



Metotlar: A Deeper Look

Java™ How to Program, 10/e



Amaçlar

- ▶ Java'da programlama Modülleri
- ▶ Static metotlar, statik fieldler ve Math statik sınıfı
- ▶ Çoklu parametre alan metod tanımlaması yapmak
- ▶ Argüman Promotion ve Casting
- ▶ Java API Paketleri
- ▶ Case Study: Güvenli Rastgele Sayı Üretme
- ▶ Case Study: Şans Oyunu (enum tipleri ile tanışma)
- ▶ Tanımlamaların kapsamı (Scope of declarations)
- ▶ Metod Aşırıyükleme (Method Overloading)



6.1 Tanıtım

- ▶ Geniş ölçekli programlar oluşturmanın en iyi yolu bu programı oldukça küçük parçalardan ya da modüllerden oluşturmaktadır.
 - **divide and conquer** (böl ve fethet)
- ▶ Bu bölümdeki konular
 - **static** metodlar
 - Method-call stack
 - Rastgele sayı üretme ile simülasyon teknikleri
 - Değişmeyen değerler tanımlama (constant)
 - Metod overloading.



6.2 Java'da Program Modülleri

- ▶ Java programları, yazdığınız yeni metodları ve sınıfları, **Java Application Programming Interface** ve diğer sınıf kütüphanelerinde bulunan önceden tanımlı metod ve sınıflarla birleştirir.
- ▶ İlgili sınıflar genellikle programlara import edilebilmeleri ve yeniden kullanılabilmeleri için paketler halinde grupperdir.



ÖNERİ

Java API'sı tarafından sağlanan zengin sınıf ve metod koleksiyonunu inceleyin (<http://docs.oracle.com/javase/9/docs/api/>). Bölüm 6.8, birkaç paketi gözden geçirmektedir.

Tekerleği yeniden icat etmeyin. Mümkün olduğunda, Java API sınıflarını ve yöntemlerini yeniden kullanın. Bu program geliştirme süresini azaltır ve programlama erro tanıtılmasını önler



6.2 Java'da Program Modülleri (Dvm.)

Sınıflar ve Metodlarla beraber Modülerlik

Sınıflar ve metodlar, bir programı kendi özel işini yapan birimler halinde ayırarak bir programı modüle etmenize yardımcı olur.

► Metod içerisindeki bildirimler

- Sadece bir kez yazılır.
- Diğer metodlardan saklanır.
- Programın farklı yerlerinde yeniden kullanılabilir.

Divide-and-conquer yaklaşımı

- Küçük ve basit parçalar ile programlar oluşturmak

► Software reusability

- Yeni programlar oluşturmak için mevcut sınıfları ve metodları yapı taşları olarak kullanın.
- Bir programın anlamlı metodlara bölünmesi, programın hata ayıklama ve bakım işlemlerini kolaylaştırır.



ÖNERİ

Yazılım yeniden kullanılabılırlığını artırmak için, her metot tek, iyi tanımlanmış bir görev gerçekleştirmeli ve metodun adı bu görevi etkin bir şekilde ifade etmelidir.

Bir görevi yerine getiren bir metodun, birçok görevi yerine getiren bir metoda göre test edilmesi ve hata ayıklamanması daha kolaydır.



6.2 Java'da Program Modülleri (Dvm.)

Metod Çağrıları Arasındaki Hiyerarşik İlişkiler

- Bir patron (çağıran) çalışanına (çağrılan metod) bir görevi gerçekleştirmesini ve görevi tamamladıktan sonra sonuçları geri bildirmesini ister.
 - Patron metodu, çalışan metodunun belirlenen görevleri nasıl gerçekleştirdiğini bilmez
 - Çalışan ayrıca patron tarafından tanınmayan diğer çalışan metodlarını da çağırabilir.
- Uygulama ayrıntılarının “gizlenmesi”, iyi yazılım mühendisliğini teşvik arttırmır.

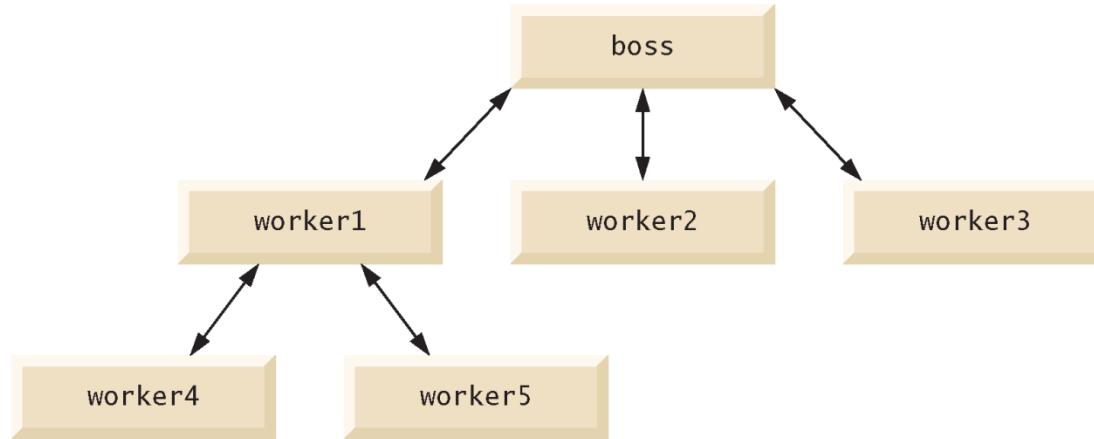


Fig. 6.1 | Hierarchical boss-method/worker-method relationship.



