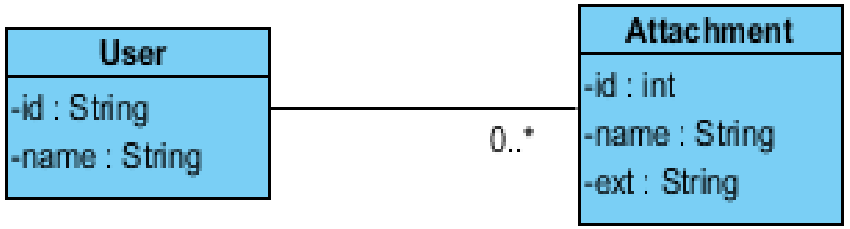
# Yazılım Testi ve Proje Yönetimi

## Sınıf/Nesne Diyagramları ve Aralarındaki İlişkiler

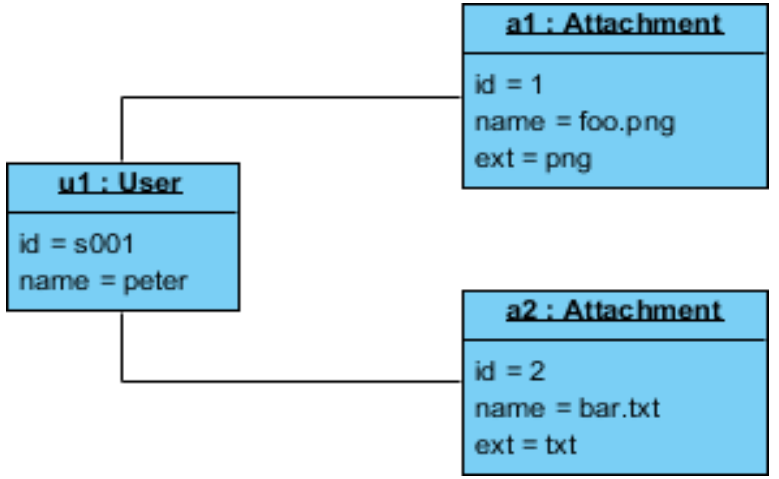
18.03.2024

# UML İle Görsel Çözümlemede Mantıksal Görünüm: Class, Object, Package, State Diagram

- Sistemin uygulama birimleri ile nasıl yapılandırıldığı açıklanır.
  - Öğeleri paketler, sınıflar ve arayüzlerdir.
  - Öğeler arasındaki ilişki; bağımlılıkları /dependiences , arayüz gerçekleşmelerini /interface realizations , parça-bütün ilişkilerin /part-whole relationships verir.




Class diyagram

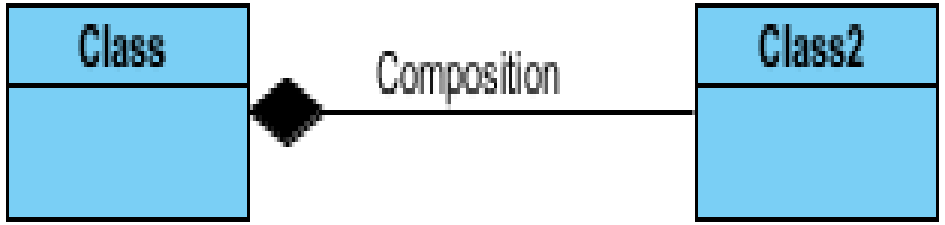



Object diyagram

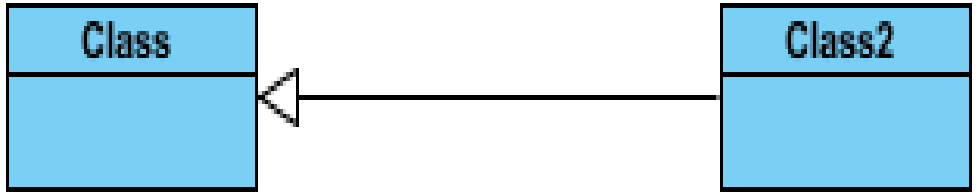
# Sınıflar arası İlişkiler


Bir modeldeki iki sınıfın birbiriyle iletişim kurması gerekiyorsa, aralarında bir bağlantı olmalıdır ve bu bir ilişki bağlayıcı /connector ile ifade edilebilir.


Child class, parent class dan bağımsız olarak vardır. Sınıf (parent class) ve Öğrenci (child class) ise, Sınıf silindiğinde Öğrenciler hala vardır.
Ama child class, parent class olmadığında anlamlı olmayacaktır.

