BMB214 Programlama Dilleri Prensipleri

Ders 12. Nesneye Yönelik Programlama Desteği

Erdinç Uzun
Tekirdağ Namık Kemal Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Giriş
- Nesneye Yönelik Programlama (Object Oriented Programming - OOP)
- Nesneye Yönelik Programlama için Tasarım Sorunları
- Smalltalk'ta Nesneye Yönelik Programlama Desteği
- C++'da Nesneye Yönelik Programlama Desteği
- Java'da Nesneye Yönelik Programlama Desteği
- C # 'da Nesneye Yönelik Programlama Desteği
- Ruby'de Nesneye Yönelik Programlama Desteği
- Nesneye Yönelik Yapıların Uygulanması
- Yansıma (Reflection)

Giriş

- Birçok nesne yönelimli programlama (OOP) dili
  - Bazıları prosedürel ve veri odaklı programlamayı destekler (örneğin, C++)
  - Bazıları fonksiyonel programı destekler (örneğin, CLOS)
  - Yeni diller diğer paradigmaları desteklemez ancak emir esaslı yapılarını kullanır (örneğin, Java ve C #)
  - Bazıları saf (pure) OOP dilidir (örneğin, Smalltalk & Ruby)
  - Bazı fonksiyonel diller OOP'yi destekler (Bu derste örnek verilmeyecektir.)

Nesneye Yönelik Programlama

- Üç ana dil özelliği:
  - Soyut veri türleri – Abstract data types (Geçen hafta işlendi…)
  - Kalıtım (Inheritance)
    - Kalıtım, OOP ve onu destekleyen dillerin ana temasıdır
  - Polimorfizm – Çok şekillilik (Polymorphism)

Kalıtım (Inheritance)

- Verimlilik artışları yeniden kullanımdan (reuse) kaynaklanabilir
  - Soyut veri türlerinin yeniden kullanılması zordur - her zaman değişiklik gerekir
  - Tüm soyut veri türleri bağımsızdır ve aynı seviyededir
- Kalıtım, mevcut sınıflara göre tanımlanan yeni sınıflara, yani ortak parçaları miras almalarına izin vererek izin verir.
- Kalıtım, yukarıdaki endişelerin her ikisine de adresler - küçük değişikliklerden sonra soyut veri türleri yeniden kullanılabilir ve bir hiyerarşide sınıfları tanımlanabilir

Nesneye Yönelik Kavramlar

- Soyut veri türleri genellikle sınıflar (classes) olarak adlandırılır
- Sınıf örneklerine nesneler (objects) denir
- Kalıtım alan bir sınıf, türetilmiş bir sınıf veya bir alt sınıftır
- Başka bir sınıfın miras aldığı sınıf, bir üst sınıf (parent class) veya superclass'tır.
- Nesneler üzerindeki işlemleri tanımlayan alt programlara metot denir

Nesneye Yönelik Kavramlar...

- Metot çağrılarına mesaj (message) denir
- Bir nesnenin tüm metot koleksiyonuna mesaj protokolü (message protocol) veya mesaj arayüzü (message interface) denir.
- Mesajların iki bölümü vardır - bir metot adı ve hedef nesne (destination object)
- En basit durumda, bir sınıf, üst sınıfının tüm varlıklarını miras alır.

Nesneye Yönelik Kavramlar...

- Kalıtım, kapsüllenmiş varlıklara erişim kontrolleri ile karmaşık hale getirilebilir
  - Bir sınıf, varlıkları alt sınıflarından gizleyebilir
  - Bir sınıf, varlıkları istemcilerden (object) gizleyebilir
  - Bir sınıf, alt sınıflarının görmesine izin verirken istemcileri için varlıkları da gizleyebilir.
- Metotları olduğu gibi miras almanın yanı sıra, bir sınıf miras alınan bir metodu değiştirebilir
  - Yenisi, miras kalanını geçersiz kılar (override)
  - Ebeveyndeki (üst, parent) metot geçersiz kılınır

Nesneye Yönelik Kavramlar...