6.3 static Metotlar, static Fields ve Class Math

- ▶ Bazen bir metot, bir nesneye bağlı olmayan bir görevi gerçekleştirir.
- ▶ Bir bütün olarak beyan edildiği sınıf'a uygulanır
 - `static` metot ya da `class method` (`sınıf metodu`) olarak bilinir.
- ▶ It's common for classes to contain convenient `static` methods to perform common tasks.
- ▶ Çeşitli görevler gerçekleştirmek için sınıfların uygun statik metotlar içermesi yaygındır.
- ▶ Bir metodu statik yapmak için, `static` anahtar kelimesini, metodun beyanında dönüş tipinden önüne ekleyin.
- ▶ `static` bir metodu çağrırmak
 - *ClassName . methodName (arguments)*



6.3 static Metotlar, static Fields ve Class Math (Dvm)

Math Sınıfinın Metotları

- ▶ Math Sınıfı, ortak matematiksel hesaplamalar yapmanıza olanak tanıyan bir dizi statik yöntem sunar.
- ▶ Metot argümanları sabitler, değişkenler ya da ifadeler olabilir.



Bilgi

Math sınıfı, java.lang paketinin bir parçasıdır ve dolayısıyla derleyici tarafından import edilir; bu nedenle, Math sınıfının metodlarını kullanmak için sınıfı import etmeniz gereklidir.



Method	Description	Example
<code>abs(x)</code>	absolute value of x	<code>abs(23.7)</code> is 23.7 <code>abs(0.0)</code> is 0.0 <code>abs(-23.7)</code> is 23.7
<code>ceil(x)</code>	rounds x to the smallest integer not less than x	<code>ceil(9.2)</code> is 10.0 <code>ceil(-9.8)</code> is -9.0
<code>cos(x)</code>	trigonometric cosine of x (x in radians)	<code>cos(0.0)</code> is 1.0
<code>exp(x)</code>	exponential method e^x	<code>exp(1.0)</code> is 2.71828 <code>exp(2.0)</code> is 7.38906
<code>floor(x)</code>	rounds x to the largest integer not greater than x	<code>floor(9.2)</code> is 9.0 <code>floor(-9.8)</code> is -10.0
<code>log(x)</code>	natural logarithm of x (base e)	<code>log(Math.E)</code> is 1.0 <code>log(Math.E * Math.E)</code> is 2.0
<code>max(x, y)</code>	larger value of x and y	<code>max(2.3, 12.7)</code> is 12.7 <code>max(-2.3, -12.7)</code> is -2.3
<code>min(x, y)</code>	smaller value of x and y	<code>min(2.3, 12.7)</code> is 2.3 <code>min(-2.3, -12.7)</code> is -12.7

Fig. 6.2 | Math class methods. (Part I of 2.)



Method	Description	Example
<code>pow(x, y)</code>	x raised to the power y (i.e., x^y)	<code>pow(2.0, 7.0)</code> is 128.0 <code>pow(9.0, 0.5)</code> is 3.0
<code>sin(x)</code>	trigonometric sine of x (x in radians)	<code>sin(0.0)</code> is 0.0
<code>sqrt(x)</code>	square root of x	<code>sqrt(900.0)</code> is 30.0
<code>tan(x)</code>	trigonometric tangent of x (x in radians)	<code>tan(0.0)</code> is 0.0

Fig. 6.2 | Math class methods. (Part 2 of 2.)



6.3 static Metotlar, static Fields ve Class Math (Dvm)

- ▶ Bir sınıfın her bir nesnesinin, sınıfın her örnek değişkeninin kendi kopyasını oluşturduğunu hatırlayın.
- ▶ Bir sınıfın her bir nesnesinin kendi ayrı kopyasına (birazdan olarak göreceğiniz gibi) ihtiyacı olmayan değişkenler vardır.
- ▶ Bu değişkenler static olarak tanımlanır ve ayrıca **sınıf değişkenleri (class variables)** olarak bilinir.
- ▶ Static değişken içeren bir nesnenin nesneleri oluşturulduğunda, o sınıfın tüm nesneleri bu static değişkenlerin bir kopyasını paylaşır.
- ▶ Bir sınıfın statik değişkenleri ve örnek değişkenleri birlikte sınıfın field'leri (alanları) olarak bilinir.



6.3 static Metotlar, static Fields ve Class Math (Dvm)

Math Class static Constants PI and E

- ▶ Matematiksel sabitler için tanımlanmış Math fieldleri
 - `Math.PI` (3.141592653589793)
 - `Math.E` (2.718281828459045)
- ▶ Math sınıfının içerisinde `public`, `final` ve `static` erişim beliyleyicileri ile tanımlanmıştır .
 - `public` olmaları bu sabitleri kendi sınıflarınızda kullanmanıza olanak sağlamaktadır.
 - `final` ile tanımlanmış bir field sabittir değeri bir kere atandıktan sonra değişmez.



6.3 static Metotlar, static Fields ve Class Math (Dvm)

Neden main static olarak tanımlanmıştır?

