



Wise Quarter  
first class IT courses

# Java

## Team 127 Ders-01

Genel Bilgilendirme  
Programlamaya Giriş



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# Onemli Hatirlatmalar

Bugun  
IT Dunyasinda  
Ilk Gununuz

- 1- Ders Tam Vaktinde Baslar
- 2- Ders Oncesi Hazirlik Yapin  
(Google en yakin arkadasiniz olmalı)
- 3- Derste Aktif Olun  
(Anlamadiklarinizi Mutlaka Sorun)
- 4- Derste Kodlari Kendiniz Yazin  
(Yetistiremiyorsaniz Onceligi Anlamaya Verin)
- 5- Ders Sonrasi Tekrar Yapin  
(En Iyi Tekrar Dersteki Kodlari Kendinizin yazmasidir)
- 6- Basari = Egitim + Calismak
- 7- Grup Calismalari Yapin  
(Derste Anlatilanlara Benzer Sorular Uretin)

# Onemli Hatirlatmalar



## Kullanimi

- 1- Ders esnasinda anlatilan konu disinda paylasim yapmayin.
- 2- Ders sirasinda ileri konulardan soru sormayin.
- 3- Dersle sirasinda veya sonrasinda, slack yazismalarinizda nezaket kurallarina uyun.
- 4- Kod paylasirken snippet kullanin
- 5- Kodla ilgili sorunlarinizda sorularinizi screenshot ile yollayin
- 6- Dersi takip edin ve daha once sorulmus konulari tekrar sormamaya ozen gösterin
- 7- Kurulumla ilgili problemleri teknik destege, dersle ilgili sorulari instructor'a sorun.



# Onemli Hatirlatmalar



- 1- Ders tam zamaninda baslar
- 2- Ders basinda bir onceki gunun kisa bir tekrari yapilir
- 3- Her konu bittiginde, o konu ile ilgili bir test yapilir.



## Programlama Dili Nedir?

Programlama dili, yazılımcının bir algoritmayı ifade etmek amacıyla, bir bilgisayara ne yapmasını istediği anlatmasının tektipleştirilmiş yoludur.



Programlama dilleri, yazılımcının bilgisayara hangi veri üzerinde işlem yapacağını, verinin nasıl depolanıp iletileceğini, hangi koşullarda hangi işlemlerin yapılacağını tam olarak anlatmasını sağlar.

Programlama dilleri sayesinde bir bilgisayarın hangi durumda ne çeşit çıktı verebileceği kontrol edilebilir.

Kısacası programlama dilleri sayesinde bilgisayarlar ve insanlar verimli bir iletişim sağlayabilirler.

## Bilgisayar'in bizim istedigimiz seyi yapabilmesi icin



### 1- Bilgisayar'in anlayacagi dili bizim bilmemiz



### 2- Yazdigimiz kodların bilgisayar tarafından anlasildigini bilmemiz



### 3- Bilgisayarin bizim icin urettigi sonucları anlamlendirmemiz gereklidir.

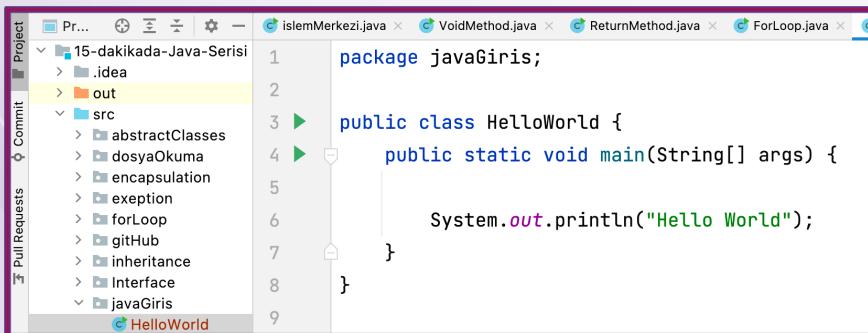
Kisaca ozetlersek, programlama dili bizimle bilgisayar arasindaki iletisimi saglayan dildir.

Peki..Bizimle bilgisayar arasindaki islemlerde patron kim ?

Kimin dedigi olur ?