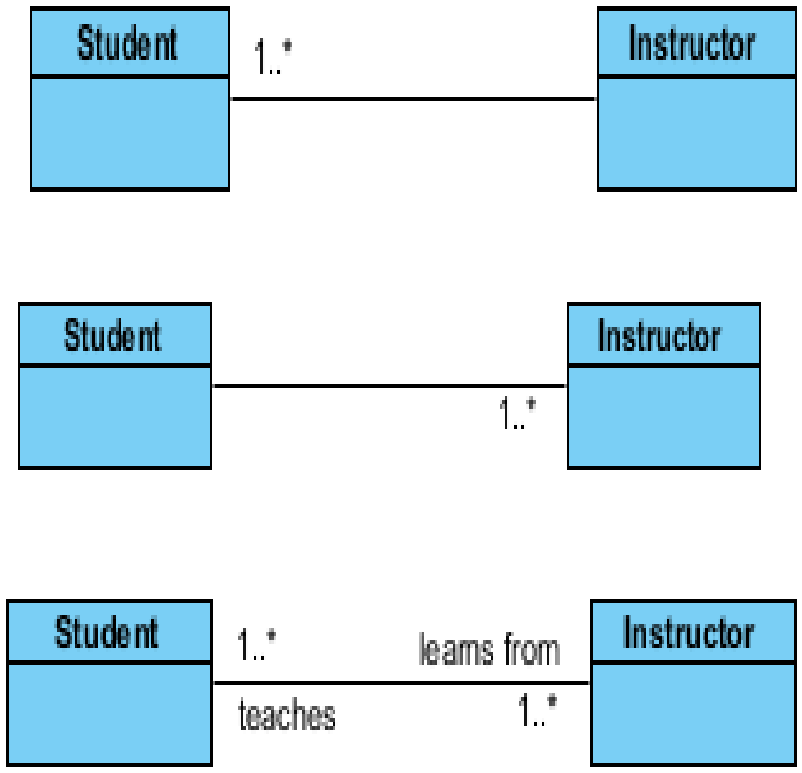


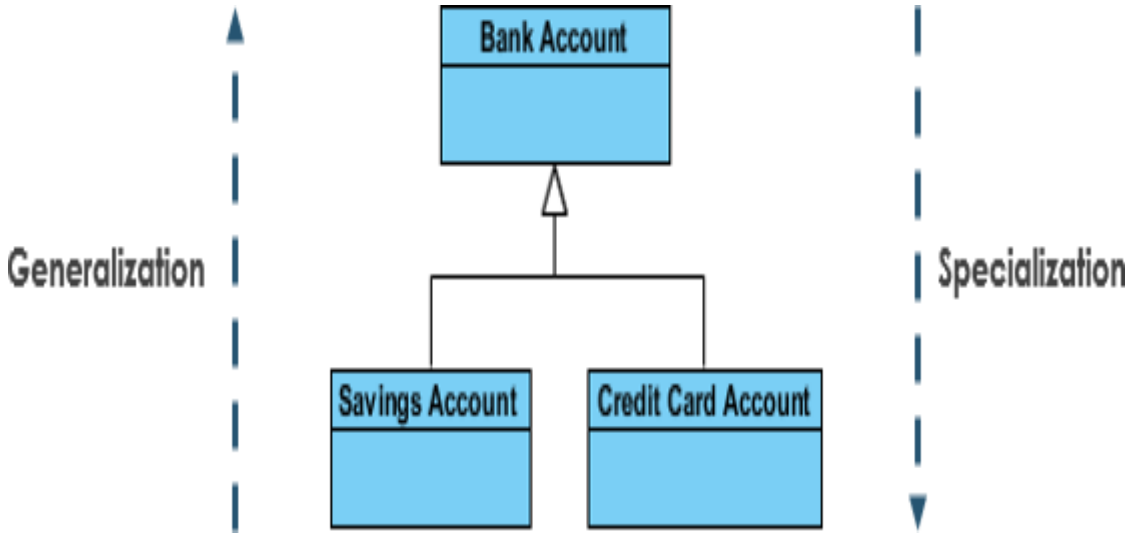

Child class, parent class dan bağımsız olarak var olamaz.
Ev (Parent class ) ve Oda (child class) ise, odalar bir evden ayrı olarak mevcut olamaz.




