Java ile Nesne Merkezli Programlamaya Giriş

7. Bölüm
Java Nesneleri: Temeller

Akın Kaldıroğlu

www.javaturk.org

Aralık 2016

Küçük Ama Önemli Bir Konu

- Bu dosya ve beraberindeki tüm, dosya, kod, vb. eğitim malzemelerinin tüm hakları **Selsoft Yazılım, Danışmanlık, Eğitim ve Tic. Ltd. Şti**.'ne aittir.
- Bu eğitim malzemelerini kişisel bilgilenme ve gelişiminiz amacıyla kullanabilirsiniz ve isteyenleri **http://www.selsoft.academy** adresine yönlendirip, bu malzemelerin en güncel hallerini almalarını sağlayabilirsiniz.
- Yukarıda bahsedilen amaç dışında, bu eğitim malzemelerinin, ticari olsun/olmasın herhangi bir şekilde, toplu bir eğitim faaliyetinde kullanılması, bu amaca yönelik olsun/olmasın basılması, dağıtılması, gerçek ya da sanal/Internet ortamlarında yayınlanması yasaktır. Böyle bir ihtiyaç halinde lütfen benimle, **akin.kaldiroglu@selsoft.academy** adresinden iletişime geçin.
- Bu ve benzeri eğitim malzemelerine katkıda bulunmak ya da düzeltme ve eleştirilerinizi bana iletmek isterseniz çok sevinirim.
- Bol Java'lı günler dilerim.

Gündem

- Bu bölümde Java'da nesne-merkezli programlamaya giriş yapılacaktır.
- Sınıf, nesne, durum, arayüz vb. temel nesne kavramları bu bölümde ele alınacaktır.
- Nesne ve sınıf değişkenleri ile nesne ve sınıf metotları da bu bölümde incelenecektir.

Sınıflar ve Nesneleri

Sınıf

- Sınıf, kendisinden üretilecek nesnelerin kalıbıdır - şablonudur.
  - Aynı sınıftan üretilen nesnelerin tipi, aynıdır.
- Sınıf, nesnelerinin özelliklerini (attributes) ve davranışlarını (behavior) tanımlar.
  - Nesnelerin özellikleri, değişkenlerle (variables),
  - Nesnelerin davranışları ise metotlarla (methods) tanımlanır.
- Nesnenin özelliklerinin bütününe **durum** (**state**), metotların bütününe de **arayüz** (**interface**) denir.

Java'da Sınıf Tanımlama I (Tekrar)

- Java'da sınıf tanımlamak için **class** anahtar kelimesi kullanılır:

```
<niteleyici>* class <SınıfİSmi>{
    <özellik>*
    <kurucu>*
    <metot>*
}
```

- Sınıfın tanımı, Java'da en geniş blok olan sınıf blokuyla yapılır.
- Sınıfın, sıfır ya da daha fazla niteleyicisi (modifier) olabilir.
- Sınıfın, geçerli ve anlamlı bir ismi olmalıdır.

Java'da Sınıf Tanımlama II (Tekrar)

- Kurucu ya da yapılandırıcı (constructor), nesne yaratılırken çağrılan özel bir metotdur.
- Zorunlu olmamakla birlikte sınıfın öğeleri, fiziksel olarak sınıf içinde, özellikler, kurucular ve metotlar olarak sıralanır.
  - Özellikler, farklı tiplerde olan nesne değişkenleridir (instance variables).
  - Metotlar ise nesnelerin sorumluluklarını yerine getiren nesne fonksiyonlarıdır (instance methods).

Nesne Yaratmak I

- Nesne yaratmak dört adımda gerçekleşir:
  - **Tanıtım (Declaration)**: Önce yaratılacak nesneyi gösterecek referans değişkeni tanıtılır.
  - **Yaratma (Instantiation)**: İkinci adımda **new** anahtar kelimesi kullanılarak nesne yaratılır. **new** operatörü, nesne yaratıldığında, onu gösteren bir referansı geriye döndürür.
  - **Başlangıç durumuna getirme (Initialization)**: Kurucu çağrısı yapılarak, nesne, başlangıç durumuna getirilir.
  - **Atama (Assignment)**: Başlangıç durumuna getirilen nesnenin referansı, kendi tipinden bir referans değişkenine atanır.

```
Car carObject = new Car();
```

Nesne Yaratmak II

- Bu dört adım, tek bir yerde, hep beraber yapılabileceği gibi, önce tanıtım, daha sonra da yaratma, başlangıç durumuna getirme ve atama hep birlikte, başka bir yerde yapılabilir.

```
Car carObject;          // Step 1
...
carObject = new Car();   // Step 2, 3 & 4
```

- Her nesne, bellekte ayrı bir alana sahiptir.
- Her nesnenin özellikleri, kendi bellek alanında tutulur.
- Referansın tipi, göstereceği nesnenin tipidir.
  - En azından şimdilik bunu bu basitlikte düşünebiliriz.

Nesne Referansı

- Nesneler, heap adı verilen ve RAM'in JVM tarafından dinamik olarak kullanılabilen alanında oluşturulurlar:
  - Nesnelerin özellikleri olan değişkenler bu alanda yaşarlar.
- Java'da, bu bellek alanına doğrudan ulaşmamız ya da müdahele etmemiz mümkün değildir.
- Java'da, nesnelere, onların referanslarıyla (reference) ulaşılır. Referansa aşağıdaki isimler de verilir:
  - Tutaç (handle): Nesneye soyut bir erişim mekanizmasıdır.
  - İşaretçi (pointer): Nesnenin bellekteki adresini tutar.
- Java'daki referanslar, C++ pointerlarının daha soyut halidir.
  - C++'ta, pointerlarla yoluyla nesnesin fiziksel adresine ulaşılabilir v pointer aritmetiği yapılabilir.

Nesnenin Durumu

Değişkenlerin Rolleri (Tekrar)

- Java'da değişkenler, basit olsun referans olsun, fonksiyonellik ya da rol açısından üçe ayrılırlar:
  - **Nesne değişkenleri** (**instance** (**object**) **variables**): Nesnenin durumunu oluşturan değişkenlerdir.
  - **Sınıf değişkenleri** (**class variables**): Nesnelerin ortak durumunu ifade eden değişkenlerdir. Değerleri nesneden nesneye değişmez.
  - **Yerel değişkenler** (**local variables**): Geçici değişkenlerdir.
- İlk ikisine **üye değişkenleri** (**member variables**), veri üyeleri (data members) ya da alanlar (fields) denir ve sınıf blokunda tanımlanır.
- Bu bölümde önce nesne ve sınıf değişkenlerini ele alacağız.

Nesne Değişkenleri

- Nesne değişkenleri (instance variables ya da fields), fonksiyonel olarak, nesnenin özelliklerini ifade ederler.
  - Nesne değişkenleri, yapısal olarak referans değişkeni olabildiği gib basit değişken de olabilir.
  - Nesne değişkenleri, sınıfın içinde ama metot ya da başlatma bloğu (initializer block) gibi herhangi bir alt blok dışında, herhangi bir yerde tanıtılmalıdır,
    - Genelde sınıfın en başında tanıtılırlar.
  - Nesne değişkenleri ilk değerlerini tanıtılırken alabildikleri gibi dah sonra bir metot içinde, genelde kurucu metotta da alabilirler.
  - Tanımlanacak nesne değişkeni sayısında bir kısıtlama yoktur.

Car Sınıfı

- Aşağıda sadece özelliklerden oluşan, bir davranışa sahip olmayan **Car** sınıfı tanımlanmıştır.
- **Car**'nin özellikleri, basit ve karmaşık veri tiplerinden değişkenlerle betimlenmiştir.
- **Car**'nin nesne değişkenleri sadece tanıtılmışlardır, bir ilk değer almamışlardır.

```
public class Car { // attribute only Car
    String make;
    String model;
    String year;
    int speed;
    int distance;
}
```

Özelliklere Erişim

- Yaratılan nesnenin özelliklerine, nesnenin referansı yoluyla erişilir.
- Erişim "." notasyonuyla olur:

```
reference.attribute
```

- Bu şekilde erişilen özelliğin değerine ulaşılabileceği gibi, ona uygun atamalar da yapılabilir:

```
carObject.speed = 60;
System.out.println(carObject.speed);
```

```
public class Car {
    String make;
    String model;
    String year;
    int speed;
    int distance;
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.make = "Mercedes";
        myCar.model = "E200";
        myCar.year = "2011";
        myCar.speed = 80;
        myCar.distance = 37_650;
        System.out.println("My Car: " + myCar.year + " " +
                           myCar.make + " " + myCar.model);

        Car yourCar = new Car();
        yourCar.make = "Toyota";
        yourCar.model = "Camry";
        yourCar.year = "2011";
        yourCar.speed = 0;
        yourCar.distance = 60_000;
        System.out.println("Your Car: " + yourCar.year + " " +
                           yourCar.make + " " + yourCar.model);
    }
}
```

Nesne Değişkenlerinin İlk Değeri

- Java'da bir değişkene ilk değer atamadan, yani onu tanımlamadan, kullanamayacağımızı daha önce belirtmiştik.
  - Çünkü değişkenlere bir ilk değer atanmazsa Java, otomatik olarak atamıyordu.
  - Bu durum sadece yerel değişkenler için geçerlidir.
- Java'da nesne değişkenlerine bir ilk değer atamazsak bile, derleyici onlara bir ilk değer atar.
  - Bu ilk değerler, değişkenlerin tipleriyle uyumlu ve en az bilgi ile verilen değerlerdir.
- Sizce, Java, yerel değişkenlere bir ilk değer vermezken neden nesne değişkenlerine bir ilk değer veriyor?