Tabii ki bilgisayarlar bizim dedigimizi yaparlar

Ama bu istegimizi bilgisayarin anlayacagi ozelliklerde yazmamiz sartiyyla.



```
Pr... Project .idea .out src abstractClasses dosyaOkuma encapsulation exception forLoop gitHub inheritance Interface javaGiris islemMerkezi.java VoidMethod.java ReturnMethod.java ForLoop.java
1 package javaGiris;
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         System.out.println("Hello World");
6     }
7 }
8
9
```

JDK Kodlari derler (Compile)

Ornegin;  
kodlarimizla, Hello World yazdirmak istiyorsak

```
110101010101010010100010101010
10010111000001000100101101010
101010010111000110
010001010010101110001000000101
0010101110001001010111
```



Binary kodlar

Islem

# Nicin Java ?

Java, 1995 Yılında ortaya çıkan high-level, Object Oriented bir program olarak ortaya çıkmıştır.

Java'yi ilk gunden itibaren populer programlama dilleri arasında one cikaran bazi ustunlukleri,

1- Öğrenmesi kolay

2- Dunyada en çok kullanılan programlama dili

3 milyar mobil cihazda Java kullanıyor.

USA'de şirket bilgisayarlarının %97'sinde, kişisel bilgisayarların %89'unda Java kullanılıyor

Linkedin, Uber, Netflix gibi pek çok popüler uygulama Java tabanlı çalışıyor.

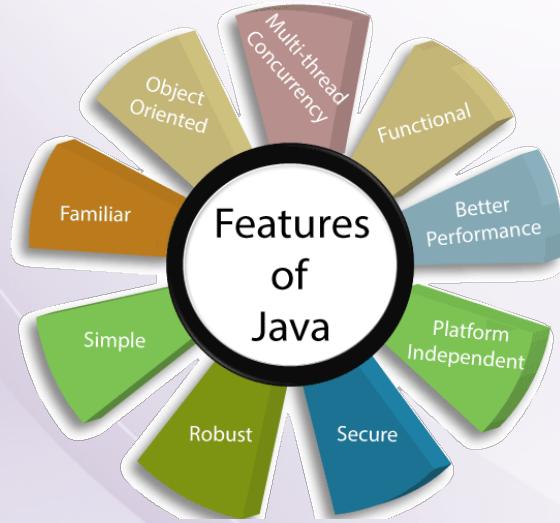
3- Güvenlidir

4- Platform independent'dir

5- Ücretsizdir.

Neden Java:

<https://youtu.be/RUYASEnT6pU>





# Nicin Java / OOP Concept



6- Java, Object Oriented Programming konseptinde calisir.

OOP Konsept, kompleks programlari yapmaya kucuk parcalardan baslayip, sonra bunlari birlestirerek istenen sonuca ulasmaya denir.



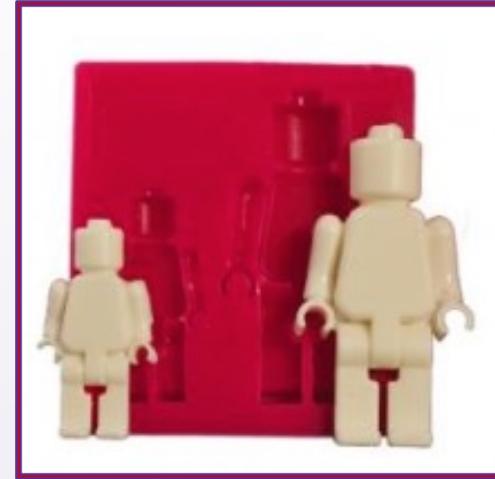
Java'da, once Object (Nesne) olusturmalıyız.

Her obje'nin 2 tur ozelligi olur.

- Feature (variable) – renk, pin sayisi vb...
- Functionality (method) – bizim girdigimiz degere gore degeri degisen ozellikler. Kanatli, tekerli vb..

# Object Nasıl Olusturulur

Objeleri olusturabilmek icin oncelikle kalip olusturmaliyiz.



Kalip varsa, bu kaliptan istedigimiz kadar obje uretebilir ve bu objeleri birlestirip istedigimiz uygulamayı elde edebiliriz.

