

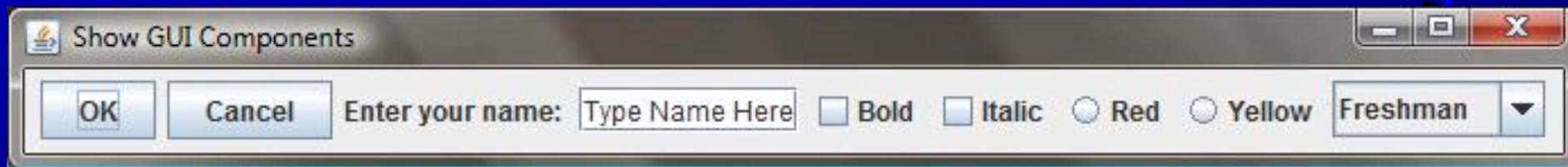
Bölüm 8

Nesneler ve Sınıflar



Motivasyon

Önceki bölümleri öğrendikten sonra, seçimleri, döngüleri, metotları ve dizileri kullanarak birçok programlama problemini çözme yeteneğine sahipsiniz. Bununla birlikte, bu Java özellikleri, grafiksel kullanıcı arayüzleri ve büyük ölçekli yazılım sistemleri geliştirmek için yeterli değildir. Aşağıda gösterildiği gibi bir grafik kullanıcı arayüzü geliştirmek istediğinizi varsayalım. Nasıl programlarsınız?



Nesne Yönelimli Programlama Kavramları

Nesneye yönelimli programlama (NYP), nesneleri kullanarak programlamayı içerir. Bir nesne, gerçek dünyada açıkça tanımlanabilecek olan bir varlığı temsil eder. Örneğin, bir öğrenci, bir masa, bir daire, bir düğme ve hatta bir borç bile nesne olarak görülebilir.

Bir nesnenin benzersiz bir kimliği, durumu ve davranışları vardır. Bir nesnenin durumu, geçerli değerleri ile birlikte bir dizi veri alanından (özellikler olarak da bilinir) oluşur. Bir nesnenin davranışı bir dizi yöntemle tanımlanır.

Nesneler (Objects)

Class Name: Circle

Data Fields:
radius is _____

Methods:
getArea

← A class template

Circle Object 1

Data Fields:
radius is 10

Circle Object 2

Data Fields:
radius is 25

Circle Object 3

Data Fields:
radius is 125

← Three objects of the Circle class

Bir nesnenin hem durumu hem de davranışı vardır. Durum nesneyi, davranış ise nesnenin ne yaptığını tanımlar.

Sınıflar (Classes)

Sınıflar, aynı türden nesneleri tanımlayan yapılardır.

Bir Java sınıfı veri alanlarını tanımlamak için değişkenleri ve davranışları tanımlamak için metotları kullanır.

Ek olarak, bir sınıf, sınıftan nesneler oluşturmak için çağrılan, yapıcılar olarak bilinen özel bir metot türü sağlar.



Similar (Classes)

```
class Circle {  
    /** The radius of this circle */  
    double radius = 1.0;  
  
    /** Construct a circle object */  
    Circle() {  
    }  
  
    /** Construct a circle object */  
    Circle(double newRadius) {  
        radius = newRadius;  
    }  
  
    /** Return the area of this circle */  
    double getArea() {  
        return radius * radius * 3.14159;  
    }  
}
```

Data field

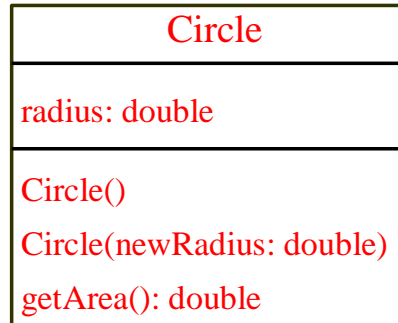
Constructors

Method



UML Sınıf Diyagramı

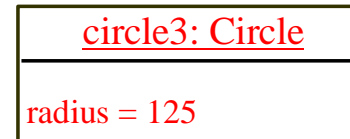
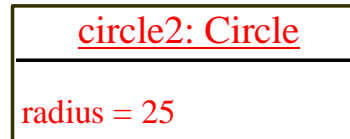
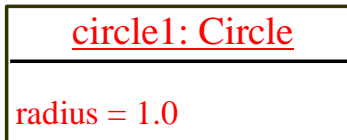
UML Class Diagram



← Class name

← Data fields

← Constructors and methods



← UML notation for objects



Örnek: Sınıfları Tanımlamak ve Nesneleri Oluşturmak

- Amaç: Nesne oluşturmayı, verilere erişmeyi ve metotları kullanmayı gösterme.

TestCircle1

Run



Örnek: Sınıfları Tanımlamak ve Nesneleri Oluşturmak

- Amaç: Nesne oluşturmayı, verilere erişmeyi ve metotları kullanmayı gösterme.

TV

TestTV

Run



Yapıcılar (Constructors)

Yapıcılar, nesneler oluşturmak için çağrılan özel bir metot türüdür.

```
Circle() {  
}
```

```
Circle(double newRadius) {  
    radius = newRadius;  
}
```



Yapıcılar (Constructors)

Parametresiz bir yapıcı, bağımsız değişken yapıcı olarak adlandırılır.

- Yapıcılar, sınıfın kendisi ile aynı isme sahip olmalıdırlar.
- Yapıcılar geri dönüş türüne sahip değildir;
- Bir nesne oluşturulduğunda, yapıcılar (new) işleci kullanılarak çağrılır.

Yapıcılar nesnelerin başlatılması rolünü oynarlar.



Yapıcıları Kullanarak Nesne Oluşturmak

```
new ClassName();
```

Örnek:

```
new Circle();
```

```
new Circle(5.0);
```



Varsayılan Yapıcı (Constructor)

Yapıcılar olmadan bir sınıf oluşturulabilir. Bu durumda, boş gövdeli bir argüman olmayan kurucu sınıfta dolaylı olarak bildirilir.

Varsayılan kurucu olarak adlandırılan bu kurucu, yalnızca sınıfta açıkça bir kurucu bildirilmezse otomatik olarak sağlanır.



Nesne Referans Değişkenlerini Bildirmek

Bir nesneye referans vermek için nesneyi bir referans değişkenine atamak gerekir.

Bir referans değişkeni bildirmek için aşağıdaki sözdizimi (syntax) kullanılmaktadır:

```
ClassName objectRefVar;
```

Örnek:

```
Circle myCircle;
```




Tek Adımda Nesneleri Bildirmek / Oluşturmak

```
ClassName objectRefVar = new ClassName();
```

Örnek:

Assign object reference Create an object

```
Circle myCircle = new Circle();
```



Nesnelere Erişim

- Nesnenin verilerine referans verme:

`objectRefVar.data`

e.g., `myCircle.radius`

- Nesnenin metodunu çağırma:

`objectRefVar.methodName (arguments)`

e.g., `myCircle.getArea()`



Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

```
SCircle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle' in bildirilmesi

myCircle

no value



Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle no value

<u>: Circle</u>
radius: 5.0

Circle' ın oluşturulması



Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

Nesne referansının
myCircle' a atanması

myCircle

reference value

: Circle

radius: 5.0



Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value

<u>: Circle</u>
radius: 5.0

yourCircle no value

yourCircle' in bildirilmesi

Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value

<u>: Circle</u>
radius: 5.0

yourCircle no value

Yeni bir Circle
object' in
oluşturulması

<u>: Circle</u>
radius: 0.0

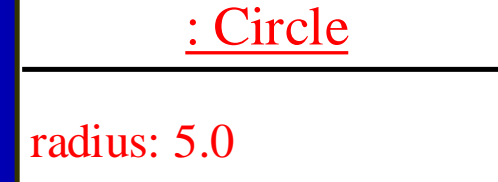
Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