InstanceVariablesInitialValues.java

Nesnelerin Durumları

- Nesneler daima anlamlı durumda olmalıdırlar.
- Nesnelerin durumlarının anlamlı olması, programcının sorumluluğundadır.
- JVM, nesnelerin olabildiğince anlamlı durumda olmaları için nesne değişkenlerine bir ilk değer veriyor.
- Aksi taktirde nesneler, ilk oluşturulduklarında kullanılamaz durumda olurlardı.

Nesne ve Referans

- Nesne ve referans, farklı kavramlardır.
  - İkisinin de tipi vardır.
  - Nesne, sınıftan türetilen ve bir duruma ve bir grup davranışa sahip olup, heap isimli bellek alanında yaşayan yapıdır.
  - Referans ise, nesneye ulaşmamızı sağlayan ve stack veya heapte bulunan bir değişkendir.
  - Bir referans ne zaman stackte ne zaman heapte bulunur?
- Bir referans, zamanın farklı anlarında, kendi tipinden farklı nesneleri gösterebilir.
  - Fakat bir anda sadece tek bir nesneyi gösterir.
  - Bir nesneye birden fazla referans olabilir.

Nesne Referansı II

- Referans ile nesne kavramlarının farklılığına güzel bir örnek de referansın, farklı zamanlarda kendi tipinden farklı nesneleri gösterebilmesidir:

```
Car myCar = new Car();
myCar.make = "Mercedes";
...
Car yourCar = new Car();
yourCar.make = "Toyota";
...
Car tmpCar = myCar;
myCar = yourCar;
yourCar = tmpCar;
System.out.println("My Car: " + myCar.year + " " +
                   myCar.make + " " + myCar.model);
System.out.println("Your Car: " + yourCar.year + " " +
                   yourCar.make + " " + yourCar.model)
```

Nesneler

- Belleğiniz elverdiği ölçüde, aynı sınıftan pek çok nesne oluşturabilirsiniz.
- Önemli olan her bir nesnenin, programladığınız iş alanındaki iş nesnelerine (business objects) karşılık gelmesidir.
- Genel olarak nesnelerin farklı durumlara sahip olması beklenir:
  - Özellikle veri tabanında anahtar alan (primary key) olan veriler, h nesne için farklı bir değer almak zorundadırlar.

Bileşik (Composite) Nesneler

- Farklı türden nesnelerden oluşan nesnelere, **bileşik nesne** (**composite object**) denir.
- Bu durumda bir sınıf, diğer sınıfın tipinden nesne değişkenlerine sahip olur.
  - Daha doğru ifade ile bir nesne, diğer nesnenin referansına sahip olur.
- Daha önce tanımladığımız **`Car`** de String tipinden değişkenlere sahipti ve bu yüzden de bir bileşik nesne idi.
  - Fakat bileşik nesne ile biz, daha çok, bizim oluşturduğumuz tiplerden nesnelere sahip bileşik nesneleri kastederiz.

Sınıf mı Nesne mi?

- Sizin de farkettiğiniz gibi "sınıf" dememiz gereken durumlarda bile "nesne" terimini kullanmamız, çok sık rastlanan bir durumdur.
- Dolayısıyla "nesne" kelimesi ile ne zaman nesneyi ne zaman sınıfı kastettiğimiz cümlenin gelişinden anlaşılır.
- İlk başlarda karışık gibi görünse de gittikçe zihniniz buna alışacaktır.

TestAttributeOnlyComposite.java

- Nesneler arasındaki ilişkilerin, referansları üzerinden kurulduğuna dikkat edin!
  - Örnekte "**`person1`**" ve "**`car1.owner`**" aynı **`Person`** nesnesine "**`car1`**" ile "**`person1.car`**" da aynı **`Car`** nesnesine referanstırlar.
- **`PersonAttributeOnlyComposite`** nesnesi ile **`CarAttributeOnlyComposite`** nesnesi arasında iki yönlü (bi-directional) ve 1-1 (One-to-One) ilişki vardır.

TestAttributeOnlyComposite.java

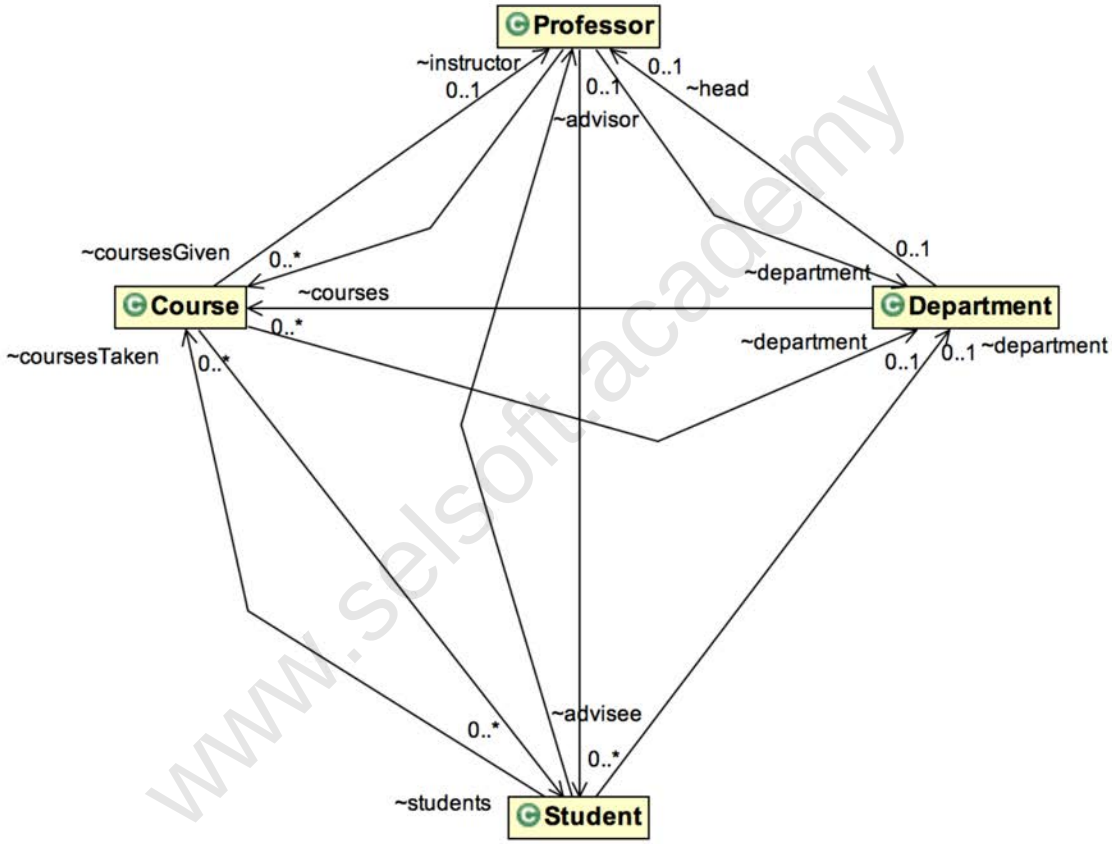
```
PersonAttributeOnlyComposite person1 =
                          new PersonAttributeOnlyComposite();
...
CarAttributeOnlyComposite car1 =
                          new CarAttributeOnlyComposite();
...
car1.owner = person1;
person1.car = car1;

System.out.println("Make of owner1's car: " + person1.car.make);
System.out.println("First name of car1's owner: " +
                   car1.owner.firstName);
```

Nesneler Arası İlişki

- Nesneler arasındaki ilişkilerin, nesnelerin referansları üzerinden kurulması, aynı nesnenin birden fazla yerde olmasının gerektiği durumlarda, aslında nesnenin bir tane olup, gerçekte farklı yerlerde bulunan şeyin, o nesneye olan değişik referansların olmasını sağlar.
- Aksi taktirde bellekte örneğin aynı **`Student`** nesnesinden birden fazla olurdu ve nesnelerden birine bir değişiklik yapıldığında diğerleri geçersiz duruma düşerdi.
  - Gerçekteki tek bir nesnenin, bellekte durumları farklı birden fazla kopyasının olması pek de hoş bir durum oluşturmazdı.
- Nesne-merkezli yazılım, birbirleriyle referansları üzerinden haberleşen nesnelerden oluşmaktadır.

University Example



Uygulama

- **UniversityExample** örneğindeki test sınıfındaki main metoda aşağıdaki yenilikleri ekleyin:
  - *Yeni iki profesör* ,
  - Yeni üç ders,
  - Yeni üç öğrenci
- Öyle ki bu yeni nesneler arasında örnekteki gibi ilişkiler kurgulayın.