Java'da obje olusturabilmemiz icin kullanmamız gereken kalip Class( Sinif )'dir.

Her bir obje bir Class'tan turetilmistir ve Class olmadan obje uretmek mumkun degildir.



# Class Hangi Bolumlerden Olusur ?

```
public class C2_MethodCreation2 {  
}  
} // Class sonu
```

Bir Class'da temel 3 bolum bulunur.

1- Class Declaration

2- Curly Braces : Suslu Parantez

3- Class Body : Suslu parantezler arasında kalan, kodlarimizi yazdigimiz bolum.



```
public class HelloWorld {  
  
    int ogrNo=1013;  
    String isim="Ali Can";  
    boolean ogrenciMi=true;  
    double notOrt=87.5;  
  
    public static void main(String[] args) {  
  
        double yazılıNotu=89;  
        double sozluNotu=92;  
  
    }  
  
    public void baskaMethod(){  
  
    }  
}
```

Bir Class Body'sinde variable ve method'lar bulunur.

1- Main Method

2- Variables

3- method'lar



# Main Method Nedir ?

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

Main Method, Java'nin calismaya basladigi giris noktasidir (Entry point)

Main method olusturulurken kullanilacak syntax (yazilacak keyword veya metin) sabittir, degistirilemez.

Parantez icine yazilan (**String[ ] args**) main method'un calismasi icin gerekli argumanların oldugu bir array'dir ve mutlaka yazilmalidir.

**NOT :** main method olmayan class'lar run edilemez (direk calistirilamaz).

```
3  public class HelloWorld {  
4  
5      public static void main(String[] k) {  
6  
7  
8  
9          }  
10     }  
11 }
```

```
3  public class HelloWorld {  
4  
5  
6      public void baskaMethod(){  
7  
8  
9      }  
10     }  
11 }
```

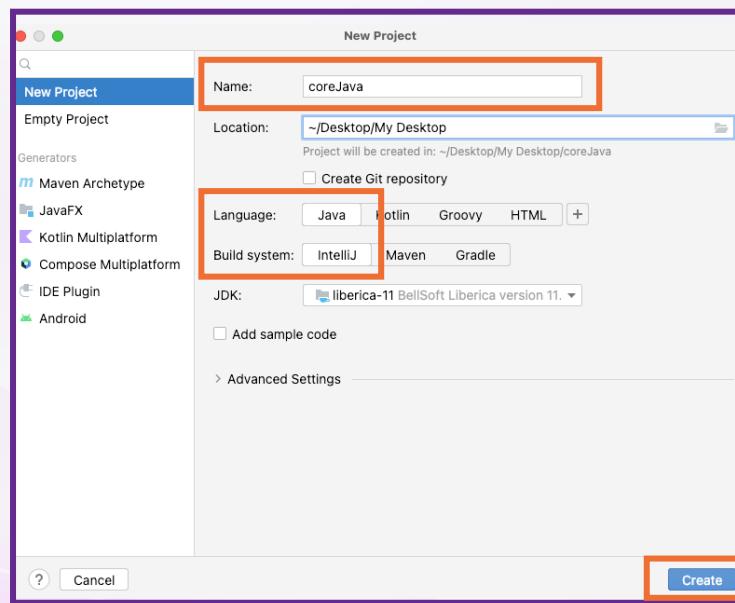
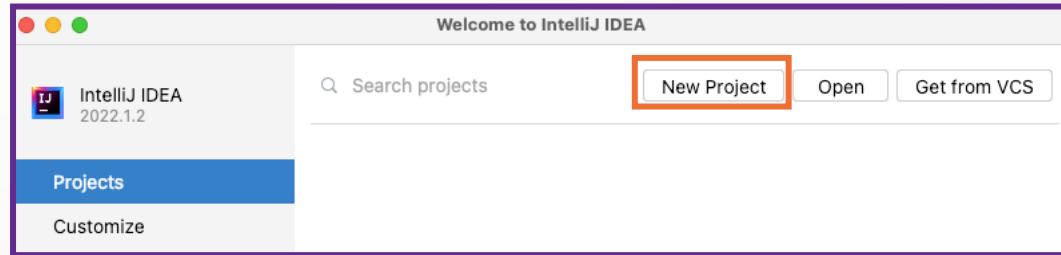
# IntelliJ'de Proje Olusturma

## IntelliJ Nedir?

Java gibi compiler programlar calismak icin ide ( Integrated Development Environment)'ye ihtiyac duyarlar.