```
Circle yourCircle = new Circle();
```

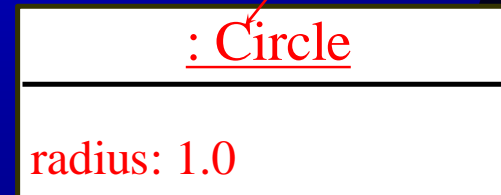
```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle reference value

Object referansının
yourCircle'a atanması



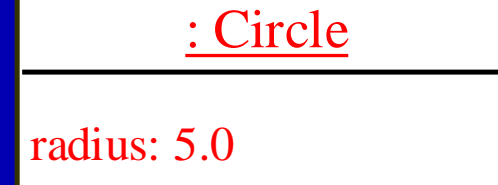
Kodun İzlenmesi (Trace Code)

```
Circle myCircle = new Circle(5.0);
```

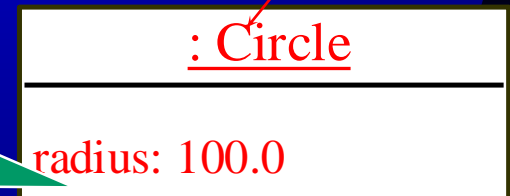
```
Circle yourCircle = new Circle();
```

```
yourCircle.radius = 100;
```

myCircle reference value



yourCircle reference value



yourCircle' ın yarıçap
değerinin değiştirilmesi

Dikkat ()Caution

Aşağıdaki yapıyı hatırlayalım;

Math.methodName(arguments) (e.g., Math.pow(3, 2.5))

Math sınıfındaki bir metodu çağırmak için. Circle1.getArea () işlevini kullanarak getArea () işlevini çağırabilir misiniz? Cevap hayır. Bu bölümden önce kullanılan tüm yöntemler, static anahtar sözcüğü kullanılarak tanımlanan statik yöntemlerdir. Ancak, getArea () statik değildir. Aşağıdaki yapı kullanılarak, nesneden çağırılmalıdır. objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).

“Statik Değişkenler, Sabitler ve Yöntemler” bölümünde daha fazla açıklama yapılacaktır.



Referans Veri Alanları

Veri alanları referans tiplerinde olabilir. Örneğin, aşağıdaki Student sınıfı, String türünün bir veri alanı adını içerir.

```
public class Student {  
    String name; // name has default value null  
    int age; // age has default value 0  
    boolean isScienceMajor; // isScienceMajor has default value false  
    char gender; // c has default value '\u0000'  
}
```



Null Değeri

Bir referans türündeki veri alanı herhangi bir nesneye referans vermiyorsa, veri alanı özel bir sabit değeri tutar, bu değere null değer denir.



Veri Alanı için Varsayılan Değer

Bir veri alanının varsayılan değeri referans türü için null, sayısal tür için 0, bir boolean türü için false ve karakter türü için '\u0000' şeklindedir.

Ancak, Java bir metot içindeki yerel bir değişkene varsayılan değer atamaz.

```
public class Test {  
    public static void main(String[] args) {  
        Student student = new Student();  
        System.out.println("name? " + student.name);  
        System.out.println("age? " + student.age);  
        System.out.println("isScienceMajor? " + student.isScienceMajor);  
        System.out.println("gender? " + student.gender);  
    }  
}
```

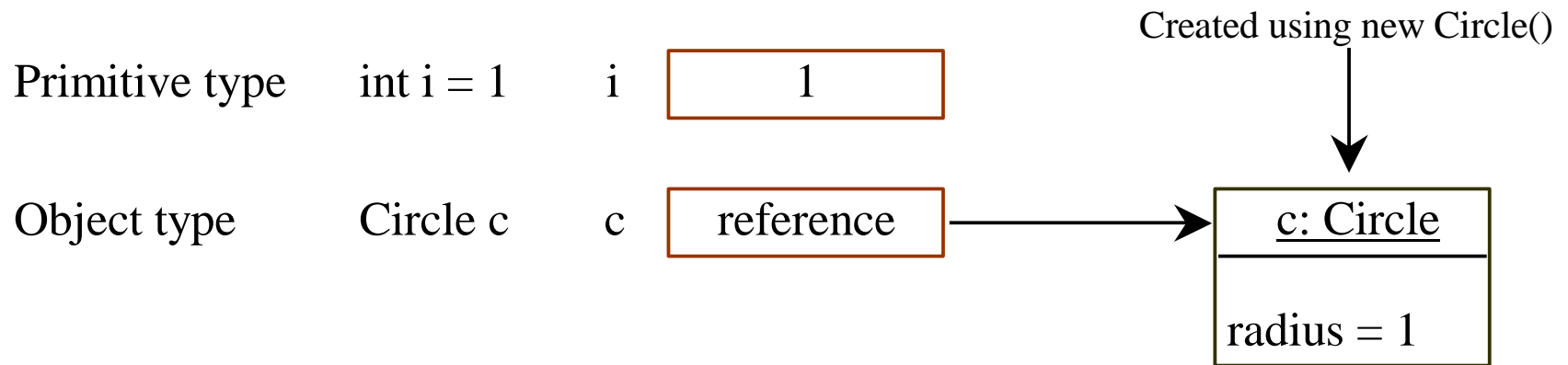
Örnek

Java, bir metot içindeki yerel değişkene varsayılan değer atamaz.

```
public class Test {  
    public static void main(String[] args) {  
        int x; // x has no default value  
        String y; // y has no default value  
        System.out.println("x is " + x);  
        System.out.println("y is " + y);  
    }  
}
```

Compilation error: variables not
initialized

İlkel Veri Türlerinin Değişkenleri ile Nesne Türleri Arasındaki Farklar



İlkel Veri Tipleri ve Nesne Türlerinin Değişkenlerini Kopyalama

Primitive type assignment $i = j$

Before:

i 1

j 2

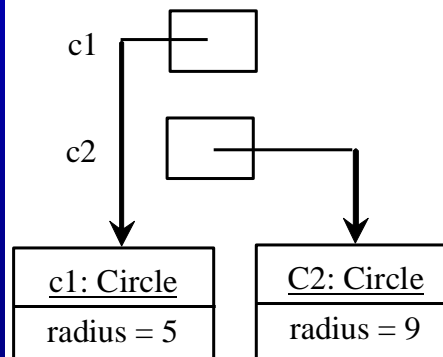
After:

i 2

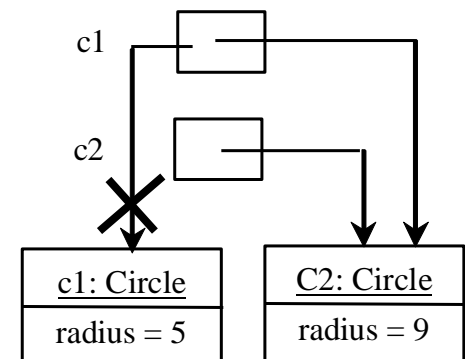
j 2

Object type assignment $c1 = c2$

Before:



After:



Çöp Toplama (Garbage Collection)

- Önceki şekilde gösterildiği gibi, $c1 = c2$ atama ifadesinden sonra, $c1$, $c2$ tarafından başvurulanan aynı nesneyi gösterir. Daha önce $c1$ tarafından başvurulanan nesne artık başvuruda bulunmuyor. Bu nesneye çöp denir. Çöp JVM tarafından otomatik olarak toplanır.



Çöp Toplama (Garbage Collection)

İPUCU: Bir nesnenin artık gerekli olmadığını biliyorsanız, açıkça nesne için bir başvuru değişkenine null atayabilirsiniz. JVM, nesneye herhangi bir değişken tarafından referans gösterilmediğinde otomatik olarak alanı toplar.



Tarih Sınıfı (The Date Class)

Java, `java.util.Date` sınıfında sistemden bağımsız bir tarih ve saat enkapsülasyonu sağlar. Geçerli tarih ve saate bir örnek oluşturmak için `Date` sınıfını kullanabilir ve tarih ve saati bir string olarak döndürmek için `toString` yöntemini kullanabilirsiniz.