Metotlar

Metotlar

- Metotlar, nesnelerin sorumluluklarını yerine getiren fonksiyonlardır.
  - Eskiden bu yana programlama dillerinde subroutine, procedure, subprogram, vb. isimlerle ifade edilmiştirler.
- Metotlar, bir nesnenin dışarıya verdiği hizmeti ya da bir başka deyişle, bir nesnenin alabileceği mesajları ifade eder.
- Metotlar, sadece bir sınıfın içinde tanımlanabilir.
  - Bir sınıfın tanımlamadığı bir metot, o sınıfın nesnelerinde çağrılamaz.
- Bir sınıfta istenildiği kadar metot tanımlanabilir.
- Bir sınıftan yaratılan bütün nesneler, o sınıfta tanımlanan metotları yerine getirir, yani hepsi aynı mesajları alır.

Metot I

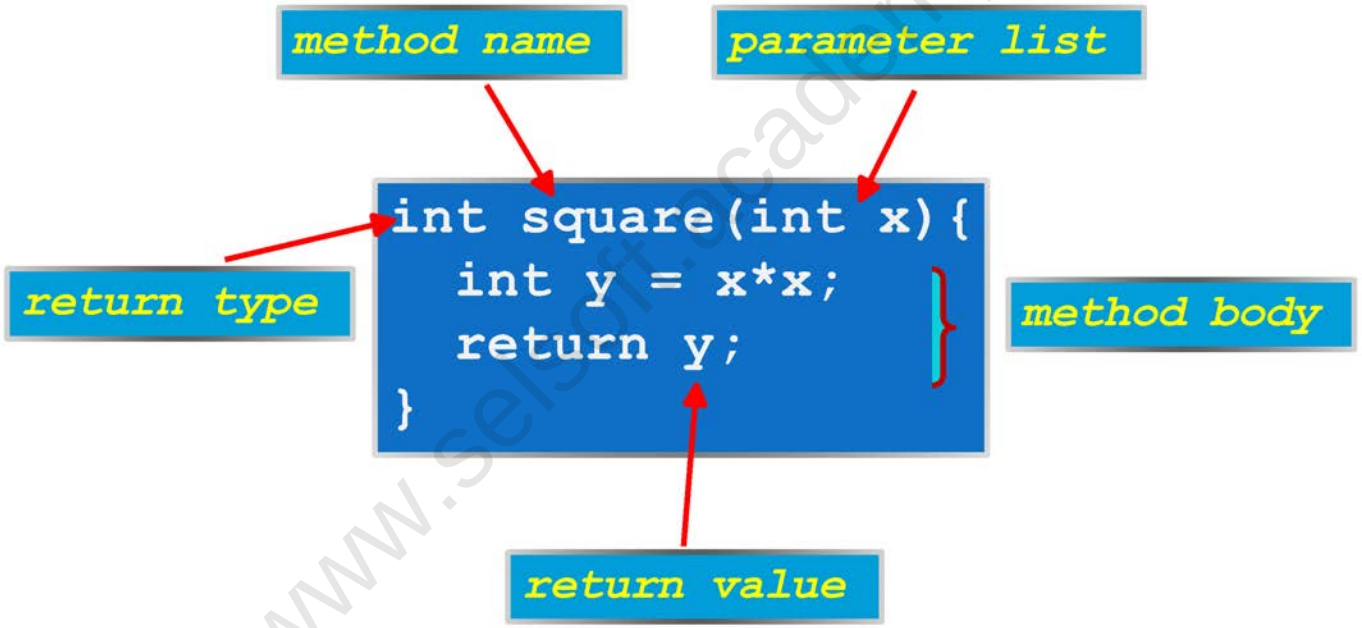
- Java'da bir metotun altı ana parçası vardır:
  - **İsim**: Anlamlı ve genelde emir kipinde bir isim.
  - **Parametreler (parameters)**: Dışarıdan geçilen parametreler.
  - **Dönüş değeri tipi (return type)**: Çağrıldığı ortama döndüreceği değerin tipi.
  - **Gerçekleştirme (body, implementation)**: İfadelerden oluşan, çalı kısım.
  - **Niteleyiciler (modifiers)**: Farklı amaçlar için değişik niteleyici anahtar kelimeler kullanılabilir.

```
<niteleyici>* <dönüş tipi> <isim>(<Parametre>*)
        throws <exception>* {
    <kod>*
}
```

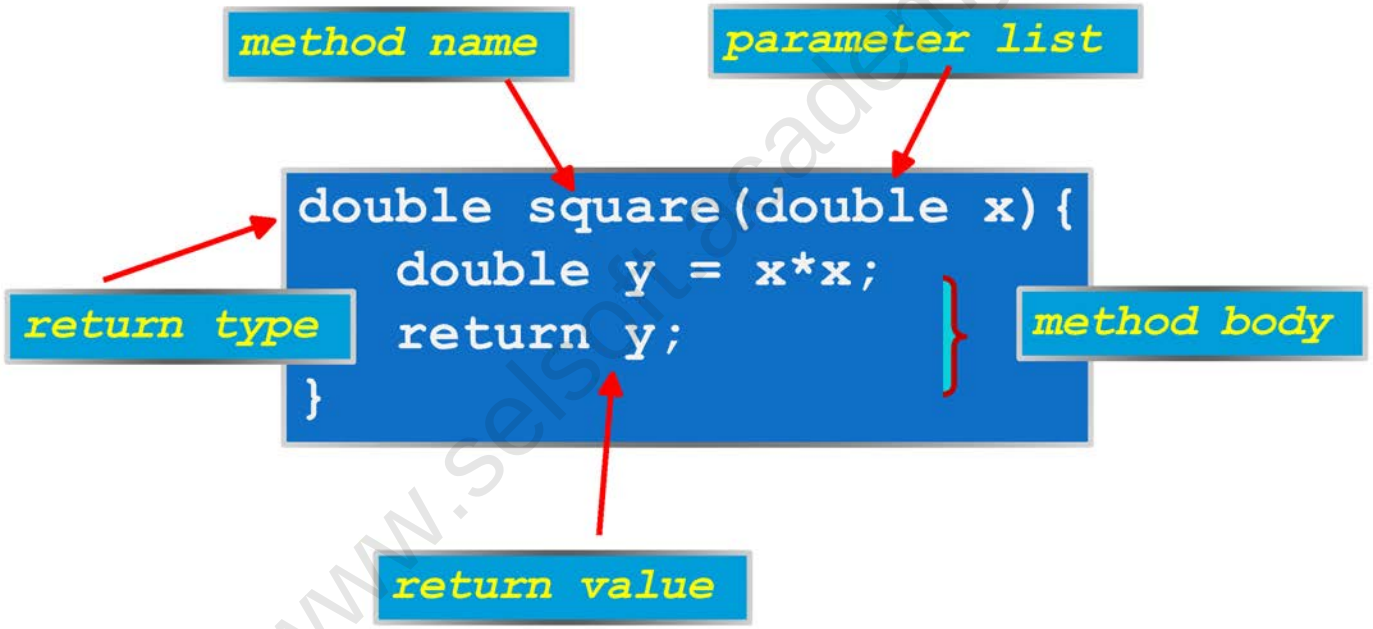
Metot - II

- Bir metodun sıfır ya da daha fazla sıra dışı durum (exceptions) fırlatabilir.
  - Bu durumda metot tanımında bu sıra dışı durumlar listelenirler.
- Sıra dışı durumlar (exceptions) daha sonra, farklı bir bölümde ele alınacaktır.
- Dolayısıyla bu bölümde metotların herhangi bir sıra dışı durum fırlatmadığı varsayılacaktır.

Metot Örneği



Metot Örneği



Metotlara Erişim

- Yaratılan nesnenin metotlarına, nesnenin referansı yoluyla erişilir.
- Erişim "." notasyonuyla olur:

```
reference.method()
```

Parametreler

- Parametre listesi, virgülle ayrılmış, "tip, isim" listesidir.
- Parametreler, basit tip olabileceği gibi referans tipler de olabilirler.
- Bir metot dışarıdan arzu edildiği kadar parametre alabilir.
  - Bir metot dışarıdan parametre almak zorunda değildir.
  - Referans parametresi olarak "null" da geçilebilir.
  - Basit tipler için "null" geçilemez.

Metot Örneği

```
int i = 4;
int iSquare = obj.square
```

returned value

actual parameter

```
int square(int x){
  int y = x*x;
  return y;
}
```

formal parameter

Gerçek ve Formal Parametreler

- Gerçek parametre (actual parameter), metot çağrıldığında ona geçilen parametrelere denir.
  - Gerçek parametrelere daha sıklıkla *argüman* da denir.
- Formal parametre (formal parameter), metot tanımında olan parametrelere denir.
- Metodun formal parametreleri ile metot çağrılırken geçilen gerçek parametreler sayı olarak aynı olmalıdır.
- Formal parametlere ile karşılık gelen gerçek parametreler tip bakımından uyumlu olmalıdır.
  - Otomatik yükseltmeler çevirme (cast) operatörüne ihtiyaç duymadan yapılır.
  - Dönüştürülebilen gerçek parametreler için çevirme (cast)

Gerçek ve Formal Parametre Uyumu

```
public void f(Student s, int i, boolean b){
    ...
}

x.f(new Student(), 5, true);

byte b = 3;
x.f(new Student(), b, false); // Yükseltme

double d = 3.14;
x.f(null,(int)d, false);      // Çevirme
```

Varsayılan Değerli Parametre

- Java'da metotlar, varsayılan değere sahip parametrelerle tanımlanamaz.