- Bir sınıfın ebeveyn sınıfından farklı olabileceği üç yol:
  - Alt sınıf, ebeveynden miras alınanlara değişkenler ve / veya metotlar ekleyebilir
  - Alt sınıf, miras alınan metotlarından bir veya daha fazlasının davranışını değiştirebilir
  - Ebeveyn sınıf, değişkenlerinden veya metotlarından bazılarını private erişime sahip olacak şekilde tanımlayabilir, bu, alt sınıfta görünmeyecekleri anlamına gelir

Nesneye Yönelik Kavramlar...

- Bir sınıfta iki tür değişken vardır:
  - Sınıf (Class) değişkenleri - sınıf - class
  - Örnek (Instance) değişkenler - nesne - object
- Bir sınıfta iki tür metot vardır:
  - Sınıf metotları - sınıfa gönderilen mesajları kabul eder
  - Örnek metotları - nesnelere gönderilen mesajları kabul eder
- Tek vs. Çoklu Kalıtım (Single vs. Multiple Inheritance)
- Yeniden kullanım için mirasın bir dezavantajı:
  - Bakımı zorlaştıran sınıflar arasında karşılıklı bağımlılıklar yaratır

Dinamik Bağlama (Dynamic Binding)

- Bir polimorfik (polymorphic) değişken, sınıfın nesnelerine ve nesillerinden herhangi birinin nesnelerine başvurabilen (veya bunları işaret edebilen) bir sınıfta tanımlanabilir.
- Bir sınıf hiyerarşisi, metotları geçersiz kılan sınıfları içerdiğinde ve bu tür yöntemler bir polimorfik değişken aracılığıyla çağrıldığında, doğru metoda bağlanma dinamik olacaktır.
- Hem geliştirme hem de bakım sırasında yazılım sistemlerinin daha kolay genişletilmesine izin verir

Dinamik Bağlama Kavramı

- Soyut bir metot (abstract method), bir tanım içermeyen bir yöntemdir (yalnızca bir protokolü tanımlar)
- Soyut bir sınıf (abstract class), en az bir sanal metot (virtual method) içeren bir sınıftır
- Soyut bir sınıf nesne olamaz

Nesneye Yöenlik Programlama Tasarım Sorunları

- Nesnelerin Ayrıcalığı
- Alt Sınıfların Alt Türleri mi?
- Tekli ve Çoklu Kalıtım (Single and Multiple Inheritance)
- Nesne Tahsisi ve Tahsisin Kaldırılması (Object Allocation and Deallocation)
- Dinamik ve Statik Bağlama
- İç içe Sınıflar (Nested Classes)
- Nesnelerin başlatılması (Initialization of Objects)

Nesnelerin Ayrıcalığı

- Her şey bir nesnedir (Everything is an object)
  - Avantaj - zarafet ve saflık
  - Dezavantaj - basit nesneler üzerinde yavaş işlemler
- Eksiksiz bir tür sistemine nesneler ekler
  - Avantaj - basit nesneler üzerinde hızlı işlemler
  - Dezavantaj - kafa karıştırıcı bir tip sistemle sonuçlanır (iki tür varlık)
- İlkel öğeler için emir esaslı bir tür sistemi ekler, ancak diğer her şeyi nesneler yapar
  - Avantaj - basit nesneler üzerinde hızlı işlemler ve nispeten küçük bir tür sistemi
  - Dezavantaj - iki tür sistem nedeniyle hala bazı karışıklıklar

Alt Sınıfların Alt Türleri mi?

- Bir ebeveyn sınıf nesnesi ile alt sınıfın bir nesnesi arasında bir "is-a" ilişkisi var mı?
  - Türetilmiş bir sınıf bir ebeveyn sınıfsa, türetilmiş sınıfın nesneleri, üst sınıf nesnesiyle aynı şekilde davranmalıdır.
- Türetilmiş bir sınıf, üst sınıfıyla bir is-a ilişkisine sahipse bir alt türdür
  - Alt sınıf yalnızca değişkenler ve metotlar ekleyebilir ve devralınan yöntemleri "uyumlu" yollarla geçersiz kılabilir
- Alt sınıflar uygulamayı devralır; alt türler arabirimi ve davranışı devralır

Tekli ve Çoklu Kalıtım (Single and Multiple Inheritance)