- ▶ JVM belirttiğiniz sınıfın main yöntemini çağrırmaya çalışır; bu noktada sınıfın hiçbir nesnesi oluşturulmamıştır.
- ▶ main, static olması, JVM'nin, sınıfın bir örneğini oluşturmadan main'in çağrıması sağlanır.



6.4 Çok Sayıda Parametre Alan Metotlar

Tanımlama

- ▶ Çok parametre gönderimi virgül ile ayırılma ile gerçekleştirilebilmektedir.
- ▶ Metot çağrısında her bir parametre için (**formal parameter** olarak da adlandırılır) bir argüman bulunmalıdır.
- ▶ Her argüman, ilgili parametrenin tipi ile tutarlı olmalıdır.



```
1 // Fig. 6.3: MaximumFinder.java
2 // Programmer-declared method maximum with three double parameters.
3 import java.util.Scanner;
4
5 public class MaximumFinder
6 {
7     // obtain three floating-point values and locate the maximum value
8     public static void main(String[] args)
9     {
10         // create Scanner for input from command window
11         Scanner input = new Scanner(System.in);
12
13         // prompt for and input three floating-point values
14         System.out.print(
15             "Enter three floating-point values separated by spaces: ");
16         double number1 = input.nextDouble(); // read first double
17         double number2 = input.nextDouble(); // read second double
18         double number3 = input.nextDouble(); // read third double
19
20         // determine the maximum value
21         double result = maximum(number1, number2, number3);
22 }
```

Fig. 6.3 | Programmer-declared method `maximum` with three double parameters.
(Part 1 of 3.)



```
23     // display maximum value
24     System.out.println("Maximum is: " + result);
25 }
26
27 // returns the maximum of its three double parameters
28 public static double maximum(double x, double y, double z)
29 {
30     double maximumValue = x; // assume x is the largest to start
31
32     // determine whether y is greater than maximumValue
33     if (y > maximumValue)
34         maximumValue = y;
35
36     // determine whether z is greater than maximumValue
37     if (z > maximumValue)
38         maximumValue = z;
39
40     return maximumValue;
41 }
42 } // end class MaximumFinder
```

Fig. 6.3 | Programmer-declared method `maximum` with three `double` parameters.
(Part 2 of 3.)



```
Enter three floating-point values separated by spaces: 9.35 2.74 5.1  
Maximum is: 9.35
```

```
Enter three floating-point values separated by spaces: 5.8 12.45 8.32  
Maximum is: 12.45
```

```
Enter three floating-point values separated by spaces: 6.46 4.12 10.54  
Maximum is: 10.54
```

Fig. 6.3 | Programmer-declared method `maximum` with three `double` parameters.
(Part 3 of 3.)



Bilgi

Metotlar en fazla bir değer döndürebilir, ancak döndürülen değer birçok değer içeren bir nesne olabilir.

Değişkenler, sınıfın birden fazla metodunda kullanılması gerekiğinde veya programın değerlerini sınıfın farklı yöntemlerine yapılan çağrılar arasında kaydetmesi gerekiyorsa alanlar (field) olarak bildirilmelidir.

6.4 Çok Sayıda Parametre Alan Metotlar Tanımlama (Dvm.)

Math.max metodunu kullanarak üç sayının maksimumunu bulma

- ▶ `Math.max` metoduna iki çağrı ile yapılabilir:
 - `return Math.max(x, Math.max(y, z));`
- ▶ İlk çağrıının ilk argümanı `x` ve ikinci argümanı `Math.max(y, z)`'dır.
- ▶ Herhangi bir metod çağrılmadan önce, argümanları değerlerinin belirlenmesi için değerlendirilmelidir.
- ▶ Bir argüman bir metot çağrıası ise, dönüş değeri değerini belirlemek için bu metot çağrıısı yapılmalıdır.
- ▶ İlk çağrıının sonucu, ikinci argümanı olarak ikinci çağrıya geçer ve ikinci çağrı iki argümanın büyüğünü döndürür.



6.4 Çok Sayıda Parametre Alan Metotlar Tanımlama (Dvm.)

String Birleştirme

String concatenation

- + ya da += ile **String** nesneler birleştirilebiliyor.
- ▶ + operatörünün iki tarafı da **String** ise, + operatörü yeni bir **String** nesne yaratmaktadır.
 - Sağ işlenenin (operand) karakterleri sol işlenenin sonuna yerleştirilir.
 - *Java'daki her ilkel değer ve nesne bir String olarak temsil edilebilir.*
- ▶ İşlenenlerden bir tanesi **String** ise diğer **String**'e çevrilir. Ardından ikisi toplanır.
- ▶ Bir boolean bir **String** ile birleştirilirse, boolean "true" veya "false" **String**'ine dönüştürülür.
- ▶ Tüm nesnelerin, kendisinin bir **String** temsilini döndüren bir **toString** yöntemi vardır.



Bilgi

String literali satırlar arasında ayırmak bir syntax hatasıdır. Gerekirse, bir String'i birkaç küçük String'e bölebilir ve istenen Stringi oluşturmak için birleştirme kullanabilirsiniz.