Bircok ide olmakla birlikte, piyasada çok kullanildigi ve kod yazimini kolaylastirdigi icin biz intelliJ kullanacagiz.

IntelliJ'de Projects menusunde New Project'i secin,



Acilan menu'de

Name kismina projenin ismini yazin,

Language kisminda Java, Build system kisminda IntelliJ secili oldugunu kontrol edip,

Create butonuna basin.

# IntelliJ'de Proje Olusturma

## 2- Package(Class Dosyasi) olusturma

Acilan projede **src**'ye sag click yapip **New--Package**'i secip istedigimiz ismi yazin ve **finish**'e basin

## 3- Class olusturma

Acilan package ismine sag click yapip **New--Class**'i secip istedigimiz ismi yazin ve **finish**'e basin

## 4- Main method olusturma

Acilan Class icinde **main veya psvm** yazdigimizda, asagida cikan **main** yazisini sectiginizde IntelliJ bizim icin main method olusturacaktir.

## 5- Ilk kodumuzu yazdirma

Main method icerisinde **sout** yazip acilan menuden **system.out.println**'i secin.

```
5  public static void main(String[] k) {  
6      System.out.println("Hello World");  
7  }  
8 }
```

Parantez icine **"Hello World"** yazin

**Yesil run tusu**'na basip class'i calistirin

# Kod'a Comment Ekleme

Kod yazarken ilk hedef calisan bir kod yazmaktır.

Ama asil hedef calisan ve Anlasilabilir kod yazmaktır.

Kodlarimizi hem kendimiz hem de bizden sonra kodlari kullanacak kisilerin daha iyi anlayabilmesi icin, class'a aciklama cumleleri ekleyebiliriz .

```
5 ►   public static void main(String[] k) {  
6  
7     // Tek satiri comment yapmak icin
```

En iyi Kod  
Calisan Kod,  
  
Daha da iyisi  
Calisan ve Anlasilir  
Kod yazmaktir.

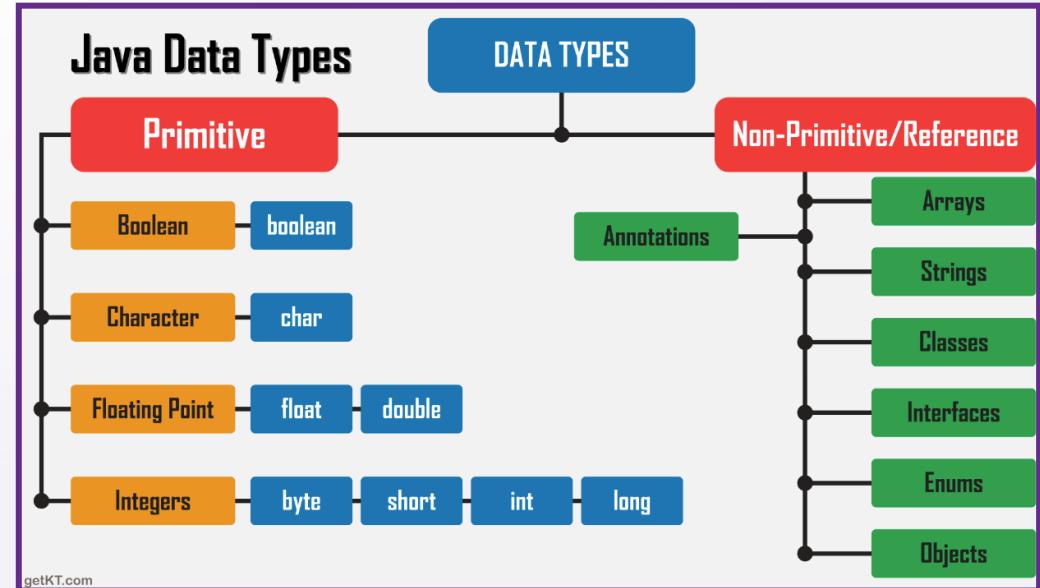
```
10 */  
11 Birden fazla  
12 satiri  
13 comment yapmak icin  
14 */
```

# Data Nedir ?

**Data** is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.

Data (**Veri**), sayilar, kelimeler, olcumler, gozlemler gibi bilgi iceren objelerin bir kolleksiyonudur.