The + sign indicates
public modifier



java.util.Date	
+Date()	
+Date(elapseTime: long)	
+toString(): String	
+getTime(): long	
+setTime(elapseTime: long): void	

Constructs a Date object for the current time.

Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.

Returns a string representing the date and time.

Returns the number of milliseconds since January 1, 1970, GMT.

Sets a new elapse time in the object.

Tarih Sınıfı Örneği (The Date Class Example)

Örneğin aşağıdaki kod,

```
java.util.Date date = new java.util.Date();  
System.out.println(date.toString());
```

String' i aşağıdaki gibi gösterir,

Sun Mar 09 13:50:19 EST 2003.



Rastgele Sınıfı (The Random Class)

`Math.random ()` yöntemini, 0.0 ile 1.0 arasında (1.0 hariç) rasgele bir double değer elde etmek için kullandık. `Java.util.Random` sınıfında daha kullanışlı bir rastgele sayı üretici sağlanmıştır.

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (exclusive).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (exclusive).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (exclusive).
+nextBoolean(): boolean	Returns a random boolean value.

Rastgele Sınıfı Örneği (The Random Class Example)

İki Rastgele nesne aynı çekirdeğe sahipse, aynı sayı dizilerini oluştururlar. Örneğin, aşağıdaki kod aynı çekirdek (3) ile iki rastgele nesne oluşturur.

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
    System.out.print(random1.nextInt(1000) + " ");
Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
    System.out.print(random2.nextInt(1000) + " ");
```

From random1: 734 660 210 581 128 202 549 564 459 961

From random2: 734 660 210 581 128 202 549 564 459 961

Örnek Değişkenleri ve Metotları

Örnek değişkenleri belirli bir örneğe aittir.

Örnek metotları, sınıfın bir örneği tarafından çağrılır.



Statik Değişkenler, Sabitler ve Metotlar

Statik değişkenler, sınıfın tüm örnekleri tarafından paylaşırlar.

Statik yöntemler belirli bir nesneye bağlı değildirler.

Statik sabitler, sınıfın bütün örnekleri tarafından paylaşılan final değişkenlerdir.



Statik Değişkenler, Sabitler ve Metotlar

Statik değişkenleri, sabitleri ve yöntemleri bildirmek için statik değiştirici (modifier) kullanılır.

declare static variable

```
static int numberOfObjects;
```

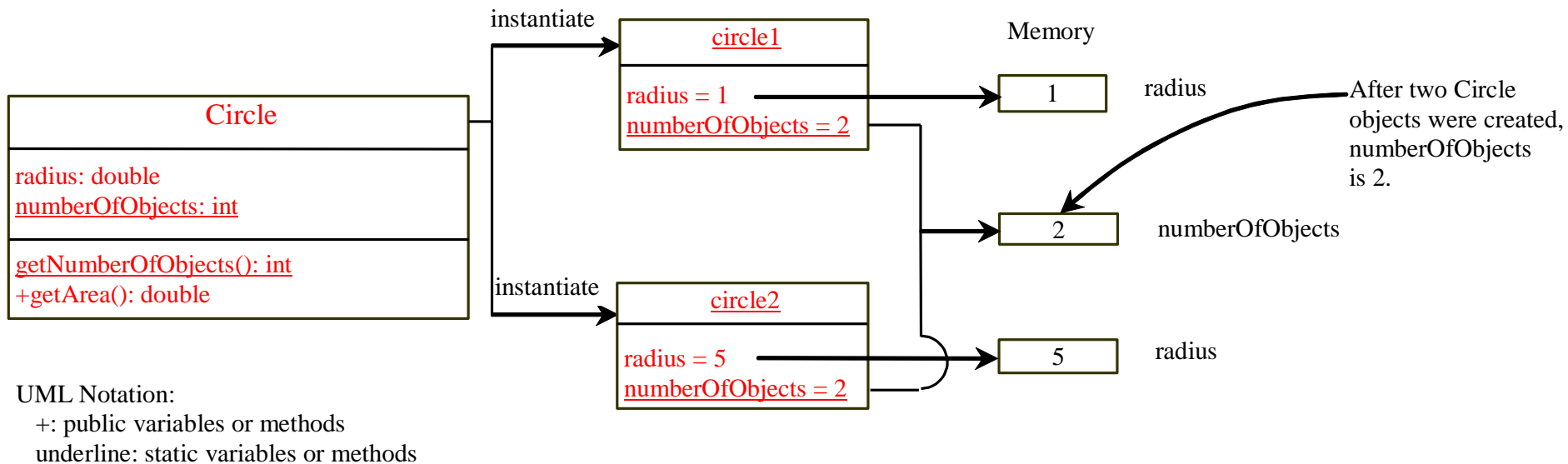
define static method

```
static int getNumberOfObjects() {  
    return numberOfObjects;  
}
```

Constants in a class are shared by all objects of the class. Thus, constants should be declared as **final static**. For example, the constant **PI** in the **Math** class is defined as follows:

```
final static double PI = 3.14159265358979323846;
```

Statik Değişkenler, Sabitler ve Metotlar



Örnek değişkenleri örneklere aittir ve birbirinden bağımsız bellek deposuna sahiptirler. Statik değişkenler aynı sınıfın tüm örnekleri tarafından paylaşılırlar.

Example of Using Instance and Class Variables and Method

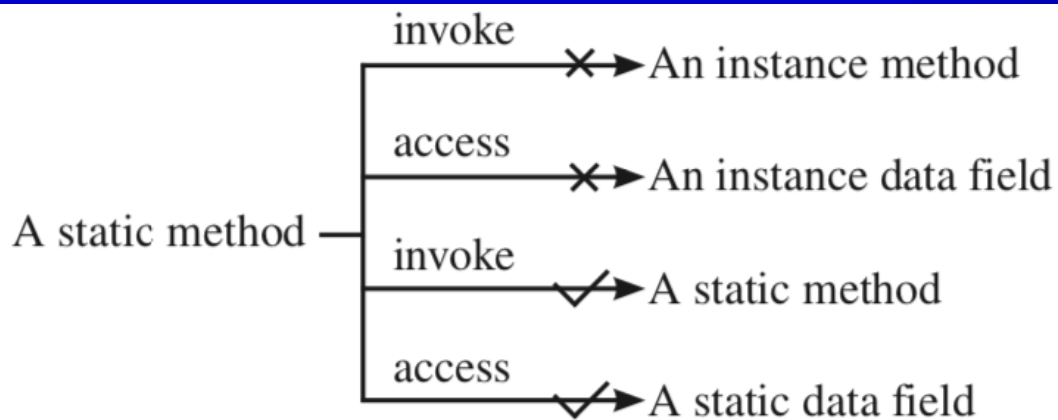
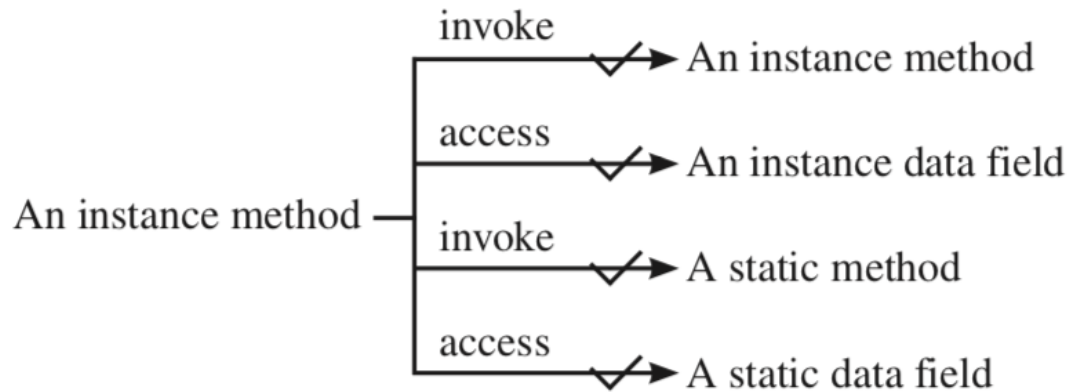
Amaç: Örnek ve sınıf değişkenlerinin rollerini ve kullanımlarını gösterin. Bu örnek, oluşturulan Circle nesnelerinin sayısını izlemek için bir sınıf değişkeni numberOfObjects ekler.

Circle2

TestCircle2

Run