- Çoklu kalıtım, yeni bir sınıfın iki veya daha fazla sınıftan miras almasına izin verir
- Çoklu mirasın dezavantajları:
  - Dil ve uygulama karmaşıklığı (kısmen ad çakışmalarından dolayı)
  - Potansiyel verimsizlik - dinamik bağlama, çoklu kalıtıma göre daha fazla maliyetlidir (ancak çok değil)
- Avantaj:
  - Bazen oldukça kullanışlı ve değerlidir

Nesne Tahsisi ve Tahsisin Kaldırılması (Object Allocation and Deallocation)

- Nesneler nereden tahsis edilir?
  - Soyut veri türleri davranırlarsa, herhangi bir yerden tahsis edilebilirler.
    - Çalışma zamanı (run-time) stack'inden tahsis edilir
    - Stack üzerinde açıkça (explicitly) oluştur (new)
  - Hepsi heap-dynamic, referanslar bir işaretçi veya referans değişkeni aracılığıyla tek tür olabilir
    - Atamayı basitleştirir - başvurunun kaldırılması örtük (implicit) olabilir
  - Nesneler stack dynamic ise, alt türlerle ilgili bir sorun vardır - (object slicing)
- Deallocation açık mı yoksa örtülü mü? (explicit or implicit)

Dinamik ve Statik Bağlama (Dynamic and Static Binding)

- Mesajların metotlara tüm bağlanması dinamik olmalı mı?
  - Hiçbiri değilse, dinamik bağlamanın avantajlarını kaybedersiniz
  - Hepsi öyleyse, verimsizdir
- Belki tasarım kullanıcının şunu belirtmesine izin vermelidir: statik veya dinamik

İç içe Sınıflar (Nested Classes)

- Yalnızca bir sınıf tarafından yeni bir sınıfa ihtiyaç duyulursa, diğer sınıflar tarafından görülebilmesi için tanımlanması için bir neden yoktur.
  - Yeni sınıf, onu kullanan sınıfın içine yerleştirilebilir mi?
  - Bazı durumlarda, yeni sınıf doğrudan başka bir sınıf yerine bir alt programın içine yerleştirilir.
- Diğer sorunlar:
- İç içe sınıfının hangi özellikleri iç içe sınıf tarafından görülebilir olmalıdır ve bunun tersi de geçerlidir

Nesnelerin başlatılması (Initialization of Objects)

- Nesneler oluşturulduklarında değerlere göre başlatılıyor mu?
  - Örtülü veya açık başlatma (Implicit or explicit initialization)
- Bir alt sınıf nesnesi oluşturulduğunda ebeveyn sınıf üyeleri nasıl başlatılır?

OOP
Smalltalk: Saf bir OOP Dili

- Her şey bir nesnedir (Everything is an object)
- Tüm nesnelerin yerel belleği vardır
- Tüm hesaplama, nesnelere mesaj gönderen nesneler aracılığıyladır.
- Emir esaslı dillerin görünüşlerinden hiçbiri
- Tüm nesneler heap'ten tahsis edilir
- Tüm tahsis geri alma örtülüdür (implicit)
- Smalltalk sınıfları iç içe olamaz

OOP
Smalltalk: Inheritance (Kalıtım)

- Smalltalk alt sınıfı, ebeveyn sınıfının (superclass, parent class) tüm örnek değişkenlerini, örnek metotlarını ve sınıf metotlarını miras alır.
- Tüm alt sınıflar alt türlerdir (hiçbir şey gizlenemez)
- Tüm miras, uygulama mirasıdır
- Çoklu miras yok

OOP
Smalltalk: Dinamik Bağlama

- Metotlara yapılan tüm mesaj bağlanmalarını dinamiktir
  - İşlem, metot için mesajın gönderildiği nesneyi araştırmaktır; Bulunmazsa, üst sınıfı olmayan sistem sınıfına kadar ebeveyn sınıfı vb. arayın
- Smalltalk'daki sadece tür denetimi dinamiktir ve tür hatası, eşleşen metodu olmayan bir nesneye bir mesaj gönderildiğinde ortaya çıkar.