Dönüş Tipi (return type) I

- Dönüş değeri, basit tip olabileceği gibi referans tip de olabilir.
- Dönüş tipi ile metot ismi arasına hiçbir şey giremez.
- Metot, dönüş değeri tipine uygun bir değer döndürmek zorundadır.
  - Değer döndürülürken, yükseltme ya da çevirme (cast) yapılabilir.
- Bir metot en fazla bir değer döndürebilir.
  - Birden fazla değer döndürmek söz konusuysa, dizi gibi torba (collection) yapıları kullanılabilir.

Dönüş Tipi II

- Metotun çağrıldığı ortam, metotun dönüş değerini almak istiyorsa, uygun bir atama yapmalıdır.
  - Dönüş değeri ile ilgilenilmediği durumlarda atamaya da gerek yoktur. Buna bir metotu yan etkisi (side effect) için çağırmak (

```
int x = a.f(8); // Normal call with return value
a.f(8.37);      // Call for side effect
```

- Bir metot, bir değer döndürmek zorunda değildir.
  - Bu durumda dönüş tipi, "**void**" olarak ifade edilir ve
  - "**return**" hiç kullanılmayabilir ya da "**return;**" ile yetinilir.
  - Bu durumda metotun çağrıldığı ortamda metottan herhangi bir atama yapılamaz.

Metot Örnekleri - I

```
boolean flag() { return true; } => boolean b = flag();

float naturalLogBase() { => float e=naturalLogBase();
    return 2.718f;
}

void doNothing() { return; }     => doNothing();

void doNothingElse() {}              => doNothingElse();
```

Metot Örnekleri - II

```
int x = a.f(8);
```

```
// Possible methods
byte f(int i){ ... }

char f(int i){ ... }

short f(int i){ ... }

int f(int i){ ... }

byte f(long l){ ... }
...

byte f(float f){ ... }
...

byte f(double d){ ... }
...
// 4 * 7 = 28 possible methods
```

Metotlar Ne İçindir?

- Metotlar temel olarak şu üç amaçla yazılırlar:
  - Nesnenin durumu hakkında bilgi vermek (accessor methods),
    - getXxx() şeklindeki metotlar.
  - Nesnenin durumunu değiştirmek (mutator methods),
    - setXxx() şeklindeki metotlar.
  - Bir hesaplama, bir sürecin adımı vb. cinsinden bir iş yapmak,
    - computePrice(), startCommunication(), fetchAccountsByType(), authenticateUser() vb. metotlar.
- Nesne yukarıdaki işleri yaparken, özellikle de üçüncüsünde başka bir nesneden hizmet alabilir (delegation).
- Metotlar, yukarıdaki üç şeyden bir ya da bir kaçını yerine getirirler.

Calculator.java

- Calculator'e üs alan (first ^ second) yeni bir metot yazın ve main metottan çağırarak test edin.

Set/Get Metotları

- set/get metotları (setter and getters) nesnesin durumunu değiştirmek ya da öğrenmek amacıyla kullanılan standart metotlardır.
- set metotları void döndürür ve durumu değiştireceği alanın tipinden argüman alır,
- get metotları ise argüman almaz ama değerini döndüreceği alanın tipinden argüman alır.
  - boolean alanların get metotları isXxx() şeklinde de yazılabilir.
- Eclipse vb. IDE'ler set/get metotlarını otomatik olarak oluşturan kısa yollara sahiptirler.

Window.java

Sağlıklı Metotlar İçin İlkeler

- Metotları oluştururken şu ilkeleri uygulamak, sınıfların sağlıklı olması açısından önemlidir:
  - Metotlar, sadece ve sadece içinde tanımlandığı sınıfın soyutladığı kavramın sorumluluk alanına giren davranışları yerine getirmelidirler.
  - Her bir metot, sadece ve sadece bir şeyi yerine getirmelidir.
    - Bu yüzden daha az satıra sahip ve daha az argüman alan metotlar tercih edilmelidir.
    - Yazmaya başladığınız metotun büyümeye ve karmaşıklaşmaya başladığı hissettiğinizde, yeni metotlar oluşturarak, metotunuzu en az karmaşıklıkta tutmaya özen gösterin.
  - Metotlarınıza, yaptıkları işlere uygun emir kipinde isimler verin.
    - Java'da metot ismi uzunluğunda bir sınır yoktur.

DirtyCalculator.java

- **`DirtyCalculator`**'ın neden problemli olduğunu tartışın.

Car.java, Person.java ve Test.java

- **`Car`** sınıfı, **`CarAttributeOnlyComposite`**'in metotlara sahip halidir.
- **`Person`** sınıfı da, **`PersonAttributeOnlyComposite`**'in bir metota sahip halidir.
- Test sınıfı da **`Test`**'dir.

Niteleyiciler (Modifiers) - I

- Java'da metotlar için aşağıdaki niteleyiciler kullanılabilir:
  - Erişim niteleyecileri: **public**, **protected**, **private**
  - **abstract**
  - **static**
  - **final**
  - **synchronized**
  - **native**
  - **strictfp**
- Her bir niteleyici en fazla bir kere kullanılabilir.
- Sıralama önemli olmamakla birlikte niteleyicilerin yukarıdaki sırayla yazılmaları bir gelenektir.

Niteleyiciler (Modifiers) - II

- abstract ve native niteleyicilerinden birine sahip bir metodun gerçekleştirmesi olamaz.
- abstract niteleyicisi, final, synchronized, native ve strcitfp niteleyecilerinden biri ile birlikte kullanılamaz.
- native ve strcitfp niteleyecileri de birlikte kullanılamaz.

- Niteleyiciler ileriki konularda, yeri geldikçe işlenecektir.

MethodModifiers.java

Uygulama

- Circle (ya da Daire) isimli bir sınıf oluşturup aşağıdaki değişken ve metotlara sahip olmasını sağlayın:
  - *double* tipinde bir yarıçap,
  - Alan ve çevre hesaplayıp *double* olarak döndüren iki metot.
- Daha sonra *main* metota sahip olan ve bu metotta iki tane Circle/Daire nesnesi oluşturup, bu nesnelerin alan ve çevrelerini hesaplatıp ekrana basan bir CircleTest/DaireTest sınıfı oluşturun ve çalıştırın.

İmza ve Arayüz - I

- Bir metotun, isim ve parametre listesinden oluşan bilgisine **imza** (**signature**) denir.
  - Dönüş değeri ve fırlatılan sıra dışı durumlar imzaya dahil değildir.
- Bir metotun, isim, parametre listesi, dönüş değeri tipi ve fırlattığı dıra dışı durumlardan oluşan bilgisine **arayüz** (**interface**) denir.

```
<niteleyici>* <dönüş tipi> <isim>(<Parametre>*) throws <exception>*

public double squared(double arg) throws IllegalArgumentException

public double squared(double arg) throws IllegalArgumentException
```

imza (signature)

arayüz (interface)

İmza ve Arayüz - II

- Bir metot ancak arayüz bilgisiyle çağrılabilir.
  - Arayüz bilgisini daha anlaşılır kılmak amacıyla JavaDoc ile arayüz dokümantasyonu (interface documentation) yapılabilir.
- Bir sınıfta imzası (ya da arayüzü) aynı olan iki tane metot olamaz.

Nesnenin Arayüzü

- Bir nesnenin sahip olduğu metot arayüzlerinin tamamına, o nesnenin arayüzü denir.
- Dolayısıyla nesne arayüzü, nesnenin sınıfında tanımlanan metot arayüzlerinin toplamıdır.

Overloading (Çoklu Kullanım)

Overloading

- Bir isim, bir sınıftaki birden fazla metotta kullanılabilir.
  - Bu duruma **çoklu kullanım** (**overloading**) denir.
- Overload edilen metotların parametre listesi, sayı ve/veya tip bakımından farklı olmalıdır.
  - Bir sınıfta imzası aynı olan iki tane metot olamaz.
- Overloading, genelde, aynı işi farklı parametrelerle yapan metotlar için kullanılır.
  - `System.out.println()` ve `System.out.print()` metotları

CalculatorOverloaded.java

Bir Soru?

- İmzası aynı olup da dönüş tipleri farklı olan metotlarla overload yapılamaz. Neden?

```
boolean method(int i, long l) {
    return true;
}

int method(int x, long y) {
    return 5;
}
```

Bir Soru?

- Çünkü bir metot döndürdüğü değer alınmadan da çağrılabilir.
  - Yan etkisi için çağırma (side-effect call)

```
boolean method(int i, long l) {
    return true;
}

int method(int i, long l) {
    return 5;
}

...

method(5, 8L); // Hangisi çağrılır?
```

PrimitiveOverloading

- Parametrelerde bir tip gerektiğinde otomatik olarak bir üst tipe yükseltilebilir.
- Peki alternatif olarak birden fazla üst tip varsa hangisine yükseltilir?
  - En az yükseltme gerektiren tipe mi yoksa standart bir tipe mi?
  - Hiç bir uygun üst tip yoksa?

```
byte b = 3;
x.method(b);  // Hangisi çağrılır?

void method(short i) {...}

void method(int i) {...}
```

PrimitiveOverloading.java

Değişen Argümanlı Metotlar

Değişen Argümanlı Metotlar - I

- Bir sınıfta bir metodun aynı tipten farklı sayıda parametre alan overloaded halleri olabilir.

```
void calculateAverage(int i, int j) { }
void calculateAverage(int i, int j, int k) { }
void calculateAverage(int i, int j, int k, int l) { }
void calculateAverage(int i, int j, int k, int l ...) {
}
...
```

  - Çünkü ne kadar argüman geçileceğini bilemezsiniz.