```

1  public class A {
2      int i = 5;
3      static int k = 2;
4
5      public static void main(String[] args) {
6          int j = i; // Wrong because i is an instance variable
7          m1(); // Wrong because m1() is an instance method
8      }
9
10     public void m1() {
11         // Correct since instance and static variables and methods
12         // can be used in an instance method
13         i = i + k + m2(i, k);
14     }
15
16     public static int m2(int i, int j) {
17         return (int)(Math.pow(i, j));
18     }
19 }

```

```

A a=new A();
int j=a.i;
a.m1();

```

Görünürlük Değiştiriciler ve Erişimci / Mutator Yöntemleri (Visibility Modifiers and Accessor/Mutator Methods)

Görünürlük değiştiricileri, bir sınıfın ve üyelerinin görünürlüğünü belirtmek için kullanılabilirler.

Varsayılan olarak, sınıfa, değişkene veya yönteme aynı paketdeki herhangi bir sınıftan erişilebilir.



Görünürlük Değiştiriciler ve Erişimci / Mutator Yöntemleri (Visibility Modifiers and Accessor/Mutator Methods)

□ `public`

Sınıf, veri veya yöntem herhangi bir paketteki herhangi bir sınıfa görünür durumdadır.

□ `private`

Verilere veya yöntemlere yalnızca bildiren sınıf tarafından erişilebilir.

Get ve set yöntemleri, `private` özellikleri okumak ve değiştirmek için kullanılır.



package p1;

```
public class C1 {  
    public int x;  
    int y;  
    private int z;  
  
    public void m1() {  
    }  
    void m2() {  
    }  
    private void m3() {  
    }  
}
```

```
public class C2 {  
    void aMethod() {  
        C1 o = new C1();  
        can access o.x;  
        can access o.y;  
        cannot access o.z;  
  
        can invoke o.m1();  
        can invoke o.m2();  
        cannot invoke o.m3();  
    }  
}
```

package p2;

```
public class C3 {  
    void aMethod() {  
        C1 o = new C1();  
        can access o.x;  
        cannot access o.y;  
        cannot access o.z;  
  
        can invoke o.m1();  
        cannot invoke o.m2();  
        cannot invoke o.m3();  
    }  
}
```

package p1;

```
class C1 {  
    ...  
}
```

```
public class C2 {  
    can access C1  
}
```

package p2;

```
public class C3 {  
    cannot access C1;  
    can access C2;  
}
```

private değiştirici, bir sınıf içinde erişimi kısıtlar, varsayılan değiştirici bir paket içinde erişimi kısıtlar ve public değiştirici sınırsız erişim sağlar.

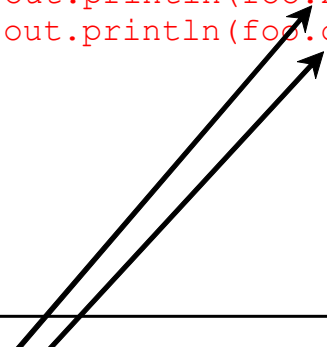
NOT

Bir nesne, (b) 'de gösterildiği gibi kendi private üyelerine erişemez. Bununla birlikte, nesne, (a) 'da gösterildiği gibi kendi sınıfında bildirilmişse problem yoktur.

```
public class Foo {  
    private boolean x;  
  
    public static void main(String[] args) {  
        Foo foo = new Foo();  
        System.out.println(foo.x);  
        System.out.println(foo.convert());  
    }  
  
    private int convert(boolean b) {  
        return x ? 1 : -1;  
    }  
}
```

(a) This is OK because object foo is used inside the Foo class

