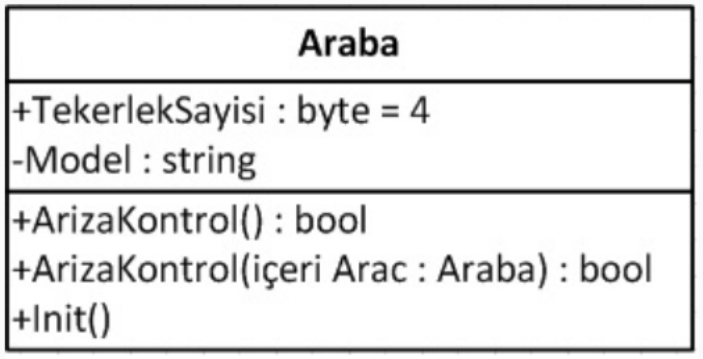
- UML de sınıf tanımlama dikdörtgen şekli ile gösterilir. Dikdörtgenin en üstünde sınıfın adı, altında özellikleri ve onunda altında fonksiyonlar gösterilir.





UML- Sınıf Diyagramları

- Sınıf diyagramında yer alan nitelik ve method isimlerinin önünde aşağıda sıralanan +/-/# (adornments) kullanılabilir.
- Private (-); Nitelik yada metota sınıf dışında erişim engellenmiştir.
- Protected (#); Nitelik yada metota erişim sınırlandırılmıştır.
- Public (+); Nitelik yada metot genel kullanıma açıktır.



UML- Sınıf Diyagramları

Sınıflar Arası İlişkiler

- Sınıf diyagramları kendi başlarına bir şey ifade etmez ancak aralarındaki ilişkilerle birlikte incelendiklerinde anlamlıdır.
- UML içerisinde sınıflar arasında 4 farklı ilişki tanımlanabilir;

1. Bağıntı İlişkisi (Association)
2. Genelleme/Kalıtım İlişkisi (Generalization/Inheritance)
3. Bağımlılık İlişkisi (Dependency) (Aggregation, Composition)
4. Gerçekleştirim İlişkisi (Realization)

UML- Sınıf Diyagramları

Sınıflar Arası İlişkiler

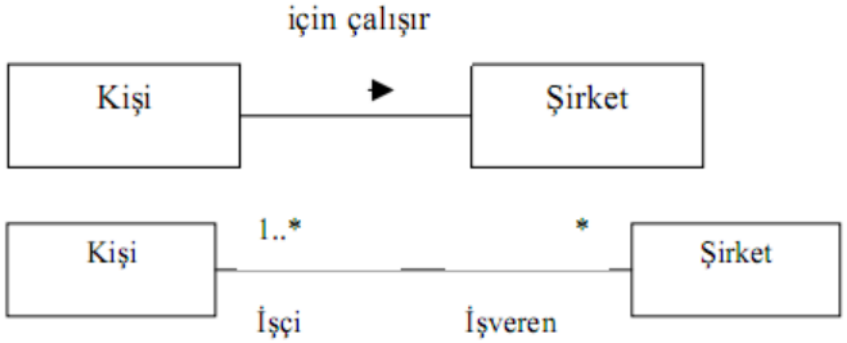
- **1. Bağıntı İlişkisi (Association Class)**
  - Bağıntı ilişkisi, sınıf diyagramlarında en çok kullanılan ve en basit ilişki türüdür. Çoğu zaman referans tutma biçimindedir.
  - Bağıntı ilişkisi iki nesne arasına çizilen düz çizgi ile belirtilir. Bağıntı ilişkileri için tanımlanmış bilgiler aşağıdaki gibidir:
    - ❖ Bağıntının Adı
    - ❖ Sınıfın bağıntıdaki rolü
    - ❖ Bağıntının çokluğu

Sınıf diyagramlarında sınıflar arasında bire n ilişki kurulabilir. Bir sınıf, n tane başka bir sınıf ile ilişkiliyse buna bire-çok (1-n) ilişki denir.