Genelleştirme /miras ilişkisi

# «Association» ilişkisinde «multiplicity» ve sınıflar arasında Genelleştirme ve Özelleştirme ilişkisi



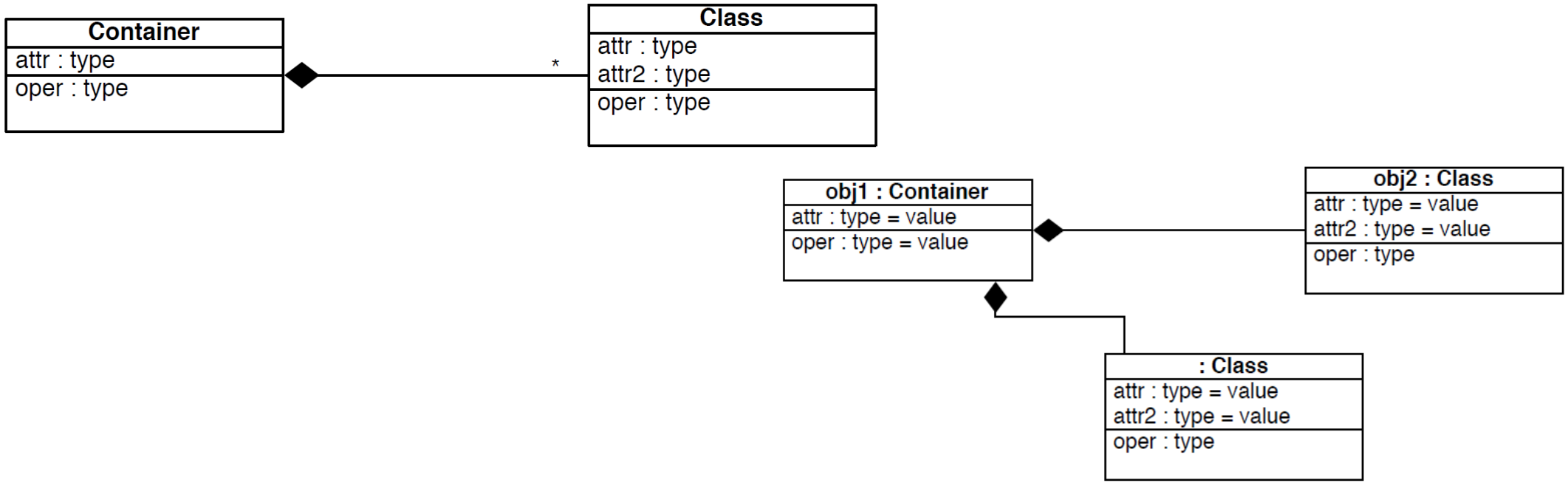



# Encapsulation /Kapsülleme

Sınıflar, verileri ve davranışları tek bir birimde kapsülleyebilir.

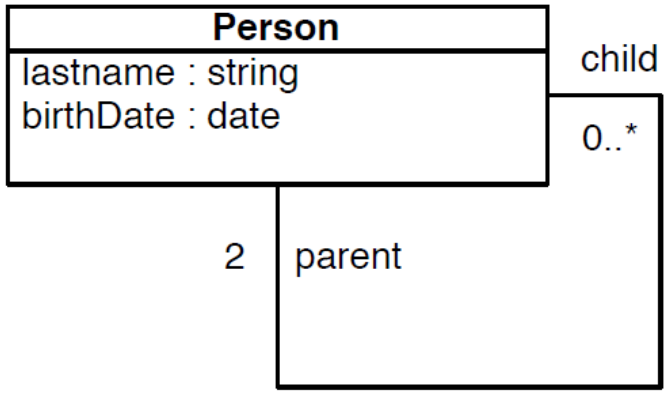
Bu da sınıfın uygulama ayrıntılarını gizlemesi ve onunla etkileşim için açık bir arayüz tanımlamasıdır.

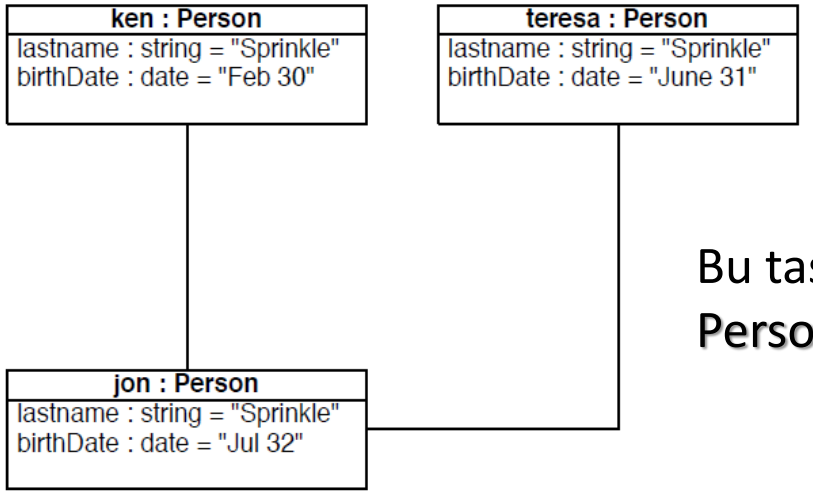
- Kapsülleme, nesnenin içsel durumuna harici kod tarafından erişilmesini veya değiştirilmesini engeller.
  - Böylece sistemin güvenliği ve güvenilirliği artar.



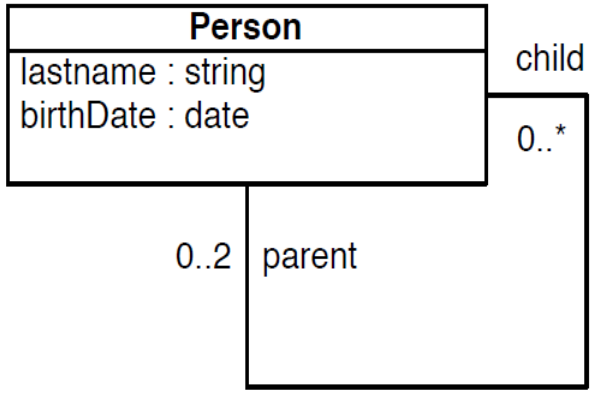
# Parent/Child kavramının Sınıf /Nesne Diyagramı ile Person

Person sınıfında çocuğun (child) ebeveyni (parent) olacaktır. Her çocuğun 2 ebeveyni vardır ve her ebeveynin 0 veya daha fazla çocuğu (child) vardır.