Değişen Argümanlı Metotlar - II

- Bu durumda metot, parametre olarak dizi (array) alacak şekilde yazılabilir.

```
void calculateAverage(int[] array) { }
...
int[] a1 = {1, 2};
calculateAverage(a1);
```

- Ama l aima dizi o

```
int[] a1 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
calculateAverage(a1);
```

Değişen Argümanlı Metotlar - III

- Bu amaçla Java SE'ye 5. sürümle beraber gelen değişen argümanlı metotlar (var args methods) kullanılabilir.
- Metot, son parametresi aşağıdaki gibi "**...**" ile tanımlanır.

```
void calculateAverage(int ... numbers)
{
   double sum = 0;
   for(int i : numbers)
      sum += i;
   double average = sum/numbers.length
}

calculateAverage(1,2,3);
calculateAverage(1,4,7,9,11,14);
calculateAverage(1,4,7,9,11,14,21,29,35);
```

- Bu duru çinde dizi (array)

Değişen Argümanlı Metotlar - IV

- Değişen argümanlı metotlarda dizi argümanı en son geçilen argüman olmalıdır.
- Bu yüzden değişen argümanlı metoda sadece bir tane "**...**" argümanı geçilebilir.
- "**...**", "[]" gibi tip ile isim arasında herhangi bir yerde olabilir.
- Örneğin, main metod aşağıdaki şekillerde de yazılabilir.

```
public static void main(String... args)
public static void main(String ... args)
public static void main(String ...args)
```

VarArgMethodDemo.java

Kurucular (Constructors)

Nesne Durumuna İlk Değer Atama

- Şu ana kadar, nesneyi oluşturduktan sonra, nesnenin alanlarına, referansı üzerinden "." ile ulaşıp, onlara değerler atadık ve nesneyi anlamlı bir duruma getirdik.

```
Person person = new Person();
// Anlamsız durum
person.tckn = "1";
person.firstName = "Zeynep";
person.lastName = "Kaya";
```

- Yukarıdaki gibi, nesnenin durumunun anlamsız olduğu bir anı ortadan kaldırmak için, nesneyi oluştururken bir metot çağrısı yapıp, nesnenin alanlarına ilk değer atayabiliriz.
  - Nesnenin durumu, referansı döndüğünde, anlamlı olur.

Kurucu I

- Nesne oluşturulurken çağrılan özel metota **kurucu**/**yapılandırıcı** (**constructor**) denir.
- Kurucu metotlar, bazı açılardan özel metotlardır ve sadece nesne oluşturuken çağrılırlar.
- Kurucular, **new** anahtar kelimesiyle kullanılırlar.
  - Java'da kurucu çağrısı yapmadan oluşturulabilen sadece 2 tip vardır: String ve dizi (array)
  - Diğer her türlü nesne, ancak ve ancak kurucu ile oluşturulur.
- Kurucu metotları ile nesnenin durumunun ilk halini alması sağlanır.
- Bu amaçla kurucu metotlar parametre tanımlayabilir ve bu durumda onlara değer geçilir.

Kurucu II

- Kurucunun ismi, içinde tanımlandığı sınıfın ismiyle aynı olmalıdır.
- Kurucunun dönüş tipi, dolayısıyla da dönüş değeri yoktur.
- Kurucular overload edilebilirler.
  - Bu durum, farklı nesne oluşturma şekillerine karşılık gelir.
- Hiçbir argüman almayan kurucuya **varsayılan kurucu** (**default constructor** ya da **no-arg constructor**) denir.
- Argüman alan kuruculara **akıllı kurucu** (**smart constructor**) denir.

Varsayılan Kurucu I

- Bir sınıfta hiç bir kurucu tanımlanmazsa, derleyici, derlenmiş . class kodunda bir tane varsayılan kurucu sağlar.

```
public class TreeWithNoConstructor {
   private String type;
   private float height;

   public static void main(String[] args) {
      TreeWithNoConstructor tree = new
                              TreeWithNoConstructor();
      tree.type = "Oak";
      tree.height = 8.74f;
   }
}
```

Varsayılan Kurucu II

- Varsayılan kurucuyu siz de sağlayabilirsiniz, ama dışarıdan argüman almadığından, nesneleri yaratanın, nesnelerin ilk durumları ile ilgili bir inisiyatifi söz konusu olmayacaktır.

```
public class TreeWithDefaultConstructor {
    private String type;
    private float height;

    public TreeWithDefaultConstructor(){
        type = "Pine";
        height = 1.0f;
    }

    public static void main(String[] args) {
        TreeWithDefaultConstructor tree = new

        TreeWithDefaultConstructor();
        tree.printInfo();
    }
}
```

Argümanlı Kurucu

- Kurucular, argüman alabilirler.
  - Bu durumda, kurucu, yaratılan nesnenin ilk halini belirleyecek şekilde dışarıdan argüman geçilerek çağrılır.
- Derleyici, hiç bir kurucu olmadığında, varsayılan kurucuyu otomatik olarak sağlamasına karşın, bir kurucu olduğunda bu iyiliğine son verir.
- Argüman alan bir kurucunun varlığında, derleyicinin, varsayılan kurucuyu sağlamamasının sebebi ne olabilir?

TreeWithArgumentConstructor.java

Overloaded Kurucular

- Bir sınıfın overloaded pek çok kurucusu olabilir.
- Bu durum, farklı parametrelerle farklı nesne oluşturma şekillerine karşılık gelir.
- Tasarım sırasında nesnelerin hangi kurucularla oluşturulacağına karar vermek gereklidir.
- Karar verilecek şey, "bir sınıfın nesnesi, hangi verilerle üretilebilmelidir?" sorusudur.

TreeWithOverloadedConstructors.java

Uygulama

- Daha önce oluşturduğunuz Circle (ya da Daire) isimli sınıfa iki tane kurucu koyun:
  - Varsayılan kurucu yarı çapı 10 olan bir Circle nesnesi oluşturur.
  - Argüman alan kurucu ise geçilen değeri, yarıçapa atar.
- Sonra CircleTest (ya da DaireTest) sınıfında, bu iki kurucu ile oluşturulan nesnelerinizin alan ve çevrelerini hesaplayıp ekrana basın.

Uygulama

- Daha önce yaptığımız University örneğindeki sınıflara uygun kurucu metotlar koyun.

Kurucuların Birbirlerini Çağırmaları

- Kurucular, bazen yapacakları işi, diğer kurucuların yardımıyla yapabilirler.
- Bu durumda kurucuların birbirlerini çağırmaları gerekir.
- Bu ise "**this**" anahtar kelimesi ile yapılır.

```
public TreeWithThis(String newType, float newHeight)
    type = newType;
    height = newHeight;
}
public TreeWithThis(String newType) {
    type = newType;
    height = 1.0f;
}
public TreeWithThis(float newHeight) {
    type = "Pine";
    height = newHeight;
}
```

TreeWithThis.java

- **this** ile, aynı metotlarda olduğu gibi, imzası uyan kurucu metot çağrılır.
- *this* çağrısı, bir kurucuda ilk çalışan satır olmalıdır. Neden?

```
public TreeWithThis(String newType, float newHeight) {
      type = newType;
      height = newHeight;
}
public TreeWithThis(String newType) {
      this(newType, 1.0f);
}
public TreeWithThis(float newHeight) {
      this("Pine", newHeight);
}
```

Uygulama

- Daha önce oluşturduğunuz Circle (ya da Daire) isimli sınıftaki varsayılan kurucuya yapacağınız değişiklikle, onun this() çağrısı ile, argüman alan kurucuyu çağırarak nesne oluşturmasını sağlayın.

Uygulama

- Daha önce yaptığımız University örneğindeki sınıfların kurucularının birbirlerini this() ile çağırmalarını sağlayın.

TreesWithoutReferences.java

- Daha önce, nesne ve onlara olan referansların ayrı şeyler olduğunu ve bir nesneye birden fazla referans olabileceği gibi, bir referansın da farklı zamanlarda aynı tipten farklı nesneleri gösterebileceğini belirtmiştik.
- Benzer şekilde bazen referans olur ama hiç bir nesneyi göstermez.
  - Null reference, null pointer.
- Bazen de nesne olur ama referansı olmaz.
  - Lost objects ya da memory leaks!

this Anahtar Kelimesi I

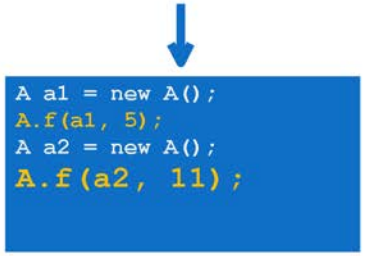
- Bir nesnenin durumunu oluşturan alanlarının değerlerinin, nesnenin, heap ismi verilen, dinamik bellekte, kendisine ayrılmış olan yerinde tutulduğunu daha önce ifade etmiştik.
- Peki, nesnelerin metotları nerede tutulurlar?
  - Metotların kodu, belleğin "code" kısmında tutulur ama her meto çağrısı için belleğin "stack" kısmında bir pencere (frame) açılır ve ilkel olsun, referans olsun, metotun yerel değişkenleri burada oluşturulur.
- Çağrılan bir metot, kendisi için çağrıldığı nesneyi bilmek zorundadır.
  - Çünkü, nesnesinin değişkenlerini kullanabilir.