UML- Sınıf Diyagramları

Sınıflar Arası İlişkiler

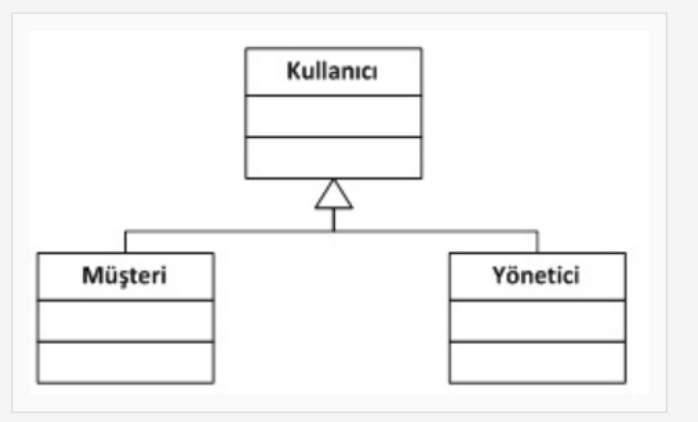
- İşçi-işveren ilişkisinde bir şirketin en az bir işçisi olduğu (1..*), bir işçinin ise 0 ya da herhangi bir sayıda(*) şirkette işçi olarak çalışmış olabileceği ifade edilmektedir. İki sınıf arasında yanlızca tek bir bağıntı çizilmesi gibi bir kısıt yoktur. En temel bağıntı ilişki tipleri aşağıdaki gibi listelenebilir;
- Bire-bir
- Bire-çok
- Bire-bir veya daha fazla
- Bire-sıfır veya bir
- Bire-sınırlı aralık (mesela:bire-[0,20]
- aralığı)
- Bire-n
- (UML de birden çok ifadesini kullanmak için '*' simgesi kullanılır.)

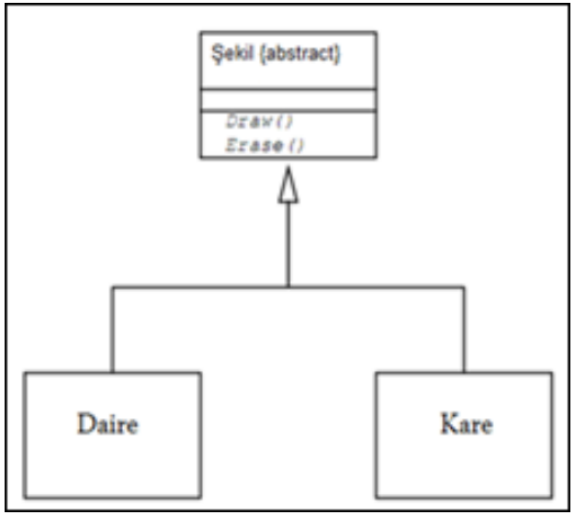


UML- Sınıf Diyagramları
Sınıflar Arası İlişkiler

- 2. Genelleme/Kalıtım İlişkisi (Generalization/Inheritance)

❖ Nesne tabanlı programlama tekniğinin en önemli parçası türetme (inheritance) dir. Türetme yoluyla bir sınıf başka bir sınıfın var olan özelliklerini alarak, o sınıf türünden başka bir nesneymiş gibi kullanılabilir.





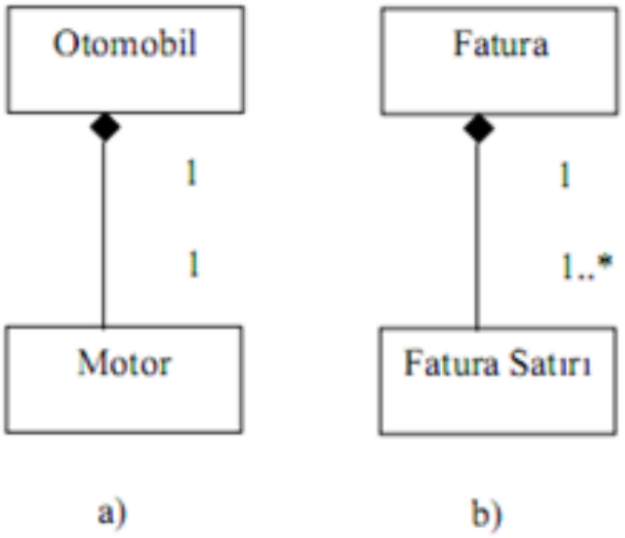
- Bir sınıfın işlevleri türetme yoluyla genişletilecekse, türetmenin yapılacağı sınıfa taban sınıf (base class), türetilmiş olan sınıfa da türemiş sınıf (derived class) denir. Şekilsel olarak türemiş sınıftan taban sınıfa bir ok olarak belirtilir.