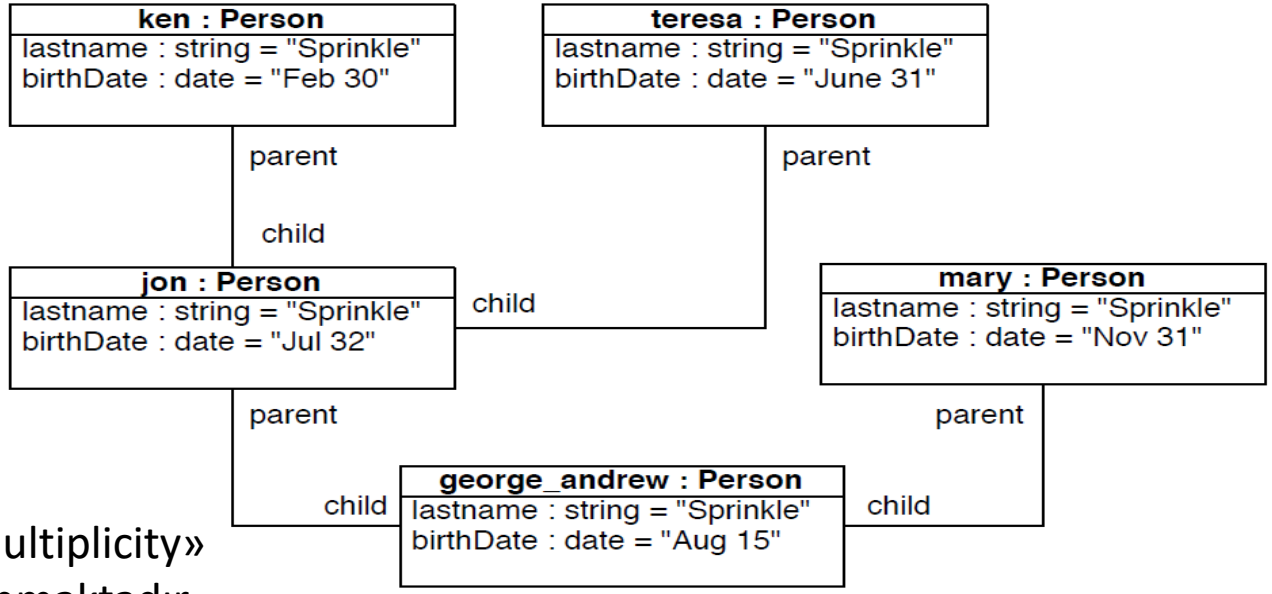



Bu tasarımda **2 ebeveyni olmayan** Person vardır.