this Anahtar Kelimesi II

- Bir nesne metotunda, o metotun kendisi için çağrıldığı nesnenin referansı, **this** ile ifade edilir.
- Bu durum, sanki, o metoda, üzerinde çağrıldığı nesnenin gizli bir referansla geçilmesi ile açıklanabilir.

```
class A{
   void f(int x){ ... }
}
```

```
A a1 = new A();
a1.f(5);
A a2 = new A();
a2.f(11);
```



this Anahtar Kelimesi III

- this, genel olarak şu 2 yerde kullanılır:
- Kurucu ya da bir nesne metotunda, aynı isimde bir yerel değişken olduğunda, nesne değişkenine ulaşmak için.
  - Bu durum, genelde aynı isim, hem nesne hem de yerel değişkend kullanıldığında söz konusu olur.
  - Böyle bir durum yoksa this kullanmaya da gerek yoktur.

```
public class Tree{
   String type;
   float height;
   public Tree(String type, float height){
      this.type = type;
      this.height = height;
   }
}
```

this Anahtar Kelimesi IV

- Herhangi bir sebeple, üzerinde metotun üzerinde çağrıldığı nesneye ulaşmak için.

```
public class Tree{
   String type;
   float height;

   public Tree grow(){
      height++;
      return this;
   }
}
```

Tree.java ve Bank.java

this Anahtar Kelimesi V

- Aşağıdaki durumlarda this kullanmaya gerek yoktur:

```
public class Tree{
   String type;
   float height;

   public void printInfo(){
      this.printType();
      this.printHeight();
   }
   public void printType(){
      System.out.println("Type: " + this.type);
   }

   public void printHeight(){
      System.out.println("Height: " + this.height)
   }
}
```

Uygulama

- Daha önce oluşturduğunuz Circle (ya da Daire) isimli sınıftaki kuruculara geçilen parametreleri, nesne değişkenleriyle aynı isimde yapın ve kurucu içindeki atamalarda nesne değişkenlerine ulaşmak için "this" referansını kullanın.
- Benzer şeyi set metotları için de yapın.

Parametre Geçme I

- Metot çağrılırken, çağrıldığı yerde metota gerçek parametreler/argümanlar geçilir ve bu argümanlar metot içinde formal parameterlerle temsil edilir.
- Geçilen argümanlarla metot parametreleri arasındaki ilişki nasıldır?

```
public class ParameterPassing{
   public void f(int k){
      k++;
   }
   public static void main(String[] args){
      ParameterPassing o = new ParameterPassing()
      int i = 5;
      o.f(i);
      System.out.println(i);
   }
}
```

```
public class ParameterPassing{

   public void f(A aa){
      aa.j++;
   }

   public void f(int k){
      k++;
   }

   public static void main(String[] args){
      ParameterPassing o = new ParameterPassing()

      int i = 5;
      o.f(i);
      System.out.println(i);

      A a = new A();
      o.f(a);
      System.out.println(a.j);
   }
}

class A{ int j = 5;}
```

Parametre Geçme II

- Java'da parametreler, değerleri ile geçilirler (pass-by-value) .
  - Bir metoda bir basit değişken geçilirken, parametrenin değeri, gerçek değişkenin değeri olarak belirlenir,
  - Benzer şekilde geçilen bir nesne ise, bu durumda o nesnenin referansının değeri, parametre değeri olarak belirlenir. (Aslında "nesne geçme" ifadesi doğru değildir, çünkü hiç bir zaman nesneye doğrudan ulaşamayız, sadece referansına ulaşabiliriz. Dolayısıyla geçilen nesne değil, referansıdır.) Referansın değeri ise zaten nesnenin adresidir.
- Dolayısıyla, her halükarda geçilen şey, gerçek argümanın değeridir.

Parametre Geçme III

- Java'da argümanlar değerleriyle geçildiklerinden, bir metota geçilen bir basit değişkenin değeri metotta değişse bile, asıl değerinde bir değişiklik olmaz.
  - Çünkü değişen parametredir, gerçek değişkenin değeri değişmez.
- Nesneler için ise, metotta parametre olarak yeni bir referans oluşturulduğundan ve her iki referans da aynı nesneyi gösterdiğinden, birbirlerinin değişikliklerinden haberdar olurlar.
  - Bu yüzden Bruce Eckels gibi bazı yazarlar, anlamayı kolaylaştırmak için, "Java'da nesneler referanslarıyla geçilir (pass-by-reference)" derler.

ObjectPassing.java

Nesnelerin Referansını Geçme

- Java'da referans ile geçme, aslında referansın değerinin kopyalanmasından başka birşey değildir.
- Yani formal parametre, nesnenin referansının gösterdiği bellek adresini gösterecek şekilde oluşturulur.
- Bu yüzden metota geçilirken oluşturulan referansa, formal parametreye, daha sonra başka bir nesne atanabilir.

```
public static void main(String[] args) {
   ObjectPassing o = new ObjectPassing();

     W w = new W(3, false);
     o.f(w);
     System.out.println("i of w is " + w.i + "  and
                         b of w is " + w.b);
}

public void f(W objectW) {
     objectW.i = 5;
     objectW.b = true;
     W ww = new W(8, true);
     objectW = ww;
     objectW.i = 12;
     objectW.b = false;
     System.out.println("i of objectW is " + objectW.i + "
                    and b of objectW is " + objectW.b);
}
```

Uygulama

- İki boyutlu koordinat sisteminde bir noktayı göstermek üzere Point (Nokta) sınıfı oluşturun. Sınıfa şunları koyun:
  - int x ve y nesne değişkenleri,
  - girilen x ve y değerlerini kullanarak (x, y)'de Point nesnesi oluşturan kurucu,
  - (0, 0)'da Point nesnesini yukarıdaki kurucuyu this ile çağırarak oluşturan varsayılan kurucu,
  - X ve Y düzlemlerinde geçilen delta kadar hareket ettiren metotlar
    - public void moveHorizontally(int delta) ve
    - public void moveVertically(int delta)
    - *public void move(int deltaX, int deltaY)*

Uygulama

- Daha önce yapmış olduğunuz Circle (Daire) sınıfını, merkezini Point (Nokta) ile temsil edecek şekilde değiştirin. Sınıfa ayrıca şunları koyun:
  - Geçilen Point nesnesi ve radius bilgisini kullanarak bir Circle nesn üreten kurucu,
  - Sadece radius bilgisini kullanarak, merkezi (0, 0) noktasında olan b Circle nesnesi üreten kurucu. Bu kurucu yukarıdaki kurucuyu *this* ile çağırmalı.
  - Daireyi, aslen merkezini, x ve y düzlemlerinde geçilen delta kadar hareket ettiren metotlar:
    - *public void moveHorizontally(int delta)* ve
    - *public void moveVertically(int delta)*
    - *public void move(int deltaX, int deltaY)*

Uygulama

- Ve Circle üzerinde bulunan diğer metotlar.

final

final Anahtar Kelimesi

- **final** anahtar kelimesi ile daha önce bir basit değişkenin nasıl bir sabite haline getirilebileceğini görmüştük.
  - **final** kullanılarak tanıtılan basit değişkenlerin değeri değiştirilemez.

```
final int i = 5;
i = 8;    // Compile-time error.
```

final Referans

- Nesneler doğrudan final yapılamaz, ancak nesnelerin alanları final yapılabilir.
  - Bu şekilde durumu değişmeyen nesne elde edilir.
- Referansın final olmasının anlamı, basit değişkenlere göre biraz farklıdır.
- final referanslar, gösterdikleri nesneden başka bir nesneyi gösteremezler.

```
final Car c = new Car();
c = new Car();     // Compile-time error.
```

```
final Car c;
c = new Car();     // Legal!
c = new Car();     // Compile-time error.
```

final Değişkenler

- final olan değişkenin, basit olsun referans olsun, tanıtıldığı yerde bir ilk değer alması zorunlu değildir.
- İlk değer ataması yapılmayan bir final değişkene boş sabite (blank final) denir ve şu üç şekilde ilk değer ataması yapılabilir:
  - Tanıtıldıktan sonra, ilk erişimde bir ilk değer vermek,
    - Bu durum özellikle yerel değişkenler için geçerlidir.
  - Kurucu metotta bir ilk değer atamak,
  - Başlatma blokunda bir ilk değer atamak (*ileride gelecek*).
- Son iki durum üye değişkenler için geçerlidir.
  - Yani final olan değişkenlere ilk ulaşımda bir ilk değer atarsanız, tanıtıldığı yerde atama zorunluluğu kalkar.