Bilgi

String birleştirme için kullanılan + işlecinin toplama için kullanılan + işleviyle karıştırılması garip sonuçlara yol açabilir. Java, bir operatörün işlenenlerini soldan sağa doğru değerlendirir. Örneğin, y tamsayı değişkeni 5 değerine sahipse, "y + 2 =" + y + 2 ifadesi "y + 2 = 52" dizgesiyle sonuçlanır; y (5) "y + 2 =" dizesine eklenir, sonra 2 değeri "y + 2 = 5" yeni büyük dizeyle birleştirilir.

Doğrusu

"y + 2 =" + (y + 2) ifadesi üretimi "y + 2 = 7".



6.5 Metot Tanımlama ve Kullanma Üzerine Notlar

- ▶ Bir metodu çağrımanın üç yolu:
- ▶ Aynı sınıfın başka bir metodunu çağırmak için kendi başına bir metot adı kullanmak
 - Bir nesneye değişkeni kullanarak, ardından bir noktadan sonra(.) ve sınıfın metodunu çağırmak için metod adı
 - Bir sınıfın statik yöntemini çağırmak için sınıf adını ve noktayı(.) kullanma



6.5 Metot Tanımlama ve Kullanma Üzerine Notlar (Dvm.)

- ▶ Statik olmayan (Non-static) metotlara genel olarak örnek metotlar (**instance methods**) denir.
- ▶ **static** bir metot, aynı sınıfın diğer **static** metotlarını doğrudan çağırabilir ve aynı sınıftaki **static** değişkenleri doğrudan kullanabilir.
 - Sınıfın örnek değişkenlerine ve örnek metotlarına erişmek için, statik bir metot, sınıfın bir nesnesini kullanmalıdır.



6.5 Metot Tanımlama ve Kullanma Üzerine Notlar (Dvm.)

- ▶ Bir metodu çağrıran ifadeye kontrolü geri döndürmenin üç yolu:
- ▶ Program akışı metot sonu sağ pa ranteze ulaştığında
- ▶ Aşağıdaki ifade çalıştırıldığında
- ▶ **return;**
 - Metodu aşağıdaki gibi bir ifade ile bir sonuç döndüğünde
 - **return expression;**



Bilgi

Sınıf tanımının dışarısında veya başka bir metodun gövdesi içinde bir metodun oluşturulması bir syntax hatasıdır.

Bir parametrenin metot gövdesinde yerel değişken olarak bildirilmesi bir derleme hatasıdır.

Değer döndürmesi gereken bir metotta bir değer döndürmeyi unutmak bir derleme hatasıdır. Boşluk dışında bir dönüş türü belirtilirse, metot, metodun dönüş türüyle tutarlı bir değer döndüren bir dönüş ifadesi içermelidir. Dönüş türü void olarak bildirilen bir metottan bir değer döndürmek bir derleme hatasıdır.

6.6 Method-Call Stack ve Stack Frames

- ▶ **Stack veri yapısı**
 - Tabak yığınına benzemektedir
 - Üstteki yığına bir tabak yerleştirilir (tabağı stack'in üzerine **pushing** olarak adlandırılır).
 - Bir tabak yığından en üst kısmından tabak alınır (tabağı yığından **pop** etmek olarak adlandırılır).
- ▶ **Last-in, first-out (LIFO) data structures**
 - Yığının üzerine push edilen son öğe, yığından pop edilen ilk öğedir.



6.6 Method-Call Stack and Activation Records (Dvm.)

- ▶ Bir program bir metodu çağrıduğunda, çağrılan metod, çağrıran yere nasıl döneceğini bilmelidir.
 - Çağırılan metodun adresi, **method-call stack** üzerine push edilir.
- ▶ Bir dizi metod çağrısı gerçekleşirse, birbirini izleyen geri dönüş adresleri, son giriş, ilk çıkış sırasına göre yığının üzerine push edilir.
- ▶ Method call stack aynı zamanda metodların yerel değişkenlerinin (parametreler de dahil olmak üzere) belleğini de içermektedir.
 - **stack frame** (veya **activation record**) olarak bilinen bu alan, method call stack'inin bir kısmı olarak depolanır.



6.6 Method-Call Stack and Activation Records (Dvm.)

- ▶ Bir metot çağrısı yapıldığında, bu metot çağrısı için stack frame method call stack üzerine push edilir.
- ▶ Metot çağrıldığı yere geri döndüğünde, stack frame yığından çıkarılır (pop edilir) ve bu yerel değişkenler artık program tarafından bilinmez.
- ▶ Stack framelerinin program yürütme yığınında saklanabileceğinden daha fazla metot çağrısı gerçekleşirse, **stack overflow** olarak bilinen bir hata oluşur.



6.7 Argument Promotion ve Casting

► Argument promotion

- Mümkünse argümanın değerini metodun umduğu tipe dönüştürmeye denilmektedir.
- Java'nın **promotion kurallarına** uyulmadıysa derleme hataları oluşabilir.
- Promotion kuralları
 - Hangi dönüşümlerein izin verildiğini kapsar.



Type	Valid promotions
double	None
float	double
long	float or double
int	long, float or double
char	int, long, float or double
short	int, long, float or double (but not char)
byte	short, int, long, float or double (but not char)
boolean	None (boolean values are not considered to be numbers in Java)

Fig. 6.4 | Promotions allowed for primitive types.