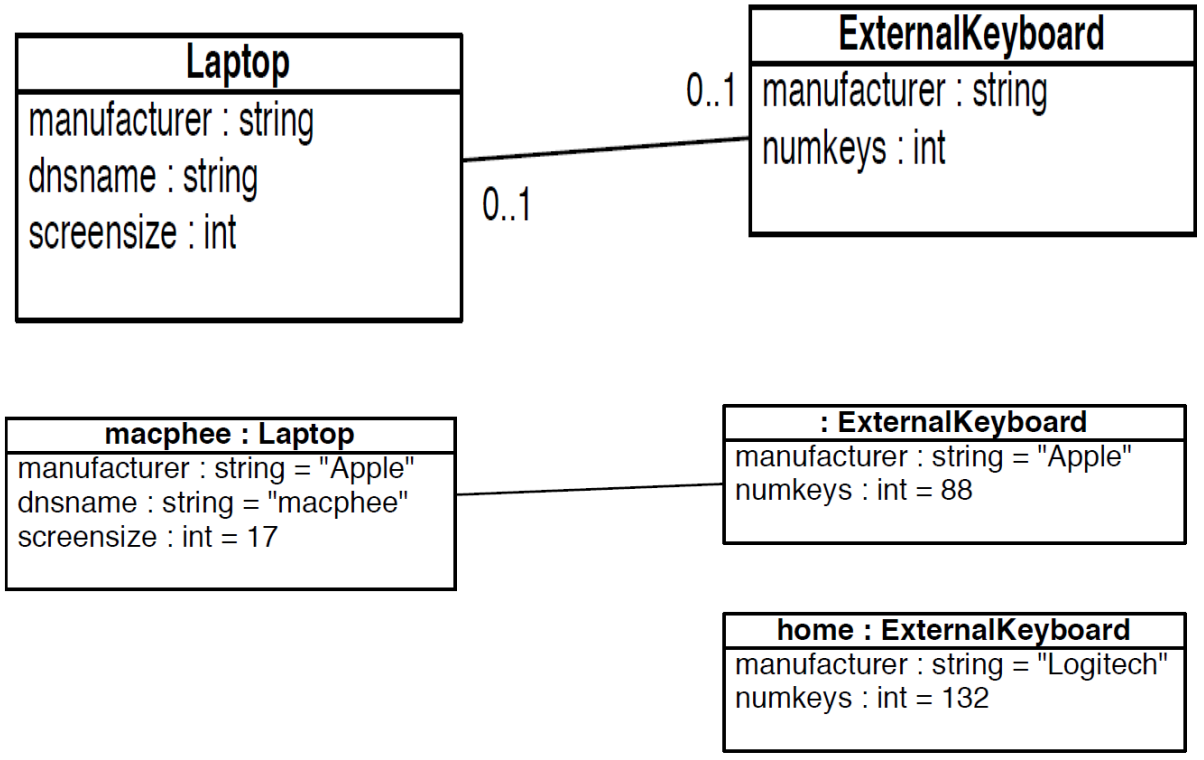

Bu tasarım ile yukarıdaki ebeveyni olmayan sınıf problemi giderilir.




Sınıf diyagramında her çocuğun (child) **0..2 sayıda ebeveyni** vardır.

İlişkilerdeki «multiplicity» önemi vurgulanmaktadır.

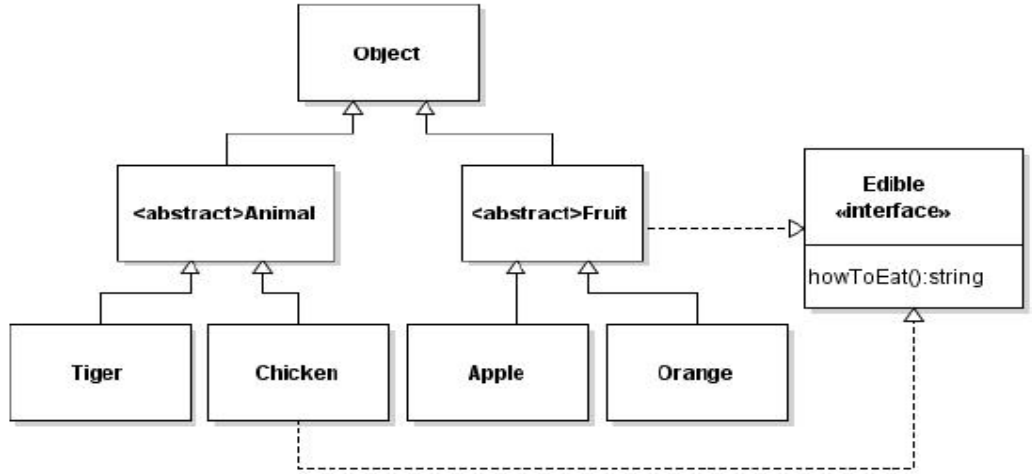
# Association /Birliktelik İlişkisi

# Abstraction /Soyutlama

- Sınıflar, karmaşık gerçek dünya kavramlarını yazılım sistemindeki daha basit, daha yönetilebilir nesnelere soyutlamanın bir yoludur.
- Bu soyutlama, gereksiz ayrıntıları göz ardı ederek bir nesnenin temel özelliklerine ve davranışlarına odaklanmayı sağlar.
- Sistemi anlamayı kolaylaştırır.

```
interface Edible {
  public abstract String howToEat(); /** Describe how to eat */
}
abstract class Animal { }
class Chicken extends Animal implements Edible {
  public String howToEat() {
    return "Chicken: Fry it";
  }
}
class Tiger extends Animal {
}/** Does not extend Edible */
abstract class Fruit implements Edible { }
class Apple extends Fruit {
  public String howToEat() {
    return "Apple: Make apple cider";
  }
}
class Orange extends Fruit {
  public String howToEat() {
    return "Orange: Make orange juice";
  }
}
public class TestEdible {
  public static void main(String[] args) {
    Object[] objects = {new Tiger(), new Chicken(), new Apple()};
    for (Object o:objects)
      if (o instanceof Edible)
        System.out.println(((Edible)o).howToEat());
  }
}
```



# Inheritance / Miras ve Polimorphism

- Sınıflar, mevcut bir sınıfın özelliklerini ve davranışını miras alan yeni sınıflar oluşturmak için kalıtımı kullanmamıza olanak tanır.
  - Miras, kodun yeniden kullanılmasına ve birden çok sınıf arasında işlevlerin kopyalanmasını önler ve sistemi daha verimli kılar ve sürdürülebilirliği kolaylaştırır.
- Sınıflar, aynı ada sahip ancak farklı parametrelere veya davranışlara sahip birden fazla yöntemi tanımlamak için polimorfizmi kullanır
  - polimorfizm, farklı girdilere ve senaryolara yanıt verebilecek daha esnek ve uyarlanabilir sistemler oluşturur.