FinalCar.java

final Parametreler

- Bir metota geçilen parametrenin değerinin o metotta değişmesini önlemek istiyorsanız, parametrenizi metot arayüzünde tanımlarken final yapabilirsiniz.

```
public void speedUp(final int newSpeed){
    newSpeed = 75;        // Compile-time error.
    speed = newSpeed;
}

public void setOwner(final String newOwner){
    newOwner = "Another Owner"; // Compile-time error.
    owner = newOwner;
}
```

static

static Anahtar Kelimesi

- Her nesne bellekte, kendi durumunu ifade eden değişken kümesine sahiptir ve bu kümede bulunan değişkenlerin değerleri, diğer nesnelerdekilerden bağımsız olarak değiştirilebilir.
  - Nesne değişkenleri (instance variables)
- Bazen nesnelerin öyle özellikleri olur ki değeri nesneden nesneye değişmez, bütün nesneler için aynıdır.
  - Bu durumda o bilgiyi, her nesnede ayrı ayrı saklanacak şekilde nesne değişkeni olarak tanımlamak uygun değildir.
- Bu şekilde, aldığı değeri nesneye bağlı olmayan değişkenler "**static**" anahtar kelimesiyle nitelendirilirler.

static Değişkenler

- Statik değişkenler, nesneden bağımsız olduklarından nesnelerin değil, nesnelerin sınıfının parçasıdırlar.
- Bu yüzden, **static** olarak nitelendirilen değişkenlere **sınıf değişkenleri** (**class variables**) de denir.
- Statik değişkenlerin sadece bir kopyası vardır, o da sınıftadır.
- Sınıf değişkenlerine hem sınıf hem de nesneler üzerinden erişilebilinir.
  - Uygun olan sınıf üzerinden erişmektir; çünkü nesne üzerinden erişildiğinde, nesne değişkeni izlenimi vermektedir.

static Metotlar

- Sadece değişkenler değil, metotlar da static olabilirler,
  - Bunlara da statik metotlar denir.
- static metotlar da static değişkenler gibi sınıfın bir parçasıdırlar,
  - Hem sınıf hem de nesne üzerinde çağrılabilirler.
    - Uygun olan sınıf üzerinden erişmektir.

StaticDemo.java

Ne Zaman **static**? I

- Statik özellikler, sınıfın bir parçası olduklarından, çağrılmaları için bir nesneye ihtiyaç yoktur.
- Dolayısıyla eğer bir bilgi bir sınıftan oluşturulan nesnelerin durumunun bir parçası olup, değeri nesneden nesneye değişmiyorsa, bir başka değişle, değeri tüm nesneler için her zaman aynı olacaksa, bu değişken static yapılır.
- Benzer şekilde bir metot eğer bir sınıfın static olan değişkenlerini kullanıyor, nesnelerin değişkenlerini kullanmıyor ise o metot da static yapılır.

Ne Zaman **static**? II

- Zaten statik metotlar doğrudan ancak statik değişkenlere ulaşabilir, nesne değişkenlerine, nesnesiz olarak ulaşamaz.
- Çünkü statik metotlar bir nesneye ihtiyaç duymazlar ve bir nesne üzerinde çağrılsalar bile sınıf üzerinde çalışırlar ve sadece sınıf değişkenlerine ulaşırlar.
- Statik metotlar için this de yoktur.

Ne Zaman **static** Değil? I

- Statik kullanımı, tamamen marjinal ve sıra dışı bir durumdur.
  - Aslolan daima nesnedir, yani nesne değişkenleri ve metotlarıdır.
- Çünkü nesneler, problemimizi modellemeye yararlar.
- Statik değişken ve metotlar ise bu modelde çok özel durumlarda ortaya çıkarlar ve kullanımları ancak bu özel durumlara has olmalıdır.
- Nesne oluşturmanın gereksiz olduğu durumların çözümü static değildir.
  - Bu durumun çözümü gereksiz nesne oluşturmamaktır.

Ne Zaman **static** Değil? II

- Bir sınıftaki tüm değişkenleri ve dolayısıyla da metotları static yapmanın sebebi olsa olsa o sınıftan nesne oluşturmanın teorik ve pratik olarak anlamlı olmamasıdır.
  - java.lang.Math sınıfında var olan E ve PI alanları ile tüm metotlar statiktir, çünkü Math sınıfının nesnelerinin olması teorik açıdan muhaldir. Pratik açıdan da zaten sınıfı, muhtemel tek nesne olarak görülebilir.
  - Benzer şekilde java.lang.System sınıfı da statik alan ve metotlara sahiptir.
- İş alanını modellemeye bir katkısı olmayan utility sınıflarından, çoğu zaman bu sınıfların nesnelerini oluşturmadan, statik metotlarla, hizmet alırız.

Statik Bulaşıcıdır

- Aşırı miktarda static kullanımı, kodunuzu nesne merkezli olmaktan çıkaracaktır.
  - Çünkü ne kadar çok statik kullanırsanız, o kadar az nesne oluşturursunuz.
- Öte taraftan static kullanımı bulaşıcıdır, statik kullanımınız arttıkça daha çok statik kullanmanız gerekecektir.
- Bu da sizi nesne oluşturmadan nesne merkezli programlamaya götürebilir.
  - C görünümlü Java kodları!

YeniJavacınınSefaleti.java

Uygulama

- Bir sınıftan kaç tane nesne oluşturulduğunu nasıl bulursunuz?
- Bir sınıf yapın ve bu sınıfın herhangi bir kurucusunu çağırarak oluşturulan tüm nesneleri sayın.
- Bu sayıyı tutacak değişkenin nesne mi yoksa sınıf değişkeni mi olması gerekir?

main Metot

- **main** metot, statik metota çok güzel bir örnektir.
- main metot, JVM'e geçilen sınıfta bulunması gereklidir ve bu metot bir nesneye ihtiyaç olmadan çağrılabilmelidir.
- Bu yüzden main metot statiktir.

```
public static void main(String[] args)
```

static ve **final** Tanımlamalar

- Hem static hem de final olan tanımlamalar ile sınıflar üzerinden erişilen ve değeri değişmeyen sabiteler oluşturulabilir.
- Bu şekilde tanımlanan sabitelerin isminde *büyük harfler* kullanılır ve kelimeler arasına "_" getirilir.
- static ve final olan değişkenlerin bu şekilde yazılmaları için public olmalarına gerek yoktur.

```
static final int MAX_UNIT_COUNT = 100;
static final double STANDARD_LOAD_RATIO = 0.8;
```

Başlatma (Initialization)

İlk Değer Atama

- Java'da üye değişkenler (member variables) için ilk değer verme aşağıda belirtilen 5 yoldan herhangi birisiyle yapılabilir:
  - Tanımlama cümleleri (definition statements)
  - Kurucular (constructors)
  - Metot çağrıları
  - Nesne (ilk değer atama) başlatma blokları (initialization blocks)
  - Statik (ilk değer atama) başlatma blokları (static initialization blocks)

Tanımlama Cümleleri

- Statik olsun olmasın, üye değişkenlere ilk değer ataması, tanıtıldığı yerde yapılabilir.

```
int minAge = 18;
static double percentage = 5.8;
```

- Metot çağrısı ile de ilk değer ataması yapılabilir.
- Bu durumda nesne değişkeni için nesne metotu, sınıf değişkeni için ise sınıf metotu kullanılımalıdır.

```
boolean isOpen = getDoorStatus();
static double pi = getPi();
```

İleri Referans Problemi

- Üye değişkenleri tanımlarken ileri referans vermek bir derleyici hatasıdır:

```
int i = j;        // Compiler error!
A a = new A(j); // Compiler error!
int j = 8;
```

- Fakat üye değişkenleri tanımlarken metot çağrısı yapılabilir.
  - Bu durum bazı tutarsızlıklara sebep verebilir:

```
int i = getJ();  // i => 0;
int j = 8;

public int getJ(){
   return j;
}
```

InitializersDemo.java

Başlatma Blokları

- İlk değer atamaları **başlatma blokları** (**initializer blocks**) içinde yapılabilir.
- Bu bloklar genel olarak sınıfın başında tanımlanır.
- Nesne ve sınıf değişkenleri için, statik olmayan ve olan olmak üzere iki türü vardır.

```
public class Car{
   int speed;
   speed = 0;     // Compiler error!
   ...
   static int carCount;
   carCount = 1; // Compiler error!
   ...
}
```

Nesne Değişkenleri Başlatma Bloku

- Nesne değişkenleri tanıtıldıktan sonra **nesne başlatma bloklarında** (**instance initializer blocks**) ilk değerleri atanabilir.
- Nesne başlatma blokları, her nesne oluşturulmasında tekrar çalıştırılır.
  - Başlatma blokları ileriye referans veremez ve değer döndüremez.

```
public class MyClass{
    int j;
    boolean b;
    {
            j = 8;
            b = true;
    }
    ...
}
```

Nesne Değişkenleri Başlatma Bloku II

- Başlatma blokları neden kullanılır?
  - Karmaşık kod gerektiren ilk değer atamaları, böyle bir bloku gerek kılar.
  - Bazı ilk değerler için sıradışı durum fırlatabilen karmaşık hesaplamalar gerekli olabilir.
- Kurucu varken neden böyle bir ilk değer atama mekanizmasına ihtiyaç vardır?
  - Birden fazla kurucu olduğu durumda, ilk değer atama kodunun he kurucuya ayrı ayrı konması gerekir.
    - Bu da bakım problemine sebep olur.
- Karmaşık ilk değer kodunun bir yerde olup, hangi kurucunun çağrıldığından bağımsız olarak, her nesne için tekrar çalışmasını sağlamanın yolu, başlatma blokudur.