OOP
Smalltalk: Değerlendirme

- Dilin sözdizimi basit ve düzenlidir
- Minimal bir dilin sağladığı gücün güzel bir örneği
- Geleneksel derlenmiş emir esaslı dillere kıyasla yavaş
- Dinamik bağlama, yazım hatalarının çalışma zamanına kadar algılanmamasına sebebiyet verir
- Grafik kullanıcı arayüzünü (GUI – Graphical User Interface) tanıttı
- En büyük etki: OOP'nin ilerlemesi

OOP
C++

- C ve SIMULA 67'den geliştirildi
- En yaygın kullanılan OOP dilleri arasında
- Karışık tür sistemi
- Yapıcılar ve yıkıcılar (Constructors - destructors)
- Sınıf varlıklarına ayrıntılı erişim kontrolleri

OOP
C++: Kalıtım

- Bir sınıfın herhangi bir sınıfın alt sınıfı olması gerekmez
- Üyeler için erişim kontrolleri (erişim beliteçleri)
  - Private (yalnızca sınıfta ve arkadaşlarda görünür) (alt sınıfların alt tür olmasına izin vermez)
  - Public (alt sınıflarda ve nesnelerde görünür)
  - Protected (sınıfta ve alt sınıflarda görünür, ancak nesnelerde görünmez)

OOP
C++: Kalıtım Örneği

```
class base_class {
  private:
    int a;
    float x;
  protected:
    int b;
    float y;
  public:
    int c;
    float z;
};

class subclass_1 : public base_class { ... };
//      Burada, b ve y protected
//      c ve z public

class subclass_2 : private base_class { ... };
//     Burada, b, y, c ve z private,
//     türetilmiş bir sınıfın herhangi bir base_class üyesine
   erişimi yoktur
```

OOP
C++ : Yeniden Dışa Aktarma (Reexportation)

- Bir alt sınıfta erişilemeyen bir üye (private türetme nedeniyle), kapsam çözümleme operatörü (: :) kullanılarak orada görünür olarak bildirilebilir

```
class subclass_3 : private base_class {
        base_class :: c;
         ...
}
```

OOP
C++ : Yeniden Dışa Aktarma (Reexportation)...

- Private türetmeyi kullanmak için bir motivasyon
  - Bir sınıf, görünür olması gereken üyeler sağlar, böylece bunlar public üye olarak tanımlanır; türetilmiş bir sınıf bazı yeni üyeler ekler, ancak nesnelerin ana sınıf tanımında herkese açık olmaları gerekse bile ebeveyn sınıfın üyelerini görmesini istemez

OOP
C++: Çoklu Kalıtım (Multiple Inheritance)

- Çoklu miras desteklenir.
  - Aynı ada sahip iki devralınan üye varsa, her ikisine de kapsam çözümleme operatörü (: :) kullanılarak başvurulabilir.

```
class Thread { ... }
class Drawing { ... }
class DrawThread : public Thread, public
   Drawing { … }
```

OOP
C++: Dinamik Bağlama

- Bir metot virtual olarak tanımlanabilir; bu, polimorfik değişkenler aracılığıyla çağrılabilecekleri ve mesajlara dinamik olarak bağlanabilecekleri anlamına gelir
- Saf bir virtual fonksiyon hiçbir tanımı yoktur
- En az bir saf virtual fonksiyon olan bir sınıf, soyut (abstract) bir sınıftır

```
class Shape {
  public:
    virtual void draw() = 0;
    ...
};
class Circle : public Shape {
  public:
    void draw() { ... }
  ...
};
class Rectangle : public Shape {
  public:
    void draw() { ... }
  ...
};
class Square : public Rectangle {
  public:
    void draw() { ... }
  ...
};
```

```
Square* sq = new Square;
Rectangle* rect = new Rectangle;
Shape* ptr_shape;
ptr_shape = sq;  // Square işaret eder
ptr_shape ->draw();  // dinamik olarak
                     // Square'a draw'ı bağlar
rect->draw();  // Statik olarak
               // Square'a draw'ı bağlar
```