# Overloading and Overriding

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10.0);
    a.p(10);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

```
20.0
10
```

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10.0);
    a.p(10);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

```
10.0
10.0
```

Pearson Education
Paul Fodor
(CS Stony Brook)

Method overloading, aynı isimde pek çok metodun println() farklı imzalarla implementasyonudur.

Üst sınıftaki (superclass) metotlara «overriding» uygulanır.
**Method overriding**: Alt sınıftaki metot implementasyonu üst sınıfta (süper class) tanımlanan ile değiştirilir.

# Yazılım Mühendisliğinde Fonksiyonal Olmayan Gereksinimler (Nonfunctional Requirements / NFR)

Kaynak: https://www.altexsoft.com/blog/non-functional-requirements/

# Fonksiyonel Olmayan Gereksinimler (NFR)

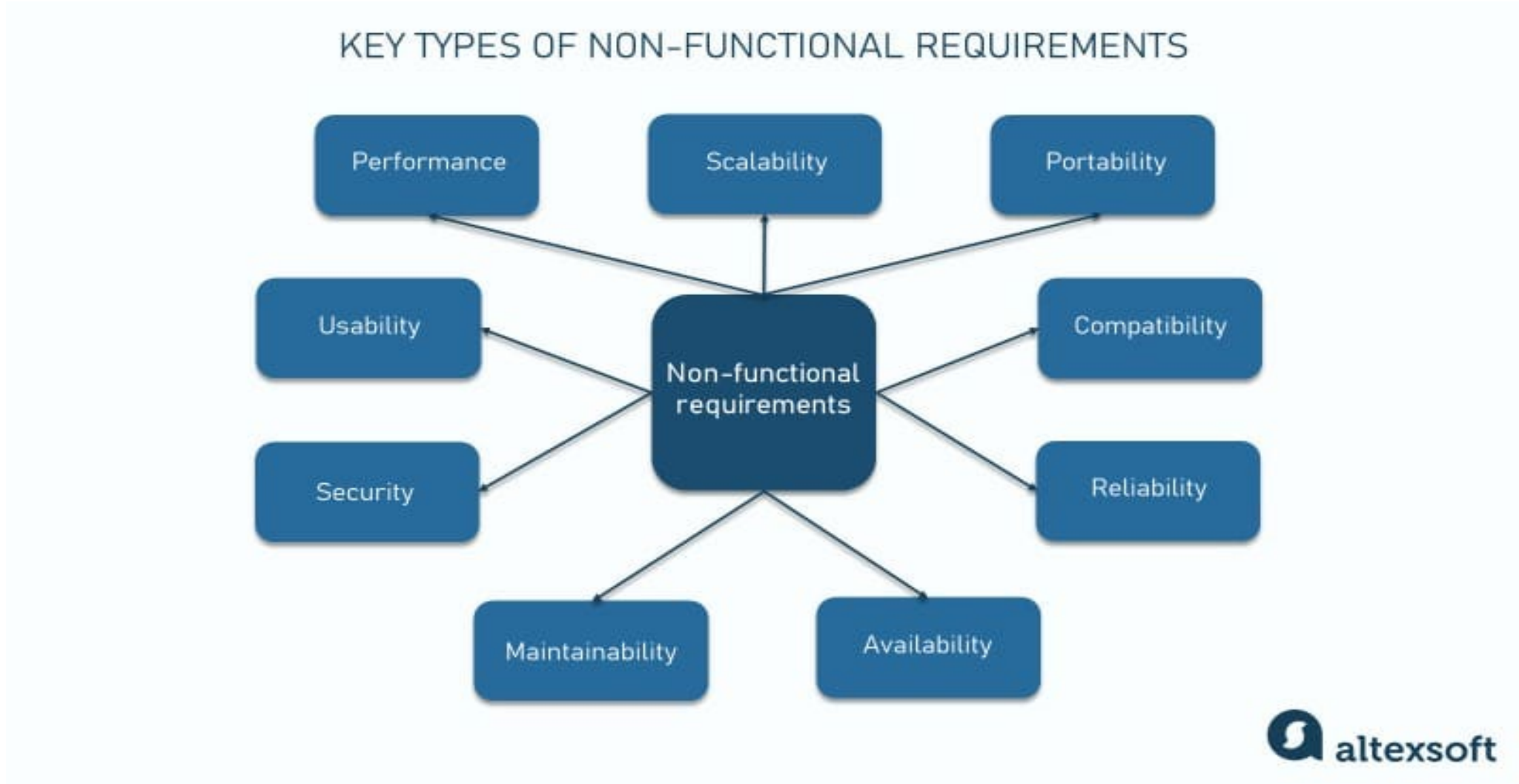
- NFR'ler, sistemin çalışma yeteneklerini ve kısıtlamalarını tanımlayan bir dizi spesifikasyondur.
  - Sistemin ne kadar iyi çalıştığını özetleyen gereksinimlerdir.
  - ✓Örneğin hız, güvenlik, güvenilirlik, veri bütünlüğü vb.
- NFR'ler, ürünün nasıl çalıştığının farklı yönlerini tanımladıkları için yazılım kalite gereksinimleri olarak anılır.