Statik Başlatma Blokları I

- **Statik başlatma blokları** (**static initializer blocks**) ise static değişkenlerde kullanılır.
- Blok başında static kelimesi kullanılır.
- Statik bloklar, sınıfın yüklenmesi sırasında bir defa çalıştırılır, bir daha çalıştırılmaz.

```
static double k;
static char c;

static{    // If static don't exists, it is
           // not initialized as static
   k = 5.0;
   c = 'c';
}
```

Statik Başlatma Blokları II

- Kurucular ve nesne başlatma blokları, nesne değişkenlerine ilk değer atamak için ideal yerlerdir.
- Fakat statik değişkenlerin ilk değer ataması kurucuya bırakılmamalıdır, çünkü hiç nesne oluşturulmayabilir.
- Dolayısıyla sınıf değişkenlerinin, nispeten karmaşık olan ya da sıradışı durum fırlatabilecek olan ilk değer atamaları, statik başlatma bloklarında yapılmalıdır.

```
static int[] ints = new int[10];
static {
    System.out.println("Initializing the
    array");
    for(int i = 0; i < ints.length; i++)
       ints[i] = i * 10;
```

InitializersBlock.java

Başlama Sırası I

- Bir sınıfta, pek çok sınıf ve nesne değişkeni, ilk değer atama blokları ve kurucular olduğunu göz önüne alındığında, bu değişkenlerin oluşturulmaları ve kurucuların çağrılması hangi sırada olur?

- InitializationOrder.java

Başlama Sırası II

- Bir sınıfa ilk defa ulaşıldığında önce o sınıfın *.class* dosyası JVM'e yüklenir.
  - Bir sınıfa ulaşmanın yolları ise şunlardır:
    - Statik bir değişkenine ulaşmak,
    - Statik bir metotunu çağırmak,
    - Nesnesini oluşturmak için kurucu çağrısı yapmak.
- Daha sonra sınıfın statik değişkenleri başlatılır.
  - Sınıfın bir nesnesi oluşturulmasa bile, sınıfa ilk kez ulaşıldığında statik değişkenleri yüklenir ve ilk değer ataması yapılır.
  - Değişkenlerin başlatılmasına, varsa statik başlatma blokları da dahildir.

Başlama Sırası III

- Sonra eğer sınıfın bir nesnesi oluşturuluyorsa, nesne değişkenleri de başlatılır.
  - Değişkenlerin başlatılmasına, varsa nesne başlatma blokları da dahildir.
- Daha sonra kurucu çağrısı yapılır.
- Her yeni nesne için bu işlemler, yani nesne değişkenlerinin başlatılması ve kurucu çağrısı tekrarlanır.
- Sınıf değişkenleri, ne kadar nesne oluşturulursa oluşturulsun, sadece ve sadece bir defa başlatılır.
- Fakat nesne oluşturulurken, her halukarda, sınıf değişkenleri nesne değişkenlerinden önce başlatılır.

Başlama Sırası IV

- Dolayısıyla başlama sırası
  - Sınıf değişkenleri (statik başlatma blokları dahil)
  - Nesne oluşturuluyorsa
    - Nesne değişkenleri (nesne başlatma blokları dahil)
    - Kurucu çağrısı

  şeklindedir.

- Birden fazla sınıf ve nesne değişkeni olduğu durumda başlatma sırası, fiziksel sırayla belirlenir, önce gelen önce başlatılır.

Uygulama

- Daha önce yaptığınız University uygulaması üzerinde başlama sırasını tahmin edin.
  - Kuruculara gerekeli print ifadelerini yazarak başlama sırasını gözlemleyin.

null

null I

- **null**, bir anahtar kelimedir, sadece referans değişkenlerine atanabilir ve referansın hiçbir nesneyi göstermediğini ifade eder.
  - **null**'ın tipi yoktur, her referans tipe atanabilir ya da çevrilebilir (cast).
  - Yani referans vardır ama bellekteki hiçbir nesneye işaret etmiyordur.
- Bu şekildeki referanslara "null pointer" denir.
- Bellekteki hiçbir nesneyi göstermeyen referanslar üzerinden herhangi bir erişim daima "*NullPointerException*"a sebep olur.

null II

- Bir referans, iki halde **null** olur:
  - Referans sadece tanımlanıp da herhangi bir nesne ataması yapılmadığında,
  - Referansa özel olarak **null** atandığında.
- Neden bir referansa **null** atanır?
  - Referans ile gösterdiği nesne arasındaki ilişki kesildiğinde ve nihayetinde bir nesneyi gösteren hiç bir referans kalmadığında, o nesne Çöp Toplayıcı (Garbage Collector) tarafından toplanıp işgal ettiği alan da belleğe geri

```
Car myCar;                  // null reference!
myCar.make = "Mercedes"      // NullPointerException!
myCar = new Car();         // Not a null reference anymore
myCar.make = "Mercedes" // Not a null pointer anymore
myCar = null;               // null reference!
```

null Geçmek ve Döndürmek

- Metotlara hiç bir zaman null referans geçmeyin ve metotlardan bazı istisnalar dışında hiç bir zaman null referans döndürmeyin.
- Çünkü null referans kullanımı devamlı null kontrolü gerektirir.
- Bazen metotların null referans döndğrme ihtimalle söz konusudur.
  - Bu durumda, metodun bunu API dokümanında (JavaDoc) belirtmesi gerekir.

```
Car car = getCar();     // Is it null?
if(car != null)
     car.accelerate(120);
```

NullExample.java

Garbage Collector

Garbage Collector - I

- Garbage Collector (Çöp Toplayıcı) ya da kısa haliyle GC, heapteki kullanılmayan nesnelerin toplanmasından sorumlu yapıdır.
- Kullanılmayan nesnelerin toplanması, bu nesnelerin kullandığı belleğin, yeni nesnelerin oluşturulabilmesi için kullanılabilir hale getirilmesi demektir.
- Aksi takdirde, bellekte kalmaya devam eden nesneler birikerek, bir süre sonra, bellek sızıntısı (memory leak), yeni nesnelerin oluşturulmasında yavaşlık hatta, fragmentasyondan dolayı, büyük nesnelerin oluşturulmasında zorluk gibi problemlere yol açacaklardır.

Garbage Collector - II

- Bellek sızıntılarının önüne geçmek amacıyla nesne oluşturmayı dikkatli yapmak ve gereksiz nesne oluşturmadan kaçınmak gereklidir.
- Oluşturulmuş nesnelere ihtiyaç kalmadığında, referanslarını null yaparak GC tarafından toplanmasına yardımcı olunabilir.

```
Car car = new Car();    // Is it null?
car.accelerate(120);
...
car = null; // Release the reference
       // so the object can be          // collected.
```

Özet

- Bu bölümde Java'da sınıf ve nesne yapılarına giriş yapıldı.
- Nesne ve sınıf değişkenleriyle kurucuların ve metotların nasıl oluşturulduğunu ve kullanıldığını ele alındı.
- **this** ve **static** anahtar kelimelerini incelendi.
- Sınıfların ve nesnelerin başlatılmasını incelendi.
- **null** ele alındı.

Ödevler

Ödevler I

- Bu bölümde örnek olarak gösterilen *University* uygulamasındaki sınıflara uygun kurucular ve sorumluluklarını yerine getiren metotlar yerleştirin.
  - Aynı uygulamaya *Classroom* isimli yeni bir sınıf ekleyin ve bu sını özelliklerini ve davranışlarını belirleyin.
  - Bu sınıfı *Course* sınıfıyla ilişkilendirin.
- Main metot içinde gerekli nesneleri oluşturup aralarındaki ilişkileri kurgulayın.

Ödevler II

- Bir kişinin kitap okumasını simule edecek şekilde Reader ve Book sınıflarını, uygun özellikler, kurucular ve metotlar ile oluşturun.
- Book'ta olabilecek özellikler
  - Author author (Sadece name özelliğine sahip olan)
  - String title
  - int noOfPages
  - String type
  - boolean isHardCover
  - int currentPage

Ödevler III

- Reader'ta olabilecek özellikler
  - String name
  - int age
  - char sex
  - Reader'a kitap okuması için read(Book book) şeklinde bir metot koy.
- İçinde bu sınıfların nesnelerinin oluşturulduğu ve main metoda sahip bir ReaderTest sınıfı oluşturun.
- Bir kaç Reader ve Book nesnesi ile okumayı simule edin.

Ödevler IV

- 2 boyutlu düzlemde bir noktayı temsil etmek üzere bir *Point* sınıfı oluşturun. Bu sınıfa, koordinat bilgileri ile uygun kurucu(lar) koyun. Ayrıca oluşturulan nokta nesnesinin orijine uzaklığını ve geçilen bir başka noktaya uzaklığını hesaplayan iki metot daha koyun.
  - Aynı sınıfın üzerine *clone* isimli bir metot koyun ve bu metotla aynı koordinatlara sahip bir başka Point nesnesi oluşturup döndürün.
  - Aynı sınıfın üzerine *move* isimli bir başka metot koyun ve bu metot ile nesneyi, "x" ya da "y" yönünde belirtilen miktar kadar hareket ettirip, aynı nesneyi geri döndürün.
- Daha sonra *PointTest* isimli bir başka sınıf oluşturun. Bu sınıfın main metotunda, 2 tane *Point* nesnesi oluşturup, üzerlerinde metot çağrıları yapın.

Ödevler V

- Daha sonra *Geometry* isimli bir başka sınıf oluşturun. Bu sınıfın üzerine, kendisine geçilen 2 *Point* nesnesi arasındaki uzaklığı hesaplayıp döndüren bir metot koyun.
  - Bu metodun statik olup olamayacağını tartışın.