Yazacagimiz her kod, yapacagimiz her program **data**'yi almak, **data**'yi islemek ve sonuc olarak bir **data** olusturmak icin kullanilir.



Yukarida da tanimlandigi gibi data'nin icerdigi bilgi çok farklı olabileceginden, tüm programlama dilleri farklı data turlerini kullanabilmek icin **kendi** kullanacakları **data turlerini** tanımlamışlardır.

Hangi programlama dilini kullanacaksak, oncelikle o dillerde kullanabilecegimiz data turlerini öğrenmeliyiz.



# Data Saklama(Store)

Her data hafizada(memory) bir yer kaplar.

Bir datanin hafizada saklanacagi en kucuk bolum bit'dir.

Her bir bit **1** veya **0** degerlerini icerir.

8 bit bir araya geldiginde bir **byte** olusur.

Her **byte**  $2^8 = 256$  farkli deger alabilir.

Sayı sistemimiz 10'luk sistem oldugu gibi hafiza da

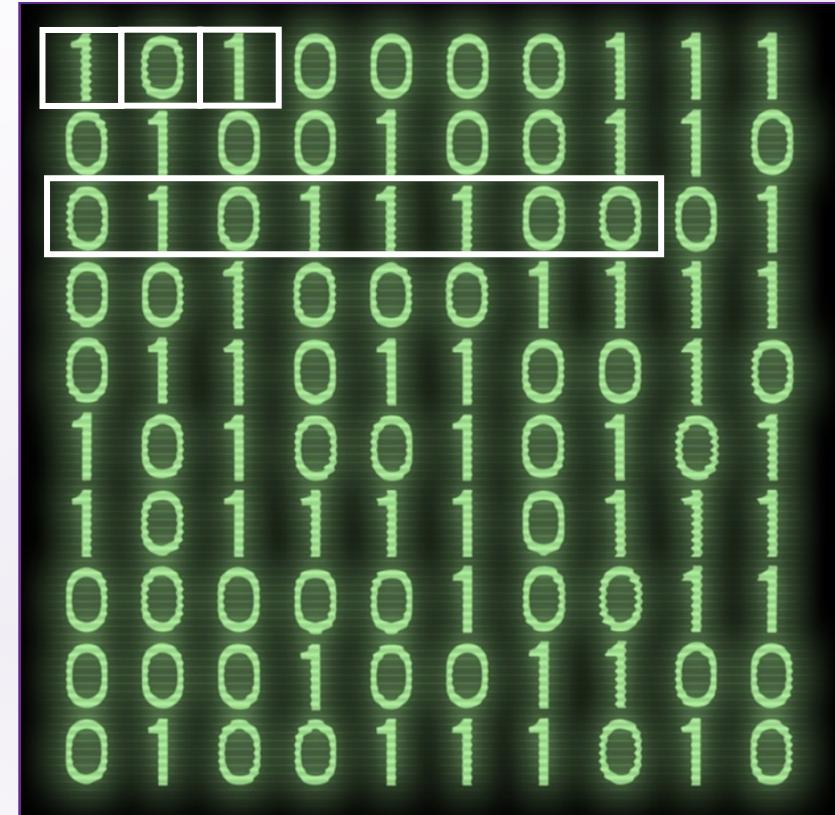
$2^{10} = 1024$ 'un katlari seklinde yapılandirilmistir.