# Fonksiyonel Gereksinimlerden Farkları

- Hem fonksiyonel hem de fonksiyonel olmayan gereksinimler, ürünün paydaşların (stakeholders) ve şirketin ihtiyaçlarını (businees needs) karşılaması için sahip olması gereken belirli özellikleri tanımlar.
- Fonksiyonel gereksinimler ürünü **ne** yapması gerektiğini tanımlar
  - Bir mesajlaşma aracının fonksiyonel gereksinimi: "Kullanıcı, mesajları gönderildikten sonra düzenleyebilmelidir."
- Fonksiyonel olmayan gereksinimler sistemin **nasıl** performans göstermesi gerektiğini belirtir.
  - Mesajlaşma platformu örneğinde, bir sistemin kullanıcı beklentilerini karşılamak için düzenleme yapması gereken hız olabilir.
    - ✓ "Tüm kullanıcıların çevrimiçi olması, LTE bağlantısı veya daha iyi bir bağlantıya sahip olmasıyla, sohbetteki tüm kullanıcılar için mesajın 0,1 saniye içinde güncellenmesi gerekiyor.

# Altersoft

| | Fonksiyonel | Fonksiyonel Olmayan |
|---|---|---|
| Objective | Ürünün ne yaptığını betimler | Ürünün nasıl çalıştığını betimler |
| End Result | Ürün ile ilgili olarak business objectives (diğer bir ifade ile needs) doğrultusunda kullanıcının (user) sağlaması gereken ilgili gereksinimleri (features) tanımlar | Geliştirilecek ürünün metriklerini (product metrics) tanımlar. |
| Focus | Kullanıcı gereksinimlerine odaklanıe | Kullanıcı beklentilerine odaklanır |
| Essentially | Zorunludur | Zorunlu olmamasına rağmen arzu edilendir. |
| Origin Type | Genellikle kullanıcı tarafından tanımlanır | Genellikle geliştiriciler ve diğer teknik ekip tarafından tanımlanır |
| Testing | Fonksiyonel olmayan testlerden önce gerçekleşir. | Fonksiyonel testlerden sonra gerçekleşir. |
| Types | kimlik doğrulama, yetkilendirme seviyeleri, veri işleme, raporlama vs. | Kullanılabilirlik, Ölçeklenebilirlik, performans, güvenilirlik vs. |


KEY TYPES OF NON-FUNCTIONAL REQUIREMENTS
Performance
Scalability
Portability
Usability
Non-functional requirements
Compatibility
Security
Reliability
Maintainability
Availability
altexsoft

# Örnek NFC Tanımlamaları

- **Performance /Verim** Sistem sonuçları ne kadar hızlı döndürüyor?
- **Scalibility /Ölçeklenebilirlik** Daha yüksek iş yükleriyle performans ne kadar değişecek?
- **Portability/ Taşınabilirlik** Yazılım hangi donanım, işletim sistemi ve tarayıcılarda ve bunların versiyonlarında çalışıyor?
- **Compatibility /Uyumluluk** Sistem diğer uygulama ve süreçlerle çakışıyor mu?
- **Reliability/Güvenilirlik** Sistemde ne sıklıkta kritik arızalar yaşanıyor?
- **Maintainability /Sürdürülebilirlik** Sorun ortaya çıktığında düzeltmek ne kadar zaman alır?
- **Availability /Erişilebilme** Ortalama sistem kesinti süresi ne kadardır?
- **Security /Güvenlik** Sistem ve verileri saldırılara karşı ne kadar iyi korunuyor?
- **Usability /Kullanılabilirlik** Sistemi kullanmak ne kadar kolay?

Microsoft Visual Studio 2022 SYSTEM REQUIREMENTS

Visual Studio 2022 is supported on the following 64-bit operating systems:

- Windows 11 version 21H2 or higher: Home, Pro, Pro Education, Pro for Workstations, Enterprise, and Education
- Windows 10 version 1909 or higher: Home, Professional, Education, and Enterprise.
- Windows Server 2022: Standard and Datacenter.
- Windows Server 2019: Standard and Datacenter.
- Windows Server 2016: Standard and Datacenter.

Hardware

- 1.8 GHz or faster 64-bit processor; Quad-core or better recommended. ARM processors are not supported.
- Minimum of 4 GB of RAM. Many factors impact resources used; we recommend 16 GB RAM for typical professional solutions.
- Windows 365: Minimum 2 vCPU and 8 GB RAM. 4 vCPU and 16 GB of RAM recommended.
- Hard disk space: Minimum of 850 MB up to 210 GB of available space, depending on features installed; typical installations require 20-50 GB of free space. We recommend installing Windows and Visual Studio on a solid-state drive (SSD) to increase performance.
- Video card that supports a minimum display resolution of WXGA (1366 by 768); Visual Studio will work best at a resolution of 1920 by 1080 or higher.
    - Minimum resolution assumes zoom, DPI settings, and text scaling are set at 100%. If not set to 100%, minimum resolution should be scaled accordingly. For example, if you set the Windows display 'Scale and layout' setting on your Surface Book, which has a 3000×2000 physical display, to 200%, then Visual Studio would see a logical screen resolution of 1500×1000, meeting the minimum 1366×768 requirement.