```
public class Test {  
    public static void main(String[] args) {  
        Foo foo = new Foo();  
        System.out.println(foo.x);  
        System.out.println(foo.convert(foo.x));  
    }  
}
```



(b) This is wrong because x and convert are private in Foo.

Neden Veri Alanları private Olmalı?

Verileri korumak için.

Sınıfın bakımını kolaylaştırmak için.



Neden Veri Alanları private Olmalı?

Private bir veri alanına, private alanı tanımlayan sınıfın dışındaki bir nesne tarafından erişilemez. Ancak, bir müşterinin sıklıkla bir veri alanını alması ve değiştirmesi gerekebilir. Private bir veri alanını erişilebilir hale getirmek için değerini döndürmek için bir getter yöntemi kullanılır. Private bir getter (veya erişimci) veri alanının güncellenmesini sağlamak için, yeni bir değer belirlemek için bir setter yöntem kullanılır.

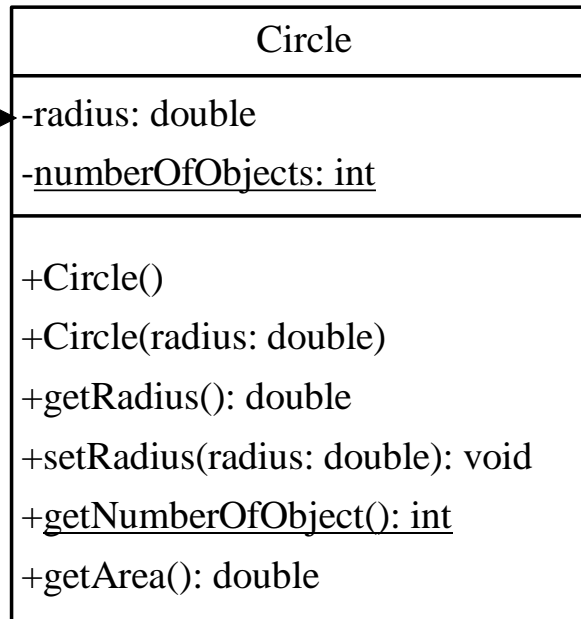


- *getter metot aşağıdaki gibi kullanılır,*
- **public** returnType *getProperty*Name()
If the **returnType** is **boolean**, the getter method should be defined as follows by convention:
- **public boolean** *isProperty*Name()
- *setter metot aşağıdaki gibi kullanılır,*
- **public void** *setProperty*Name(*dataType* *propertyValue*)



Veri Alanı Kapsülleme Örneği (Example of Data Field Encapsulation)

The - sign indicates private modifier



The radius of this circle (default: 1.0).

The number of circle objects created.

Constructs a default circle object.

Constructs a circle object with the specified radius.

Returns the radius of this circle.

Sets a new radius for this circle.

Returns the number of circle objects created.

Returns the area of this circle.

Circle3

TestCircle3

Run

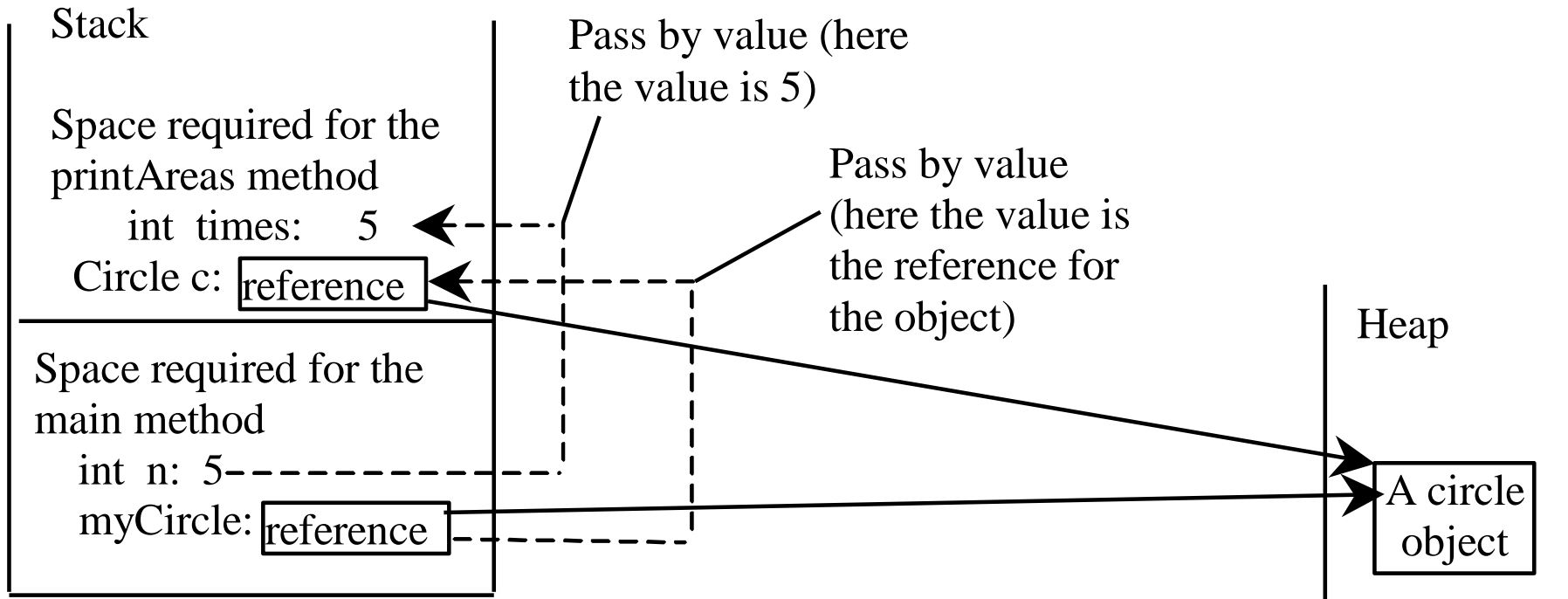
Nesneleri Yöntemlere Geçirmek

- ❑ Passing by value for primitive type value
(the value is passed to the parameter)
- ❑ (değer parametreye aktarılır)
- ❑ Passing by value for reference type value
(the value is the reference to the object)
- ❑ (değer, nesneye yapılan referanstır)

TestPassObject

Run

Nesneleri Yöntemlere Geçirmek



The value of `n` is passed to `times`, and the reference to `myCircle` is passed to `c` in the `printAreas` method.

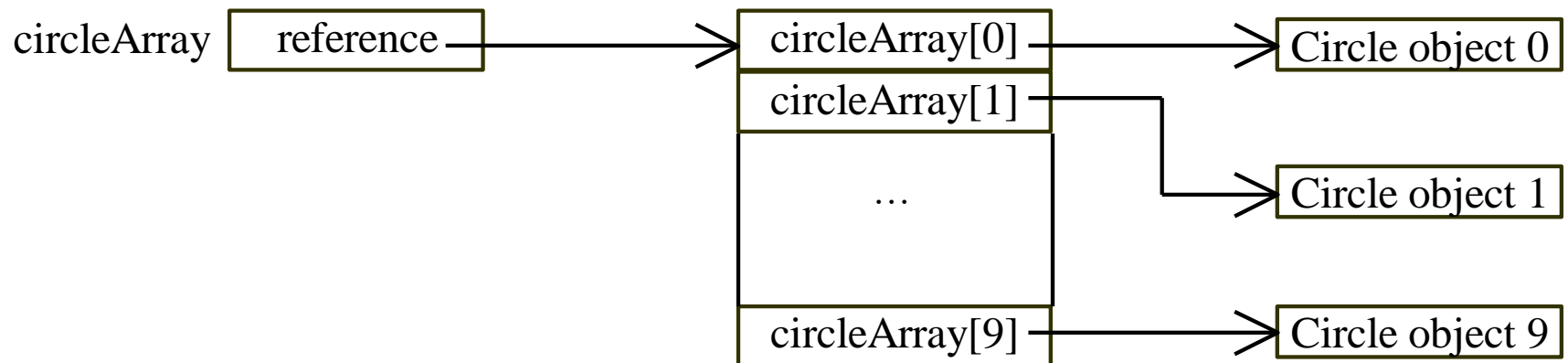
Nesneler Dizisi

- `Circle[] circleArray = new Circle[10];`
To initialize **circleArray**, you can use a **for** loop as follows:
- `for (int i = 0; i < circleArray.length; i++) {`
- `circleArray[i] = new Circle();`
- `}`

Bir nesne dizisi aslında referans değişkenlerin bir dizisidir. Bu yüzden `circleArray [1] .getArea ()` işlevini kullanmak, bir sonraki şekilde gösterildiği gibi iki referans seviyesini içerir. `circleArray`, dizinin tamamına başvurur. `circleArray [1]`, bir `Circle` nesnesine başvuruyor.

Nesneler Dizisi

```
Circle[] circleArray = new Circle[10];
```



Nesneler Dizisi

Summarizing the areas of the circles

TotalArea

Run



Değiştirilebilir Nesneler ve Sınıflar

(Immutable Objects and Classes)

Bir nesnenin içeriği, nesne oluşturulduktan sonra değiştirilemezse, nesneye değişmez nesne, sınıfına ise değişmez sınıf denir. Yukarıdaki örnekte Circle sınıfındaki set yöntemini silerseniz, sınıf değişmez olur çünkü radius özeldir ve set yöntemi olmadan değiştirilemez.

Tüm private veri alanlarına ve mutator' lara sahip olmayan bir sınıf mutlaka değiştirilemezdir. Örneğin, aşağıdaki sınıf Student tüm özel private alanlarına sahiptir ve mutator' ları yoktur, ancak değişkendir.



Örnek

```
public class Student {
    private int id;
    private BirthDate birthDate;

    public Student(int ssn,
        int year, int month, int day) {
        id = ssn;
        birthDate = new BirthDate(year, month, day);
    }

    public int getId() {
        return id;
    }

    public BirthDate getBirthDate() {
        return birthDate;
    }
}
```

```
public class BirthDate {
    private int year;
    private int month;
    private int day;

    public BirthDate(int newYear,
        int newMonth, int newDay) {
        year = newYear;
        month = newMonth;
        day = newDay;
    }

    public void setYear(int newYear) {
        year = newYear;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student(111223333, 1970, 5, 3);
        BirthDate date = student.getBirthDate();
        date.setYear(2010); // Now the student birth year is changed!
    }
}
```

Hangi Sınıf Değişmezdir (immutable)

Bir sınıfın değişmez olması için, tüm veri alanlarını özel olarak işaretlemeli ve mutator yöntemleri içermemeli ve değişken bir veri alanı nesnesine bir referans verecek hiçbir erişimci yöntemi sağlamamalıdır.



Değişkenlerin Kapsamı

- ❑ Örnek ve statik değişkenlerin kapsamı tüm sınıftır. Bir sınıf içinde herhangi bir yerde bildirilebilirler.
- ❑ Bir yerel değişkenin kapsamı, bildiriminden başlar ve değişkeni içeren bloğun sonuna kadar devam eder. Yerel bir değişken, kullanılmadan önce açıkça başlatılmalıdır.



```
public class Circle {  
    public double getArea() {  
        return radius * radius * Math.PI;  
    }  
  
    private double radius = 1;  
}
```

(a) The variable **radius** and method **getArea()** can be declared in any order.

```
public class F {  
    private int i;  
    private int j = i + 1;  
}
```

(b) **i** has to be declared before **j** because **j**'s initial value is dependent on **i**.



this Anahtar Kelimesi (The this Keyword)

- Bu anahtar kelime, bir nesnenin kendisine başvuran bir referansın adıdır. Bu anahtar kelimenin yaygın bir kullanımı, bir sınıfın gizli veri alanlarına referanstır.
- Bir yapıcının aynı sınıftaki başka bir yapıcıyı çağırmasını sağlamak için bu anahtar kelimenin başka bir yaygın



Refers to data field **radius** in this object.