1024 byte = 1 KB

1024 KB = 1 MB

1024 MB = 1 GB

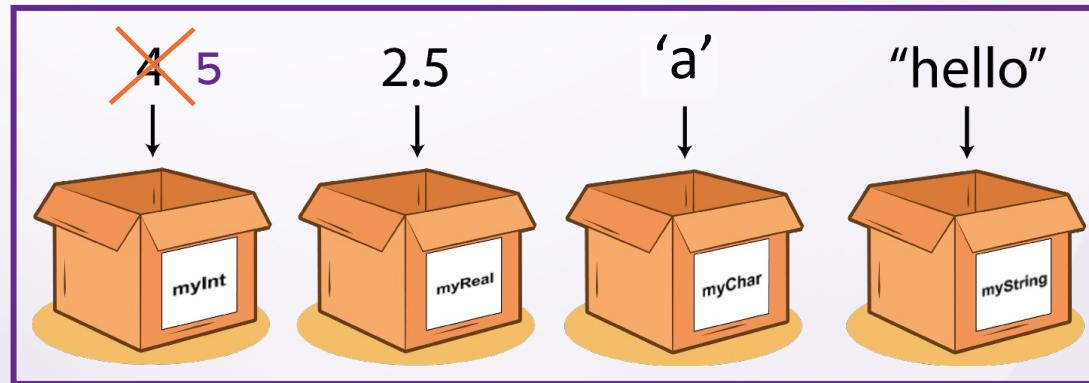
1024 GB = 1 TB ....



# Variables (Data Kullanma )

Java hafizadaki dataları **variable** veya **objeler** yardımıyla kullanır.

Bir datayı hafızada store etmek istedigimizde, Java hafızada o datayı tutabilecegi bir alan ayırır ve o alanı isimlendirir.



Biz ne zaman o data üzerinde degisiklik yapmak istesek, ismini söylememiz yeterli olur.

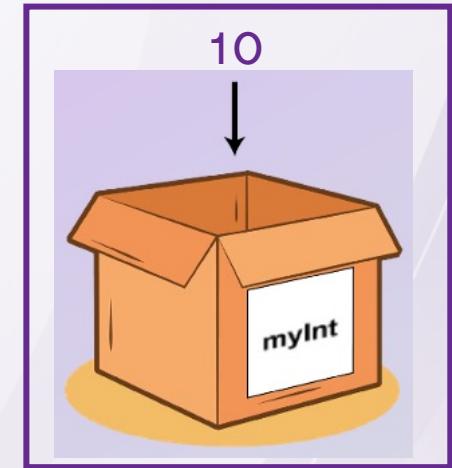
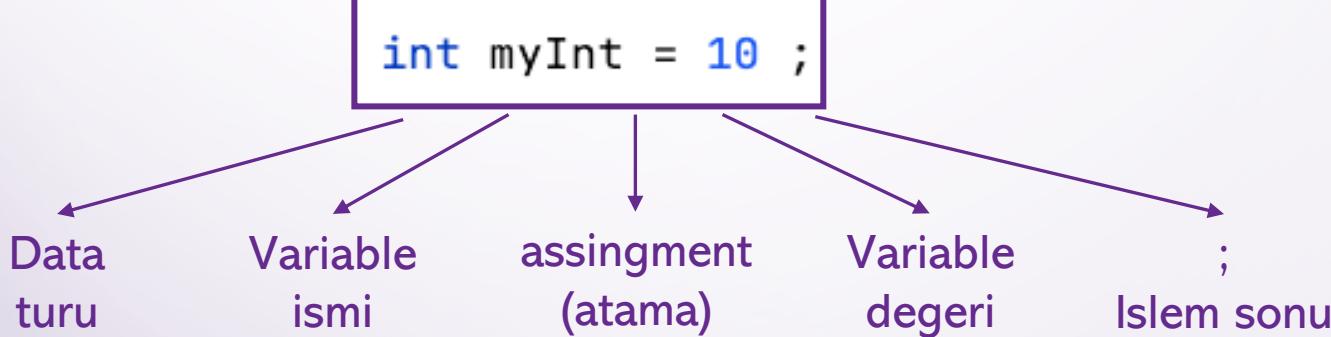
Ornegin myInt'i bir artır dedigimizde, Java myInt ismindeki variable'i bulur, icindeki değer olan 4'u bir artırıp 5 yapar.

Bu islemden sonra myInt variable'ının değeri 5 olacaktır.