- Nesneler stack'ten tahsisi, oldukça farklıdır
  - `Square sq;` // Stack'ten bir Square nesnesi ayırır
  - `Rectangle rect;` // Stack'ten bir rectangle nesnesi ayırır
  - `rect = sq;` // sq nesnesinden veri üye değerlerini kopyalar
  - `rect.draw();` // Rectangle'dan draw'ı çağırır

OOP
C++: Değerlendirme

- C++, kapsamlı erişim kontrolleri sağlar (Smalltalk'ın aksine)
- C++ çoklu miras sağlar
- C++'da, programcı tasarım sırasında hangi yöntemlerin statik olarak bağlanacağına ve hangilerinin dinamik olarak bağlanacağına karar vermelidir.
  - Statik bağlama daha hızlıdır!
- Smalltalk tür denetimi dinamiktir (esnektir, ancak güvensizdir)
- Yorumlama ve dinamik bağlama nedeniyle Smalltalk, C++'dan ~10 kat daha yavaştır

- C++ ile yakın ilişkisi nedeniyle, bu dilden farklılıklara odaklanacağız.
- Genel özellikleri
  - İlkel türler dışındaki tüm veriler nesnelerdir
  - Tüm ilkel türlerin, bir veri değerini depolayan sarmalayıcı sınıfları vardır
  - Tüm nesneler heap-dynamic, referans değişkenleri aracılığıyla referans alınır ve new komutu ile yer tahsisi yapılır
- Çöp toplayıcı (Garbage Collector - GC) nesnenin kapladığı depolamayı (memory) geri vermek üzereyken bir finalize metodu örtülü (implicit) olarak çağrılır

OOP
Java – Kalıtım (Inheritance)

- Yalnızca tek kalıtım desteklenir, ancak çoklu kalıtımın (interface) bazı faydalarını sağlayan soyut bir sınıf kategorisi vardır.
- Bir interface yalnızca metot bildirimlerini ve adlandırılmış sabitleri içerebilir, ör.

```
public interface Comparable <T> {
                    public int comparedTo (T b);
    }
```

- Metotlar final olabilir (override edilemez)
- Tüm alt sınıflar alt türlerdir

OOP
Java – Dinamik Bağlama (Dynamic Binding)

- Java'da, metot final olmadığı sürece tüm mesajlar dinamik olarak metotlar bağlıdır (örneğim, override edilemez, bu nedenle dinamik bağlama bir amaca hizmet etmez)
- Her ikisi de override'a izin vermeyen metotlar static veya private olması durumunda statik bağlama da kullanılır.

OOP
Java – Nested Classes (İç içe geçmiş Sınıflar)

- İç içe geçmiş sınıf hariç tümü, paketlerindeki (packages) tüm sınıflardan gizlenir
- Doğrudan iç içe geçmiş statik olmayan sınıflara innerclasses denir
  - Bir iç sınıf, iç içe geçmiş sınıfının üyelerine erişebilir
  - Statik iç içe geçmiş bir sınıf, yuvalama sınıfının üyelerine erişemez
- İç içe geçmiş sınıflar anonim (anonymous) olabilir
- Yerel (Local) iç içe geçmiş bir sınıf, iç içe sınıfın sınıfının bir metodunda tanımlanır
  - Erişim belirticisi kullanılmaz

OOP
Java – Değerlendirme

- OOP'yi desteklemek için tasarım yöntemleri C++'ya benzer
- Prosedürel programlama için destek yok
- Ebeveynsiz sınıf yok
- Dinamik bağlama, metot çağrılarını metot tanımlarına bağlamanın "normal" yolu olarak kullanılır
- Çoklu miras için basit bir destek biçimi sağlamak için interface'ler kullanır

- Genel özellikleri
  - Java'ya benzer OOP desteği
  - Hem class hem de struct içerir
  - Sınıflar Java'nın sınıflarına benzer
  - Struct daha az güçlü stack-dynamic yapılardır (örneğin, kalıtım yok)

OOP
C# - Kalıtım (Inheritance)

- Sınıfları tanımlamak için C++ sözdizimini kullanır
- Ebeveyn sınıftan miras alınan bir metot, türetilmiş sınıfta tanımını new ile işaretleyerek değiştirilebilir.
- Üst sınıf sürümü, önek tabanı ile açıkça çağrılabilir:

```
base.Draw()
```