6.7 Argument Promotion ve Casting(Dvm.)

- ▶ Şekil 6.4'teki tabloda bulunan değerlerin daha düşük tiplere dönüştürülmesi, alt tipin daha yüksek tipin değerini gösterememesi durumunda farklı değerlerle sonuçlanacaktır.
- ▶ Dönüşüm nedeniyle bilginin kaybedileceği durumlarda, Java derleyici, dönüştürmenin gerçekleşmesini açıkça zorlamak için bir cast operatörünü kullanmanızı gerektirir; aksi halde derleme hatası oluşur.



6.8 Java API Paketleri

- ▶ Java, ilgili sınıfların kategorileri olarak gruplandırılmış paket adı verilen birçok tanımlanmış sınıf içerir.
- ▶ Java'nın büyük bir gücü Java API'nın binlerce sınıfından oluşmasıdır.
- ▶ Bu derste kullanacağımız bazı temel Java API paketleri Şekil 6.5'de açıklanmıştır.
- ▶ Java Paketlerine aşağıdan ulaşabilirsiniz:
 - <http://docs.oracle.com/javase/9/docs/api/overview-summary.html>



Package	Description
<code>java.awt.event</code>	The Java Abstract Window Toolkit Event Package contains classes and interfaces that enable event handling for GUI components in both the <code>java.awt</code> and <code>javax.swing</code> packages. (See Chapter 12, GUI Components: Part 1, and Chapter 22, GUI Components: Part 2.)
<code>java.awt.geom</code>	The Java 2D Shapes Package contains classes and interfaces for working with Java's advanced two-dimensional graphics capabilities. (See Chapter 13, Graphics and Java 2D.)
<code>java.io</code>	The Java Input/Output Package contains classes and interfaces that enable programs to input and output data. (See Chapter 15, Files, Streams and Object Serialization.)
<code>java.lang</code>	The Java Language Package contains classes and interfaces (discussed throughout the book) that are required by many Java programs. This package is imported by the compiler into all programs.
<code>java.net</code>	The Java Networking Package contains classes and interfaces that enable programs to communicate via computer networks like the Internet. (See online Chapter 28, Networking.)
<code>java.security</code>	The Java Security Package contains classes and interfaces for enhancing application security.

Fig. 6.5 | Java API packages (a subset). (Part 1 of 4.)



Package

Description

`java.sql`

The **JDBC Package** contains classes and interfaces for working with databases. (See Chapter 24, Accessing Databases with JDBC.)

`java.util`

The **Java Utilities Package** contains utility classes and interfaces that enable storing and processing of large amounts of data. Many of these classes and interfaces have been updated to support Java SE 8's new lambda capabilities. (See Chapter 16, Generic Collections.)

`java.util.concurrent`

The **Java Concurrency Package** contains utility classes and interfaces for implementing programs that can perform multiple tasks in parallel. (See Chapter 23, Concurrency.)

`javax.swing`

The **Java Swing GUI Components Package** contains classes and interfaces for Java's Swing GUI components that provide support for portable GUIs. This package still uses some elements of the older `java.awt` package. (See Chapter 12, GUI Components: Part 1, and Chapter 22, GUI Components: Part 2.)

`javax.swing.event`

The **Java Swing Event Package** contains classes and interfaces that enable event handling (e.g., responding to button clicks) for GUI components in package `javax.swing`. (See Chapter 12, GUI Components: Part 1, and Chapter 22, GUI Components: Part 2.)

Fig. 6.5 | Java API packages (a subset). (Part 2 of 4.)



Package	Description
<code>javax.xml.ws</code>	The JAX-WS Package contains classes and interfaces for working with web services in Java. (See online Chapter 32, REST-Based Web Services.)
<code>javafx</code> packages	JavaFX is the preferred GUI technology for the future. We discuss these packages in Chapter 25, JavaFX GUI: Part 1 and in the online JavaFX GUI and multimedia chapters.

Fig. 6.5 | Java API packages (a subset). (Part 3 of 4.)



Package	Description
<i>Some Java SE 8 Packages Used in This Book</i>	
<code>java.time</code>	The new Java SE 8 Date/Time API Package contains classes and interfaces for working with dates and times. These features are designed to replace the older date and time capabilities of package <code>java.util</code> . (See Chapter 23, Concurrency.)
<code>java.util.function</code> and <code>java.util.stream</code>	These packages contain classes and interfaces for working with Java SE 8's functional programming capabilities. (See Chapter 17, Java SE 8 Lambdas and Streams.)

Fig. 6.5 | Java API packages (a subset). (Part 4 of 4.)



6.9 Case Study: Secure Random-Number Generation

- ▶ Simulation ve game playing
 - element of chance
 - Class `SecureRandom` (package `java.security`)
- ▶ Such objects can produce random `boolean`, `byte`, `float`, `double`, `int`, `long` and Gaussian values
- ▶ `SecureRandom` nesneleri tahmin edilemeyen **nondeterministic random sayılar** üretebilmektedir.
SecureRandom Sınıfı için dökümentasyon
 - <https://docs.oracle.com/javase/9/docs/api/java/security/SecureRandom.html>