```
private double radius;  
  
public void setRadius(double radius) {  
    this.radius = radius;  
}
```

(a) `this.radius` refers the `radius` data field in this object.

Here, **radius** is the parameter in the method.

```
private double radius = 1;  
  
public void setRadius(double radius) {  
    radius = radius;  
}
```

(b) `radius` is the parameter defined in the method header.



```

public class Circle {
    private double radius;

    ...

    public double getArea() {
        return this.radius * this.radius * Math.PI;
    }

    public String toString() {
        return "radius: " + this.radius
            + "area: " + this.getArea();
    }
}

```

(a)

Equivalent

```

public class Circle {
    private double radius;

    ...

    public double getArea() {
        return radius * radius * Math.PI;
    }

    public String toString() {
        return "radius: " + radius
            + "area: " + getArea();
    }
}

```

(b)



Gizli Veri Alanlarına Referans

```
public class Foo {  
    private int i = 5;  
    private static double k = 0;  
  
    void setI(int i) {  
        this.i = i;  
    }  
  
    static void setK(double k) {  
        Foo.k = k;  
    }  
}
```

Suppose that f1 and f2 are two objects of Foo.

Invoking f1.setI(10) is to execute
this.i = 10, where **this** refers f1


Invoking f2.setI(45) is to execute
this.i = 45, where **this** refers f2



Aşırı Yüklü Yapıcıyı (Constructor) Çağırma

```
public class Circle {  
    private double radius;
```

```
    public Circle(double radius) {  
        this.radius = radius;  
    }
```

 this must be explicitly used to reference the data field radius of the object being constructed

```
    public Circle() {  
        this(1.0);  
    }
```

 this is used to invoke another constructor

```
    public double getArea() {  
        return this.radius * this.radius * Math.PI;  
    }  
}
```

 Every instance variable belongs to an instance represented by this, which is normally omitted

Array Lists



Veri yapısı nedir?

Veri yapısı, bir şekilde organize edilmiş bir veri koleksiyonudur. Bir veri yapısı sadece veri depolamakla kalmaz, aynı zamanda yapı içindeki veri işleme operasyonlarını da destekler. Örneğin, bir dizi, sıralı düzende tutululan bir veri yapısıdır.

Dizinin boyutunu bulabilir, dizide veri saklayabilir, alabilir ve değiştirebilirsiniz.

Dizi kavramı basit ve kullanımı kolaydır, ancak iki kısıtı bulunmaktadır:



Dizilerin Kısıtları

- ❑ Bir dizi oluşturulduktan sonra dizinin boyutu bir daha değiştirilemez.
- ❑ Dizi, ekleme, silme, sıralama ve arama işlemleri için yetersiz destek sağlar.



Nesne Dizileri

- Diziler nesne türünde değişken tutabilir.

Nesne türünde değişken tutan dizi tanımı

– `SinifAdı[] degiskenAdi=new SinifAdi[];`

Nesne türünde değişken tutan dizi örneği

`Araba[] a1=new Araba[5];`

`Circle[] circleArray = new Circle[10];`



Nesne Dizileri

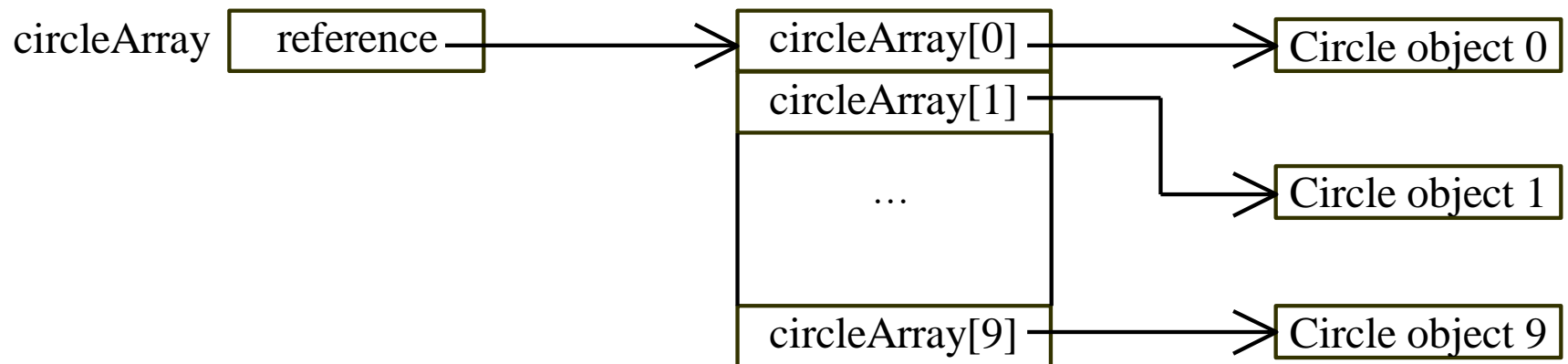
circleArray dizisini başlatmak için aşağıdaki gibi **for** döngüsü kullanabiliriz:

```
for (int i = 0; i < circleArray.length; i++) {  
    circleArray[i] = new Circle();  
}
```

*Bir nesne dizisi aslında referans değişkenlerin bir dizisidir. Bu yüzden `circleArray [1] .getArea ()` işlevini kullanmak, bir sonraki şekilde gösterildiği gibi iki referans seviyesini içerir. `circleArray`, dizinin tamamına başvurur. `circleArray [1]`, bir `Circle` nesnesine başvuruyor.

Nesne Dizileri