- Ebeveyn sınıfın hiçbir üyesi (member) private değilse alt sınıflar alt türlerdir
- Yalnızca tek kalıtım

OOP
C# - Dinamik Bağlama (Dynamic Binding)

- Metot çağrılarının metotlara dinamik bağlanmasına izin vermek için:
  - Temel sınıf metodu virtual olarak işaretlenmiştir
  - Türetilmiş sınıflarda karşılık gelen metotlar override olarak işaretlenmiştir
- Soyut metodlar abstract olarak işaretlenmiştir ve tüm alt sınıflarda uygulanmalıdır
- Tüm C# sınıfları nihayetinde tek bir kök sınıftan oluşur: Object

OOP
C# - İç İçe Geçmiş Sınıflar (Nested Classes)

- Doğrudan bir iç içe geçmiş bir nested C# sınıfı, Java statik iç içe sınıfı gibi davranır
- C#, Java'nın statik olmayan sınıfları gibi davranan iç içe geçmiş sınıfları desteklemez

OOP
C# - Değerlendirme

- C#, nispeten yakın zamanda tasarlanmış C tabanlı bir OO dilidir
- C#'ler ile Java'nın OOP desteği arasındaki farklar nispeten küçüktür

OOP
Ruby

- Her şey bir nesnedir (object)
- Tüm hesaplama mesaj geçişi (message passing) yoluyla yapılır
- Sınıf tanımları yürütülebilir, ikincil tanımların mevcut tanımlara üye eklemesine izin verir
- Metot tanımları da çalıştırılabilir
- Tüm değişkenler nesnelere türsüz başvurulardır
- Erişim kontrolü, veriler ve metotlar için farklıdır
  - Tüm veriler için private'tır ve değiştirilemez
  - Metotlar public, private veya protected olabilir
  - Metot erişimi çalışma zamanında kontrol edilir
- Getters ve setters kısayollarla tanımlanabilir

OOP
Ruby

- Kalıtım (Inheritance)
  - Devralınan metotlara erişim denetimi, ebeveyn sınıftakinden farklı olabilir
  - Alt sınıflar mutlaka alt türler değildir
- Dinamik Bağlama (Dynamic Binding)
  - Tüm değişkenler tipsiz ve polimorfiktir
- Değerlendirme
  - Soyut (Abstract) sınıfları desteklemez
  - Çoklu mirası tam olarak desteklemiyor
  - Erişim kontrolleri, OOP'yi destekleyen diğer dillerden daha zayıftır

OOP Diğer Diller

- Python
  - self kullanımına dikkat
  - 12. Nesne Yönelimli Programlama - Python - Erdinç Uzun (erdincuzun.com)
- Javascript
  - Prototype
  - Traditional way: class, new, ES6
  - Inheritance in JavaScript - Learn web development | MDN (mozilla.org)
  - 03. Javascript ve ES6 - Erdinç Uzun (erdincuzun.com)
- PHP
  - PHP OOP Classes and Objects (w3schools.com)

Instance Data Storage

- Sınıf örneği kayıtları (Class Instance Records - CIR'ler) bir nesnenin durumunu depolar
  - Statik (derleme zamanında (compile-time) oluşturulur)
- Bir sınıfın bir ebeveyn'i varsa, alt sınıf örnek değişkenleri (subclass instance variables) ebeveyn CIR'ye eklenir
- CIR statik olduğundan, tüm örnek değişkenlerine erişim kayıtlarda olduğu gibi yapılır
  - Verimli

Metot Çağrılarının Dinamik Bağlanması (Dynamic Binding of Methods Calls)

- Statik olarak bağlı bir sınıftaki metotların CIR'ye dahil edilmesi gerekmez; Dinamik olarak bağlanacak metotların CIR'da girişleri olmalıdır
  - Dinamik olarak bağlı yöntemlere yapılan çağrılar, CIR'deki bir işaretçi (pointer) aracılığıyla karşılık gelen koda bağlanabilir.
  - Depolama yapısı bazen sanal metot tabloları (*virtual method tables* - vtable) olarak adlandırılır
  - Metot çağrıları, vtable'ın başlangıcından itibaren ofsetler olarak temsil edilebilir