6.9 Case Study: Random-Number Generation (Dvm.)

- ▶ SecureRandom sınıfının `nextInt` metodu tarafından üretilen değerlerin aralığı, genellikle belirli bir Java uygulamasında gerekli olan değerler aralığından farklıdır.
- ▶ Bir int argümanı alan `SecureRandom nextInt` yöntemi, argüman değerine 0'dan başlayarak, ancak buna dahil olmayan bir değer döndürür.



```
1 // Fig. 6.6: RandomIntegers.java
2 // Shifted and scaled random integers.
3 import java.security.SecureRandom; // program uses class SecureRandom
4
5 public class RandomIntegers
6 {
7     public static void main(String[] args)
8     {
9         // randomNumbers object will produce secure random numbers
10        SecureRandom randomNumbers = new SecureRandom();
11
12        // Loop 20 times
13        for (int counter = 1; counter <= 20; counter++)
14        {
15            // pick random integer from 1 to 6
16            int face = 1 + randomNumbers.nextInt(6);
17
18            System.out.printf("%d ", face); // display generated value
19
20            // if counter is divisible by 5, start a new line of output
21            if (counter % 5 == 0)
22                System.out.println();
23        }
24    }
25 } // end class RandomIntegers
```

Fig. 6.6 | Shifted and scaled random integers. (Part I of 2.)



1	5	3	6	2
5	2	6	5	2
4	4	4	2	6
3	1	6	2	2

6	5	4	2	6
1	2	5	1	3
6	3	2	2	1
6	4	2	6	4

Fig. 6.6 | Shifted and scaled random integers. (Part 2 of 2.)



6.9 Case Study: Random-Number Generation (Cont.)

- ▶ Fig 6.7: Rolling a Six-Sided Die 6,000,000 Times



```
1 // Fig. 6.7: RollDie.java
2 // Roll a six-sided die 6,000,000 times.
3 import java.security.SecureRandom;
4
5 public class RollDie
6 {
7     public static void main(String[] args)
8     {
9         // randomNumbers object will produce secure random numbers
10        SecureRandom randomNumbers = new SecureRandom();
11
12        int frequency1 = 0; // count of 1s rolled
13        int frequency2 = 0; // count of 2s rolled
14        int frequency3 = 0; // count of 3s rolled
15        int frequency4 = 0; // count of 4s rolled
16        int frequency5 = 0; // count of 5s rolled
17        int frequency6 = 0; // count of 6s rolled
18
19        // tally counts for 6,000,000 rolls of a die
20        for (int roll = 1; roll <= 6000000; roll++)
21        {
22            int face = 1 + randomNumbers.nextInt(6); // number from 1 to 6
23        }
```

Fig. 6.7 | Roll a six-sided die 6,000,000 times. (Part I of 3.)



```
24     // use face value 1-6 to determine which counter to increment
25     switch (face)
26     {
27         case 1:
28             ++frequency1; // increment the 1s counter
29             break;
30         case 2:
31             ++frequency2; // increment the 2s counter
32             break;
33         case 3:
34             ++frequency3; // increment the 3s counter
35             break;
36         case 4:
37             ++frequency4; // increment the 4s counter
38             break;
39         case 5:
40             ++frequency5; // increment the 5s counter
41             break;
42         case 6:
43             ++frequency6; // increment the 6s counter
44             break;
45     }
46 }
47 }
```

Fig. 6.7 | Roll a six-sided die 6,000,000 times. (Part 2 of 3.)



```
48     System.out.println("Face\tFrequency"); // output headers
49     System.out.printf("1\t%d\n2\t%d\n3\t%d\n4\t%d\n5\t%d\n6\t%d\n",
50                         frequency1, frequency2, frequency3, frequency4,
51                         frequency5, frequency6);
52 }
53 } // end class RollDie
```

Face	Frequency
1	999501
2	1000412
3	998262
4	1000820
5	1002245
6	998760

Face	Frequency
1	999647
2	999557
3	999571
4	1000376
5	1000701
6	1000148

Fig. 6.7 | Roll a six-sided die 6,000,000 times. (Part 3 of 3.)



6.10 Case Study: A Game of Chance; enum Tipi İle Tanışma

- ▶ Craps Oyunu:
 - İki tane zar atıyorsunuz. Her zarın 6 tane kenarı var. İki zar altıldıkten sonra üst tarafta kalan sayılar toplanıyor.
 - İlk atışta eğer toplam 7 ya da 11 ise kazanıyorsunuz. Eğer ilk atışta toplam 2,3 ya da 12 ise kaybediyorsunuz (craps). Eğer toplam 4,5,6,8,9 veya 10 ise bu toplam sizin puanınız oluyor. Kazanmak için aynı puanı bir daha atmaya çalışıyorsunuz. Eğer yapamadan yine toplamda 7 atarsanız kaybediyorsunuz.



```
1 // Fig. 6.8: Craps.java
2 // Craps class simulates the dice game craps.
3 import java.security.SecureRandom;
4
5 public class Craps
{
6     // create secure random number generator for use in method rollDice
7     private static final SecureRandom randomNumbers = new SecureRandom();
8
9
10    // enum type with constants that represent the game status
11    private enum Status { CONTINUE, WON, LOST };
12
13    // constants that represent common rolls of the dice
14    private static final int SNAKE_EYES = 2;
15    private static final int TREY = 3;
16    private static final int SEVEN = 7;
17    private static final int YO_LEVEN = 11;
18    private static final int BOX_CARS = 12;
19
```