UML- Sınıf Diyagramları
Sınıflar Arası İlişkiler

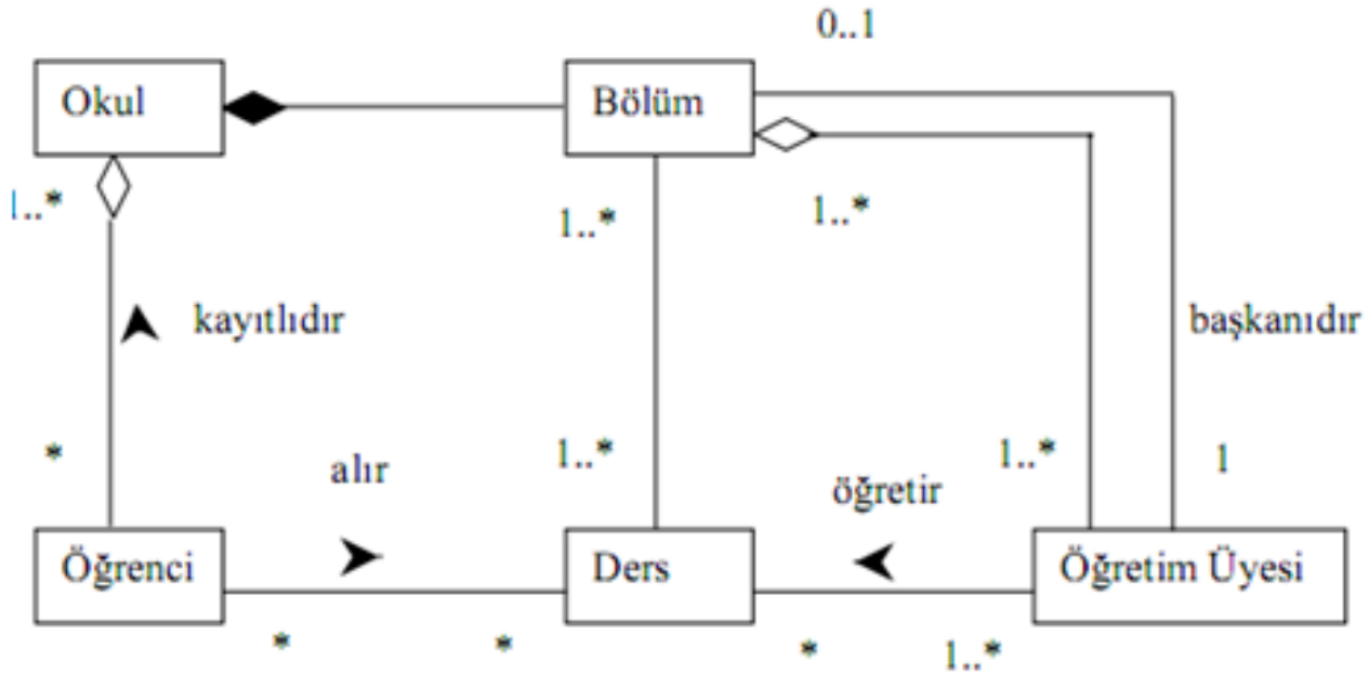
- 3.Bağımlılık İlişkisi (Dependency) (Aggregation (İçerme), Composition (Oluşum))
- *Aggregation (İçerme)*

UML- Sınıf Diyagramları
Sınıflar Arası İlişkiler

- 3.Bağımlılık İlişkisi (Dependency) (Aggregation (İçerme), Composition (Oluşum))
- *Composition (Oluşum)*