Reflection (Yansıma)

- Yansımayı (Reflection) destekleyen bir programlama dili, programlarının türlerine ve yapılarına çalışma zamanı (run-time) erişimine sahip olmasına ve davranışlarını dinamik olarak değiştirebilmesine olanak tanır.
- Bir programın türleri ve yapısı meta veriler (metadata) olarak adlandırılır
- Bir programın meta verilerini inceleyen süreci iç gözlem (introspection) olarak adlandırılır.
- Bir programın yürütülmesine müdahale etmeye intercession denir

Reflection...

- Yazılım araçları için reflection kullanımları:
  - Sınıf tarayıcılarının (class browsers) bir programın sınıflarını numaralandırması gerekir
  - Görsel IDE'ler, geliştiricinin türü doğru kod oluşturmasına yardımcı olmak için tür bilgilerini kullanır
  - Hata ayıklayıcıların (Debuggers) private alanları ve sınıf metotlarını incelemeleri gerekir
  - Test sistemleri bir sınıfın tüm metotlarını bilmelidir

Reflection
Java

- Java.lang.Class'dan sınırlı destek
- Java çalışma zamanı, programdaki her nesne için bir class örneği oluşturur
- Class'ın getClass metodu, bir nesnenin Class nesnesini döndürür

```
float[] totals = new float[100];
Class fltlist = totals.getClass();
Class stg = "hello".getClass();
```

- Nesne yoksa class alanını kullanın

```
Class stg = String.class;
```

Reflection
Java

- Sınıfın dört yararlı metodu vardır:
  - getMethod bir sınıfın belirli bir public metodunu arar
  - getMethods, bir sınıfın tüm public metotların bir dizisini döndürür
  - getDeclaredMethod bir sınıfın belirli bir metodunu arar
  - getDeclaredMethods, bir sınıfın tüm metotlarının bir dizisini döndürür
- Method sınıfı, getMethod tarafından bulunan metodu yürütmek için kullanılan invoke yöntemini tanımlar.

Reflection
C#

- .NET dillerinde, derleyici ara kodu (intermediate code) programla ilgili meta verilerle (metadata) birlikte bir derlemeye yerleştirir.
- System.Type, reflection için ad alanıdır (namespace)
- getClass yerine getType kullanılır
- .class alanı yerine typeof operatörü kullanılır
- System.Reflection.Emit namespace'i, ara kod oluşturma ve bir derlemeye içine yerleştirme sağlar (Java bu özelliği sağlamaz)

Reflection Dezavantajları

- Performans maliyetleri
- Private alanları ve metotları ortaya çıkarır
- Erken tür denetiminin avantajlarını ortadan kaldırır
- Bazı reflection kodları bir güvenlik yöneticisi altında çalışmayabilir, bu da kodu taşınamaz hale getirir

Java Annotation vs. C#

- Bir program öğesi (genellikle bir sınıf, metot veya alan) üzerindeki açıklama, o öğeyi fazladan kodla tanımlamak için kullanılabilen bu program öğesine eklenen bir meta veri (metadata) parçasıdır.
- Meta verilerinizin ne zaman erişilebilir hale getirileceğinin kontrolü, iki dil arasında farklıdır.
- Java'da buna annotions, C#'da buna attribute denir. C#, ilk sürümünden beri bunlara sahipti, Java'da 1.5 sürümüyle birlikte ortaya çıktı.
- Bir özniteliğe iyi bir örnek, hizmet dışı kalan öğeleri işaretlemek için C #'daki [Obsolete] attribute (Java'da kullanımdan deprecated aynıdır). Obsolete özniteliği, mesajları yazdırmak ve öğenin kullanımının bir hata mı yoksa uyarı mı olduğunu belirtmek için bağımsız değişkenler alabilir. Dil platformu, tanımlanmış birçok ek açıklama ile birlikte gelir, ancak kendi notlarınızı eklemenize izin verir.
- Ruby’de RubyAnnotations çok yaygındır - ancak sınıf metotları yapılır.
- Python’da decorator konusu karşımıza çıkar.