Fig. 6.8 | Craps class simulates the dice game craps. (Part 1 of 5.)



```
20 // plays one game of craps
21 public static void main(String[] args)
22 {
23     int myPoint = 0; // point if no win or loss on first roll
24     Status gameStatus; // can contain CONTINUE, WON or LOST
25
26     int sumOfDice = rollDice(); // first roll of the dice
27
28     // determine game status and point based on first roll
29     switch (sumOfDice)
30     {
31         case SEVEN: // win with 7 on first roll
32         case YO_LEVEN: // win with 11 on first roll
33             gameStatus = Status.WON;
34             break;
35         case SNAKE_EYES: // lose with 2 on first roll
36         case TREY: // lose with 3 on first roll
37         case BOX_CARS: // lose with 12 on first roll
38             gameStatus = Status.LOST;
39             break;

```

Fig. 6.8 | Craps class simulates the dice game craps. (Part 2 of 5.)



```
40     default: // did not win or lose, so remember point
41         gameStatus = Status.CONTINUE; // game is not over
42         myPoint = sumOfDice; // remember the point
43         System.out.printf("Point is %d%n", myPoint);
44         break;
45     }
46
47     // while game is not complete
48     while (gameStatus == Status.CONTINUE) // not WON or LOST
49     {
50         sumOfDice = rollDice(); // roll dice again
51
52         // determine game status
53         if (sumOfDice == myPoint) // win by making point
54             gameStatus = Status.WON;
55         else
56             if (sumOfDice == SEVEN) // lose by rolling 7 before point
57                 gameStatus = Status.LOST;
58     }
59 }
```

Fig. 6.8 | Craps class simulates the dice game craps. (Part 3 of 5.)



```
60     // display won or lost message
61     if (gameStatus == Status.WON)
62         System.out.println("Player wins");
63     else
64         System.out.println("Player loses");
65 }
66
67 // roll dice, calculate sum and display results
68 public static int rollDice()
69 {
70     // pick random die values
71     int die1 = 1 + randomNumbers.nextInt(6); // first die roll
72     int die2 = 1 + randomNumbers.nextInt(6); // second die roll
73
74     int sum = die1 + die2; // sum of die values
75
76     // display results of this roll
77     System.out.printf("Player rolled %d + %d = %d%n",
78                     die1, die2, sum);
79
80     return sum;
81 }
82 } // end class Craps
```

Fig. 6.8 | Craps class simulates the dice game craps. (Part 4 of 5.)



Player rolled 5 + 6 = 11
Player wins

Player rolled 5 + 4 = 9
Point is 9
Player rolled 4 + 2 = 6
Player rolled 3 + 6 = 9
Player wins

Player rolled 1 + 2 = 3
Player loses

Player rolled 2 + 6 = 8
Point is 8
Player rolled 5 + 1 = 6
Player rolled 2 + 1 = 3
Player rolled 1 + 6 = 7
Player loses

Fig. 6.8 | Craps class simulates the dice game craps. (Part 5 of 5.)



6.10 Case Study: A Game of Chance; enum Tipi İle Tanışma (Dvm.)

- ▶ Notes:
 - **myPoint** 0 olarak başlatılmıştır .
 - is initialized to 0 to ensure that the application will compile.
 - **myPoint e** If ilk değerini atamazsanız derleyici hata verir.
 - **gameStatus** is **switch**'in her **case** ifadesinde bir değer aldığı için kullanılmaya başlanmadan ilk değeri aldığı garantilenmiştir.



6.10 Case Study: A Game of Chance; enum Tipi İle Tanışma (Dvm.)

enum tipinde Status

- ▶ **enum** tipi sabitleri temsil eder .
- ▶ **enum** sınıflar gibi küme parantezleri ile tanımlanır.
- ▶ Bu parantezler içerisinde virgülle ayrılarak herbiri farklı değeri temsil eden sabitler (**enum constants**) tanımlanmaktadır.
- ▶ Bu sabitler farklı kelimelerden oluşmalıdır (unique olmalıdır).
- ▶ enum tipindeki değişkenler bu tanımlanmış sabitlerden bir tanesine atanabilmektedir.



Öneri

Enum constantları büyük harf ile tanımlayın ki diğer değişkenlerden farklı olduğu anlaşılsın.

Status.WON, Status.LOST gibi enum değerlerini kullanmak bunun yerine 0 veya 1 gibi değerleri kullanmaya göre kodun okunurluğunu ve anlaşılırlığını artırmaktadır.



6.10 Case Study: A Game of Chance; enum Tipi İle Tanışma (Dvm.)

Neden bazı sabitler enum Constantlar olarak
tanımlanmamıştır?

*Java int bir değerin enum sabiti ile
karşılaştırılmasına izin vermemektedir.*

► Bir int değerinin int tipine dönüştürülmesi switch case ile yapılabilir ancak bu da zahmetlidir. Ayrıca kodun okunurluğunu da azaltmaktadır.



6.11 Tanımlamaların Kapsamı (Scope of Declarations)

- ▶ Bir değişkenin kapsamı (scope) program içerisinde erişilebildiği kısmı olarak bilinir.
 - Bu değişken programın bu kısmı içerisinde kapsamın içinde olarak bahsedilir.