# Visual Studio IDE için Taşınabilirlik Gereksinimleri

Kaynak: Digital Licenses

# Compatibility /uyumluluk Örnekleri

- Uyumluluk, bir sistemin aynı ortamda başka bir sistemle nasıl bir arada olabildiğine ve etkileşime girebileceğini tanımlar.
  - Yazılım ürününün diğer programlarla ne kadar uyumlu olduğunu belirtilir.
    - ✓ İşletim sistemine yüklenen yazılımın, güvenlik duvarı veya antivirüs korumasıyla uyumlu olması gerekir.
- Sağlık veri yönetimi yazılımı, benzeri farklı elektronik sağlık sistemlerindeki hasta verileri ile güvenli ve etkin bir şekilde iletişimde olabilmeli ve değişiklikler yapabilmelidir.
- Video konferans sistemi geniş aralıkta, farklı markaların webcam, microphone, speakeraygıtları ile uyumlu olmalıdır.
- ERP sistemi, sorunsuz veri transferi ve işlemlerdeki süreklilik için şirketin diğer legacy software ile uyumlu olmalıdır
- Banka yazılımı, verinin alınması ve aktarımı içim ortak kullanılan finansal veri formatları ile (CSV, XLSX, QIF gibi) uyumlu olmalıdır.

Bunun için iletişim kurulacak hedefte olan diğer sağlık sistemleri listelenmelidir; 3.parti yazılımlarla ve yazılım ekosistemindeki diğer uygulamalarla uyumluluk tanımlanmalıdır. Aynı zamanda beklenilen performans düzeyleri de belirlenmelidir.

# Reliability /Güvenilirlik  Gereksinim Örnekleri

- Sistemin veya bileşenin tanımlanlı koşullarda belirli bir süre boyunca hatasız (without failure)  çalışma olasılığının ne kadar olduğunu belirtir.
  - Bu olasılık yüzdelerle ifade edilir.
- Sistem 1 ay süresince <use case< lerin %95 inde failure olmaksızın çalışmalıdır.
- Tüm  finansal iletimler %100 accuracy ile işlenmelidir. Ve sistem veri bütünlüğünü (data integrity) her zaman gerçekleştirmelidir.
- Sistem hatalardan (errors) arınmayı  ve düzeltmeleri   veri kaybı ve yanlış veri işleme olmaksızın gerçekleştirebilmelidir.

Bu gereksinim kriterini belirlemek üzere kritik arıza/failure  (bileşen veya sistemin, beklenen teslim, servis  veya sonuçtan sapması) ve normal kullanım koşullarını tanımlamak zor olabilir.

Bu ölçüme daha basit bir yaklaşım, belirli bir süre boyunca üretimde bulunan kritik hataların sayısını saymaktır.

Ayrıca kritik arızalar arasındaki süreyi takip edebilir veya arızaya kadar geçen ortalama süre hesaplanabilir.

# Maintainability /Sürdürülebilirlik

- Bir çözümün veya bileşeninin sabitlenmesi, performansı veya diğer nitelikleri artırmak amacıyla değiştirilmesi veya değişen ortama uyarlanması için gereken süreyi tanımlar.
- Bir sistem arızasının ardından sistemi geri yüklemek için gereken ortalama süre (MTTR/Mean Time To Repair) 10 dakikadan fazla olmamalıdır.
- Video yayın hizmeti, ara belleğe almayı önlemek için video kalitesini kullanıcının internet hızına göre otomatik olarak ayarlamalıdır.

Bunun için modülerliğin tanımı ve kullanım ömrünün düşünülmesi gerekir.

# Avaliability / Erişilebilme

- Belirli bir zamanda sistemin bir kullanıcı tarafından ne kadar erişilebilir olduğunu açıklar.
- Başarılı isteklerin beklenen yüzdesi olarak ifade edilebilir.
  - Belirli bir süre boyunca sistemin işletim için erişilebilir olduğu sürenin yüzdesi olarak da tanımlanabilir.
    - ✓ Örneğin sistem bir ay boyunca yüzde 98 oranında kullanılabilir durumda olabilir.

Tamamen hatasız uygulama yoktur. Geçilemeyecek eşik tanımlanmalıdır.

- Sistemin tamamına erişilebileceği hedeflenebilir, ancak farklı ortamlar (ödeme iş akışı, açılış sayfaları, gösterge tabloları) varsa, her birinin kendi makul arıza limiti (failure) ve kullanılabilirlik gereksinimi olabilir.
- Farklı yük senaryoları açıklanabilir. Sistem, farklı iş yüklerine bağlı olarak farklı kesintilerle karşılaşabilir.
- Performans ölçümlerine benzer şekilde, normal ve olası anormal durumları tanımlamak üzere farklı durumlar dikkate alınabilir.