```
Circle[] circleArray = new Circle[10];
```



İki aşamalı referans

Dizi nesnenin referans değişkenine referans eder.

Nesne Dizileri

Summarizing the areas of the circles

TotalArea

Run

Nesne Yönelimli Veri Yapıları

Nesne yönelimli düşüncede, veri yapısı, veri veya elemanlar olarak adlandırılan diğer nesneleri depolayan bir nesnedir. Bu yüzden bazıları bir veri yapısını bir konteyner nesnesi (*container object*) veya bir koleksiyon nesnesi (*collection object*) olarak adlandırırlar. Bir veri yapısını tanımlamak esasen bir sınıf deklare etmektir. Veri yapısı sınıfı, veri depolamak için veri alanlarını kullanmalı ve ekleme-silme gibi işlemleri desteklemek için yöntemler sağlamalıdır. Veri yapısı oluşturmak, sınıftan bir örnek oluşturmaktır. O zaman veri yapısına, bir öge eklemek veya veri yapısından bir ögeyi silmek gibi veri yapısını değiştirmek için kullanılan örnek(instance) üzerindeki yöntemleri uygulayabilirsiniz.



Listeler

Liste, verileri sırayla depolamak için popüler bir veri yapısıdır. Örneğin, bir öğrenci listesi, mevcut odaların bir listesi, şehirlerin bir listesi ve bir kitap listesi vb. Listeler kullanılarak depolanabilir. Bir listedeki ortak işlemler genellikle şunlardır:

- Listeden bir eleman bulup getirmek
- Listeye yeni eleman eklemek
- Listeden bir eleman silmek
- Listenin kaç tane elemanı olduğunu bulmak
- Bir elemanın listede olup olmadığını bulmak
- Listenin boş olup olmadığını bulmak



Listeyi uygulamanın iki yolu

- Bir yol, elemanları saklamak için bir dizi kullanmaktır. Dizi dinamik olarak oluşturulur. Dizinin kapasitesi aşılsa, daha büyük bir dizi oluşturulur ve tüm öğeleri geçerli diziden yeni diziye kopyalanır.
- Diğer yol ise bağlantılı bir yapı (linked structure) kullanmaktır. Bağlantılı bir yapı düğümlerden oluşur. Her düğüm bir elemanı tutmak için dinamik olarak oluşturulur. Bir liste oluşturmak için tüm düğümler birbirine bağlanır. (Daha sonra işlenecek)



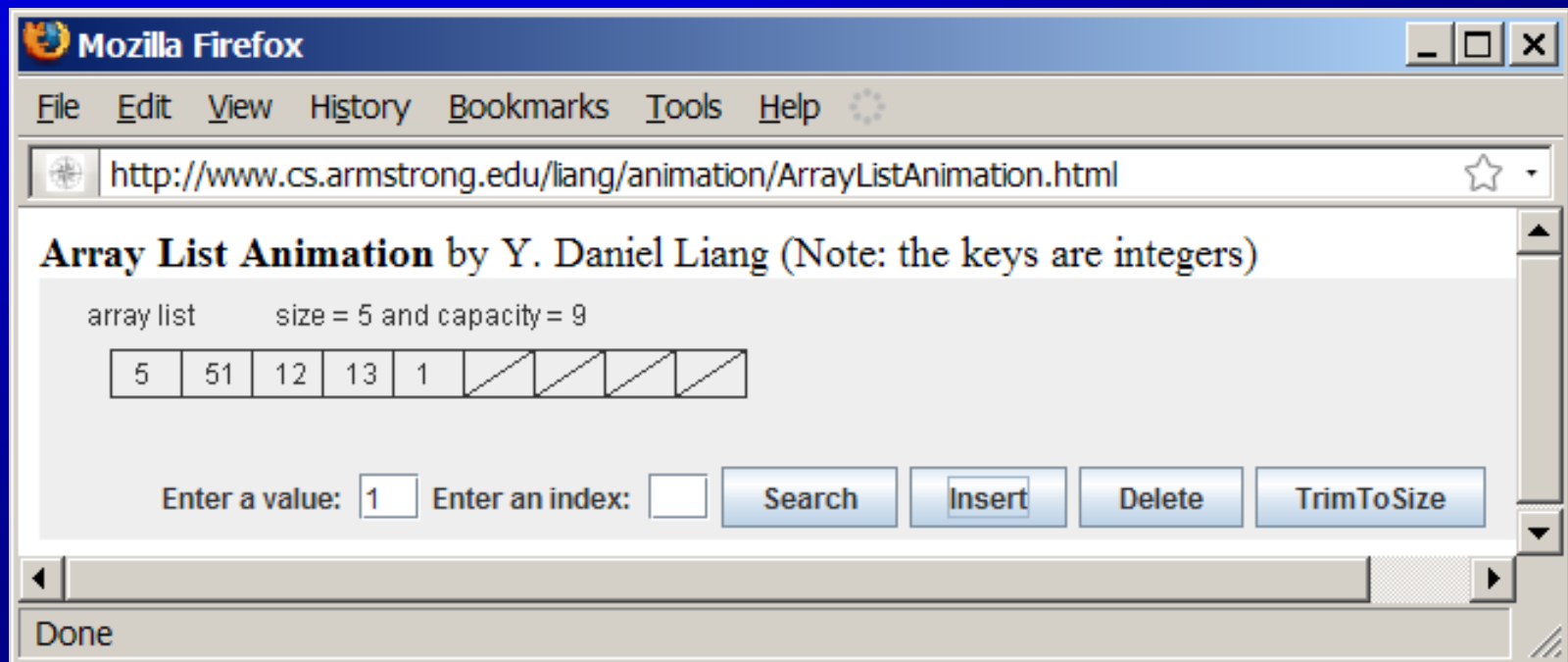
Array Lists

- Dizi sabit boyutlu bir veri yapısıdır. Bir dizi oluşturulduktan sonra, boyutu değiştirilemez.
- Dinamik olarak büyüyeabilen dizi tanımlamak için ArrayList kullanılır. İşin püf noktası, geçerli dizi listede yeni öğeler tutamazsa, geçerli diziyi değiştirmek için daha büyük bir dizi oluşturmaktır(büyüyüp küçülebilen yapılar).
- İlk olarak, bir dizi (Object [] tipindeki veri) default boyutta yaratılır. Diziye yeni bir öge eklerken, önce dizide yeterli yer olduğundan emin olun. Değilse, geçerli boyutun iki katı boyutta yeni bir dizi oluşturun. Öğeleri geçerli diziden yeni diziye kopyalayın. Yeni dizi şimdi geçerli dizi haline gelir.



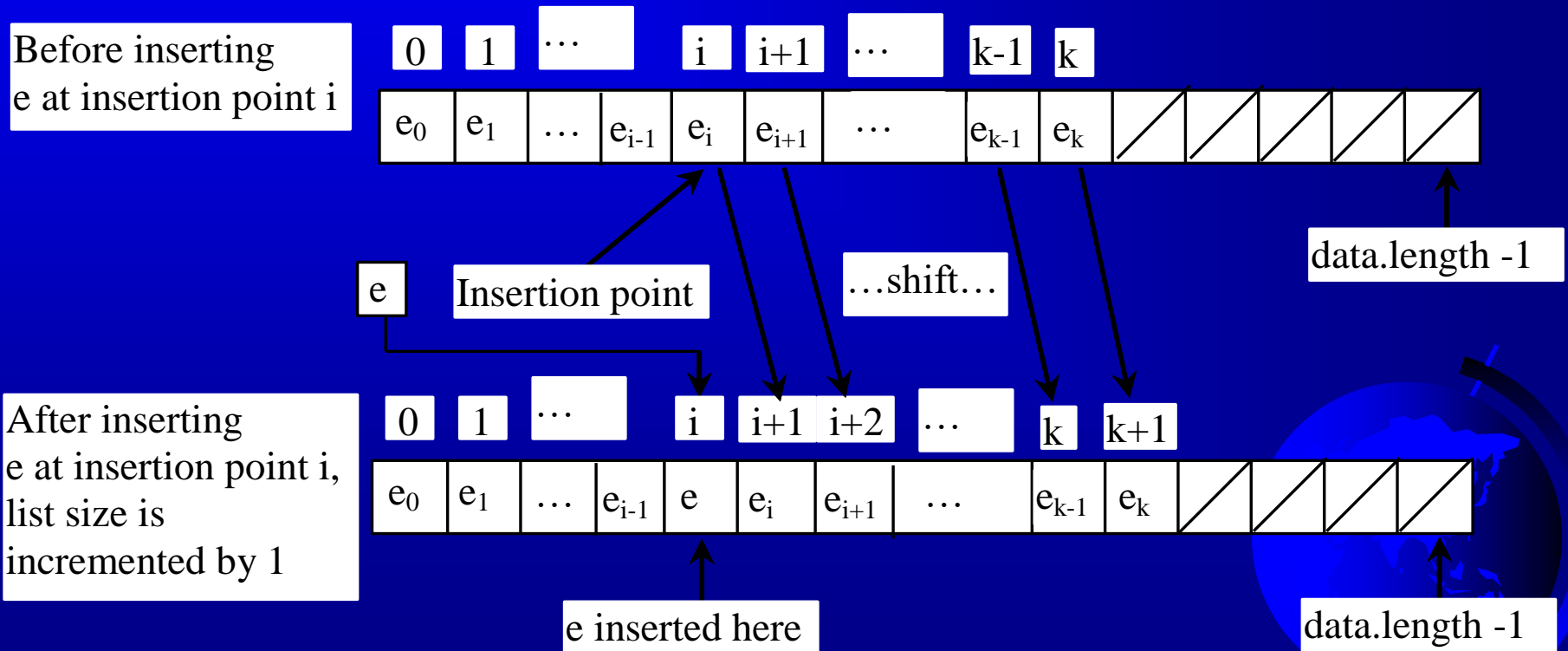
Array List Animation

<https://yongdanielliang.github.io/animation/web/ArrayListAnimation.html>



Ekleme

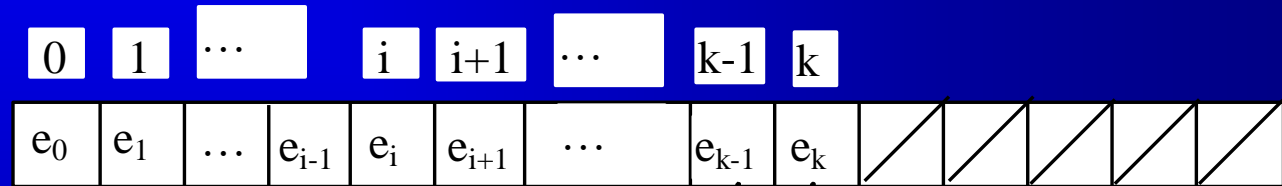
Belirtilen bir indekse yeni bir öge eklemekten önce, indeksden sonraki tüm öğeleri sağa kaydırın ve liste boyutunu 1 arttırın.



Silme

Belirtilen bir indeksdeki bir öğeyi kaldırmak için indeksden sonraki tüm öğeleri bir konum sola kaydırın ve liste boyutunu 1 azaltın.

Before deleting the element at index i

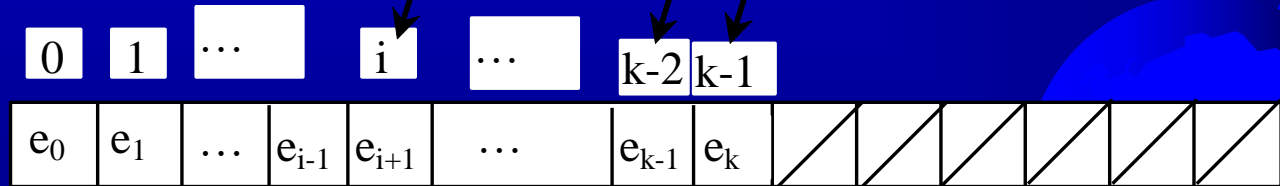


Delete this element

...shift...

$\text{data.length} - 1$

After deleting the element, list size is decremented by 1



$\text{data.length} - 1$

ArrayList Sınıfı

Nesneleri saklamak için bir dizi oluşturabilirsiniz. Ancak dizi oluşturulduktan sonra dizinin boyutu sabittir. Java, sınırsız sayıda nesneyi saklamak için kullanılabilecek ArrayList sınıfını sağlar.

java.util.ArrayList

+ArrayList()
+add(o: Object) : void
+add(index: int, o: Object) : void
+clear(): void
+contains(o: Object): boolean
+get(index: int) : Object
+indexOf(o: Object) : int
+isEmpty(): boolean
+lastIndexOf(o: Object) : int
+remove(o: Object): boolean
+size(): int
+remove(index: int) : Object
+set(index: int, o: Object) : Object

Creates an empty list.

Appends a new element o at the end of this list.

Adds a new element o at the specified index in this list.

Removes all the elements from this list.

Returns true if this list contains the element o.

Returns the element from this list at the specified index.

Returns the index of the first matching element in this list.

Returns true if this list contains no elements.

Returns the index of the last matching element in this list.

Removes the element o from this list.

Returns the number of elements in this list.

Removes the element at the specified index.

Sets the element at the specified index.

Aşağıdaki ifade bir **ArrayList** oluşturur ve **cities** değişkenine referansını bağlar. Bu **ArrayList** nesnesi dizgeleri (string) saklamak için kullanılmaktadır.

□ *ArrayList<String> cities = new ArrayList<String>();*

Aşağıdaki ifade bir **ArrayList** oluşturur ve **dates** değişkenine referansını bağlar. Bu **ArrayList** nesnesi tarihleri (date) saklamak için kullanılmaktadır

□ *ArrayList<java.util.Date> dates = new ArrayList<java.util.Date>();*



ArrayList Örneği