6.11 Tanımlamaların Kapsamı (Scope of Declarations) (Dvm.)

- ▶ Temel scope kuralları:
 - Parametrenin scope'u ait olduğu metodun içerisindeindedir.
 - Bir yerel değişkenin scope'u tanımlamanın başladığı yerden tanımlanmanın ait olduğu bloğun sonuna kadardır.
 - Örn. **For** ifadesinin headerında tanımlanan değişken sadece ilgili **for** ifadesi içerisindeinden erişilebilmektedir
 - Bir metodun ya da bir field'in kapsamı tanımlı olduğu sınıfın içerisindeidir.
 - Herhangi bir blok değişken tanımlaması içerebilmektedir.
 - Eğer yerel bir değişkenin adı sınıfın bir fieldi ile aynı isimde ise sınıfın fieldi yerel değişkenin kapsamı geçene kadar saklanmış durumda olmaktadır. Buna **shadowing** denilmektedir.



```
1 // Fig. 6.9: Scope.java
2 // Scope class demonstrates field and local variable scopes.
3
4 public class Scope
5 {
6     // field that is accessible to all methods of this class
7     private static int x = 1;
8
9     // method main creates and initializes local variable x
10    // and calls methods useLocalVariable and useField
11    public static void main(String[] args)
12    {
13        int x = 5; // method's local variable x shadows field x
14
15        System.out.printf("local x in main is %d%n", x);
16
17        useLocalVariable(); // useLocalVariable has local x
18        useField(); // useField uses class Scope's field x
19        useLocalVariable(); // useLocalVariable reinitializes local x
20        useField(); // class Scope's field x retains its value
21
22        System.out.printf("%nlocal x in main is %d%n", x);
23    }
24}
```

Fig. 6.9 | Scope class demonstrates field and local-variable scopes. (Part 1 of 3.)



```
25 // create and initialize local variable x during each call
26 public static void useLocalVariable()
27 {
28     int x = 25; // initialized each time useLocalVariable is called
29
30     System.out.printf(
31         "%nlocal x on entering method useLocalVariable is %d%n", x);
32     ++x; // modifies this method's local variable x
33     System.out.printf(
34         "local x before exiting method useLocalVariable is %d%n", x);
35 }
36
37 // modify class Scope's field x during each call
38 public static void useField()
39 {
40     System.out.printf(
41         "%nfield x on entering method useField is %d%n", x);
42     x *= 10; // modifies class Scope's field x
43     System.out.printf(
44         "field x before exiting method useField is %d%n", x);
45 }
46 } // end class Scope
```

Fig. 6.9 | Scope class demonstrates field and local-variable scopes. (Part 2 of 3.)



```
local x in main is 5
```

```
local x on entering method useLocalVariable is 25  
local x before exiting method useLocalVariable is 26
```

```
field x on entering method useField is 1  
field x before exiting method useField is 10
```

```
local x on entering method useLocalVariable is 25  
local x before exiting method useLocalVariable is 26
```

```
field x on entering method useField is 10  
field x before exiting method useField is 100
```

```
local x in main is 5
```

Fig. 6.9 | Scope class demonstrates field and local-variable scopes. (Part 3 of 3.)



6.12 Method Overloading (Metot Aşırı Yükleme)

- ▶ Method overloading (Metot Aşırı Yüklenmesi)
 - Aynı sınıftha aynı isim ile tanımlanan metodlarda geçerlidir.
 - Ancak bu metodlar farklı parametrelere sahip olmalıdır.
- ▶ Derleyici, parametre sayısı, tipi ve yerlerine göre uygun metodu çağrılmaktadır.
- ▶ Benzer işleri gerçekleştiren aynı isimde farklı tipte parametre alan metodlar için kullanılmaktadır.



```
1 // Fig. 6.10: MethodOverload.java
2 // Overloaded method declarations.
3
4 public class MethodOverload
5 {
6     // test overloaded square methods
7     public static void main(String[] args)
8     {
9         System.out.printf("Square of integer 7 is %d%n", square(7));
10        System.out.printf("Square of double 7.5 is %f%n", square(7.5));
11    }
12
13    // square method with int argument
14    public static int square(int intValue)
15    {
16        System.out.printf("%nCalled square with int argument: %d%n",
17                          intValue);
18        return intValue * intValue;
19    }
20}
```

Fig. 6.10 | Overloaded method declarations. (Part I of 2.)



```
21 // square method with double argument
22 public static double square(double doubleValue)
23 {
24     System.out.printf("%nCalled square with double argument: %f%n",
25                         doubleValue);
26     return doubleValue * doubleValue;
27 }
28 } // end class MethodOverload
```

```
Called square with int argument: 7
Square of integer 7 is 49
```

```
Called square with double argument: 7.500000
Square of double 7.5 is 56.250000
```

Fig. 6.10 | Overloaded method declarations. (Part 2 of 2.)



6.12 Method Overloading (cont.)

Overloaded Metotları Ayırt Edebilme

- ▶ Derleyici overloaded metotları metotlarının imzalarından (**signatures**) ayırt edebilir. **signatures**—metot adı, parametre sayısı, parametre tipleri ve parametre sıraları.
- ▶ Metotlar geri dönüş tüplerinden ayırt edilemezler.
- ▶ Ama overloaded metotların farklı dönüş tipleri olabilir.



Common Programming Error 6.8

Declaring overloaded methods with identical parameter lists is a compilation error regardless of whether the return types are different.