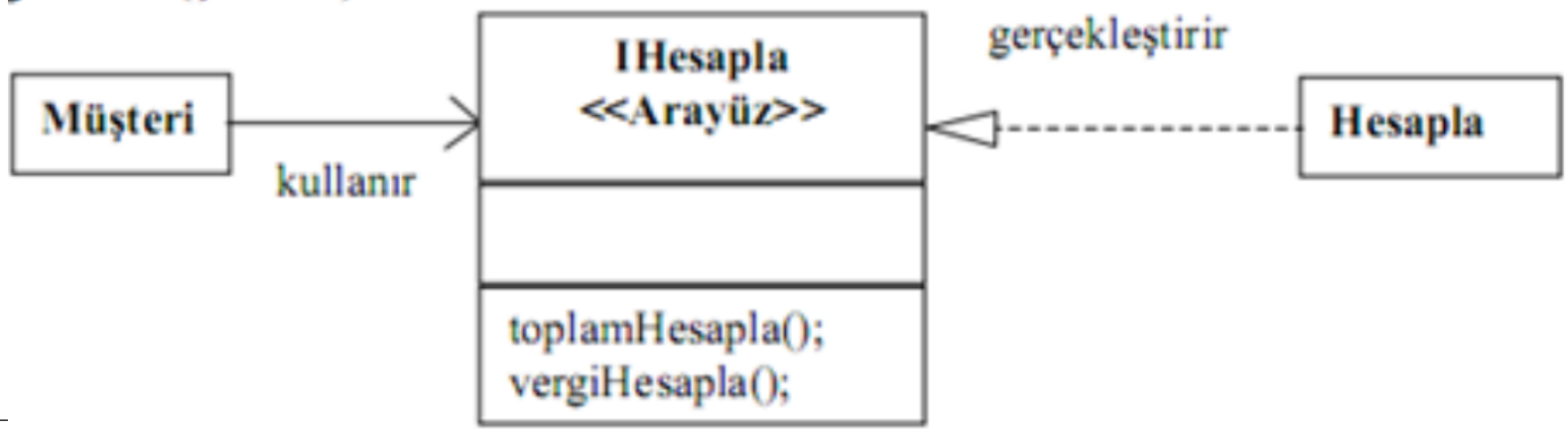


UML- Sınıf Diyagramları
Bağıntı Örneği



UML- Sınıf Diyagramları
Sınıflar Arası İlişkiler

- 4. Gerçekleştirim (Realization) İlişkisi
- Gerçekleştirim ilişkisi en çok kullanıcı arayüzlerinin (user interface) modellenmesinde kullanılır. Arayüz yalnızca metot adlarını ve bunların parametrelerini içermektedir. Program yazarken, yalnızca arayüzlerin kullanılması ve arayüzü gerçekleştiren sınıfın diğer sınıflardan ayrı tutulması, yazılımın geliştirilmesi ve bakımında önemli kolaylık sağlar. Gerçekleştirim ilişkisi kesikli bir çizginin ucuna yerleştirilen içi boş bir üçgen ile gösterilir.



Java Programlama Dili

- Java ™platformu, aynı yazılımın birçok değişik ag (network) ortamında veya değişik tür makinalarda çalışması fikri ile geliştirilmiş yeni bir teknolojidir.

- Java teknolojisi kullanılarak aynı uygulamayı değişik ortamlarda çalıştırabiliriz– örneğin Pc'lerde, Macintosh bilgisayarlarda, hatta cep telefonlarında.

“Bir Kere Yaz Her Yerde Çalıştır”

- Java uygulamaları JVM (Java Virtual Machine) tarafından yorumlanır(interpreted).
- JVM , işletim sisteminin en tepesinde bulunur.
- Java uygulamaları değişik işletim sistemlerinde, herhangi bir değişiklik yapmadan çalışabilir, Java’nın felsefesi olan“ bir kere yaz her yerde çalıştır” sözü gerçekleştirilmiştir.

Java İle Neler Yapılabilir?

- Java Programlama dili ile projelerimizi diğer programlama dillerine göre daha kolay ve sağlıklı yazmak mümkündür.
- GUI (graphical user interface , grafiksel kullanıcı arayüzü) uygulamaları, Appletler.
- Servlet, Jsp(web tabanlı uygulamalar).
- Veritabanlarına erişim ile alakalı uygulamalar.
- Cep telefonları, Smart kartlar için uygulamalar.

Java'nın Başarılı Olmasındaki Sebepler

- **Nitelikli bir programlama dili olması**
  - C++ da olduğu gibi bellek problemlerinin olmaması.
  - Nesneye yönelik (Object -Oriented) olması
  - C/C++/VB dillerinin aksine dinamik olması.
  - Güvenli olması.
  - Internet uygulamaları için elverişli **(Applet, JSP, Servlet, EJB, Corba, RMI).**
  - **Kolay ve anlaşılır**
  - **Error handling,**database access, network programming ve distributed computing.
- **Platform bağımsız olması: bir kere yaz her yerde çalıştır.**

Nesneye Yönelik Programlama Dili JAVA

- Dinamik bellek yönetimi
- Veri güvenliği
- İnternet tabanlı uygulama olanağı
- Kolay ve anlaşılır
- Hata yönetme
- Veritabanı bağlantısı kurabilme
- Ağ programlama
- Mobil programlama
- Platform bağımsız

Java Sanal Makinesi (JVM – Java Virtual Machine)

- Sanal makineyi yöneten işlemci.
- Java derleyicisi baytkod üretir.
- JVM, baytkod komut kümesini adım adım işler, bundan dolayı yorumlanan (interpreted) bir dildir.
- JVM, class dosyası içerisindeki her bytecode satırını üzerinde çalıştığı platforma uygun bir biçimde makine koduna dönüştürerek çalışmasını sağlar.
- JVM, her ne kadar sanal da olsa sonuçta kendine ait bir komut seti olan, bu komut setine uygun komutları (bytecode) çalıştırabilen bir makinedir. Bu açıdan bakıldığında JVM, komut seti bytecode olan bir işlemcidir.
- Yani teorik olarak JVM üzerinde sadece Java programlarını değil, hangi dilde yazılırsa yazılsın bytecode formatına çevirebileceğiniz her programı çalıştırabilisiniz.

Kaynak Kod Çalışma Evreleri

Derleme anı (Compile time)

Çalıştırma anı (Run time)