```
ArrayList al = new ArrayList();  
    System.out.println("Initial size of al: " +  
al.size());
```

```
// Eleman Ekleme
```

```
al.add("C");
```

```
al.add("A");
```

```
al.add("E");
```

```
al.add("B");
```

```
al.add("D");
```



ArrayList Örneği

```
import java.util.*;  
public class ArrayList01 {  
    public static void main(String[] args) {  
        ArrayList<String> list = new ArrayList<String>();  
        list.add("İzmir");  
        list.add("Erzurum");  
        list.add("Giresun");  
        list.add("Konya");  
        list.add("Antalya");  
        System.out.println(list);  
        System.out.println("3: " + list.get(3));  
        System.out.println("0: " + list.get(0)); }  
}
```



TABLE 11.1 Differences and Similarities between Arrays and **ArrayList**

Operation	Array	ArrayList
Creating an array/ArrayList	<code>String[] a = new String[10]</code>	<code>ArrayList<String> list = new ArrayList<>();</code>
Accessing an element	<code>a[index]</code>	<code>list.get(index);</code>
Updating an element	<code>a[index] = "London";</code>	<code>list.set(index, "London");</code>
Returning size	<code>a.length</code>	<code>list.size();</code>
Adding a new element		<code>list.add("London");</code>
Inserting a new element		<code>list.add(index, "London");</code>
Removing an element		<code>list.remove(index);</code>
Removing an element		<code>list.remove(Object);</code>
Removing all elements		<code>list.clear();</code>

Bir diziden ArrayList oluşturma örneği:

```
String[] array = {"red", "green", "blue"};  
ArrayList<String> list = new ArrayList<>(Arrays.asList(array));
```



LISTING 11.8 TestArrayList.java

```
1  import java.util.ArrayList;
2
3  public class TestArrayList {
4      public static void main(String[] args) {
5          // Create a list to store cities
6          ArrayList<String> cityList = new ArrayList<>();
7
8          // Add some cities in the list
9          cityList.add("London");
10         // cityList now contains [London]
11         cityList.add("Denver");
12         // cityList now contains [London, Denver]
13         cityList.add("Paris");
14         // cityList now contains [London, Denver, Paris]
15         cityList.add("Miami");
16         // cityList now contains [London, Denver, Paris, Miami]
17         cityList.add("Seoul");
18         // Contains [London, Denver, Paris, Miami, Seoul]
19         cityList.add("Tokyo");
20         // Contains [London, Denver, Paris, Miami, Seoul, Tokyo]
```

```
22 System.out.println("List size? " + cityList.size());
23 System.out.println("Is Miami in the list? " +
24     cityList.contains("Miami"));
25 System.out.println("The location of Denver in the list? "
26     + cityList.indexOf("Denver"));
27 System.out.println("Is the list empty? " +
28     cityList.isEmpty()); // Print false
29
30 // Insert a new city at index 2
31 cityList.add(2, "Xian");
32 // Contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
33
34 // Remove a city from the list
35 cityList.remove("Miami");
36 // Contains [London, Denver, Xian, Paris, Seoul, Tokyo]
37
38 // Remove a city at index 1
39 cityList.remove(1);
40 // Contains [London, Xian, Paris, Seoul, Tokyo]
41
42 // Display the contents in the list
43 System.out.println(cityList.toString());
```

```
45     // Display the contents in the list in reverse order
46     for (int i = cityList.size() - 1; i >= 0; i--)
47         System.out.print(cityList.get(i) + " ");
48     System.out.println();
49
50     // Create a list to store two circles
51     ArrayList<Circle> list = new ArrayList<>();
52
53     // Add two circles
54     list.add(new Circle(2));
55     list.add(new Circle(3));
56
57     // Display the area of the first circle in the list
58     System.out.println("The area of the circle? " +
59         list.get(0).getArea());
60 }
61 }
```



Array ve Array List seçimi

Array

- Eğer eleman sayısı belirliyse ve değişme ihtimali düşükse

ArrayList

- Eğer eleman sayısı değişkense
- List arayüzünün ilave metotlarına ihtiyaç varsa.