Ozetle; biz bir değeri store etmek istedigimizde data turune uygun bir variable oluşturup ona bir isim veririz, ne zaman o datayı kullanmak istesek Java'ya variable ismini söylememiz yeterli olacaktır. myInt'i artır, myInt'i degistir, myInt'i sil vb...

# Variable Nasil Olusturulur ?

Variable olusturmak ve deger atamak icin Java'nin belirledigi syntax asagidaki gibidir.

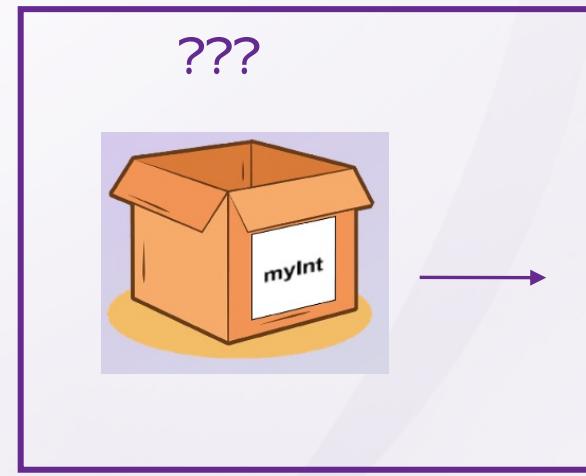
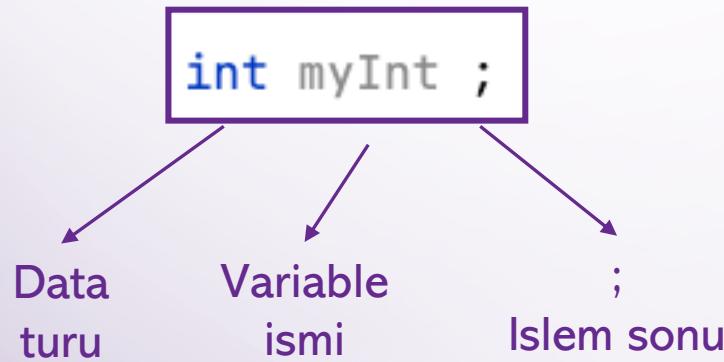


Bir variable olusturmak icin 2 islem vardir;

- 1- declaration (tanimlama) : esitligin sol tarafı
- 2- deger atama (assignment) : esitlik ve esitligin sag tarafı

# Variable Declaration

Java'nin datayı store edebilmesi için variable'a ihtiyacı vardır. Bir variable'in oluşturulması için de mutlaka declaration gereklidir.



Declaration için data turu ve variable ismi yeterlidir.

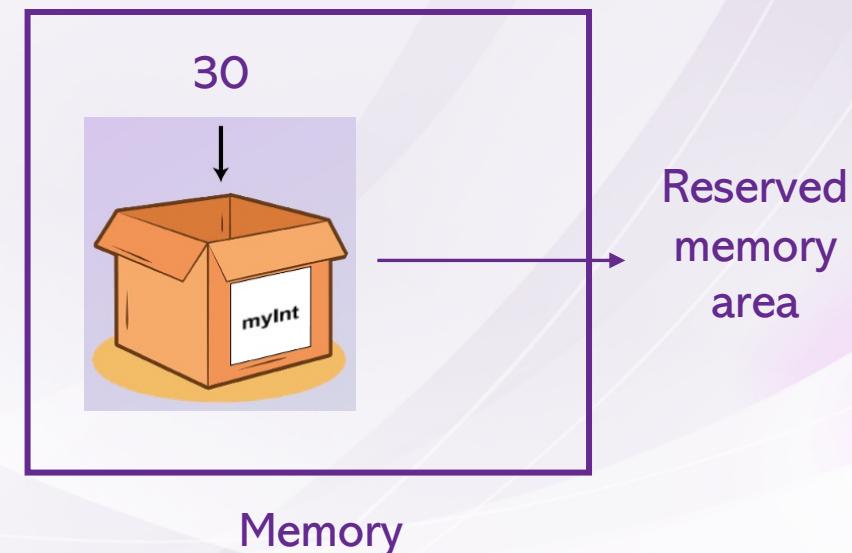
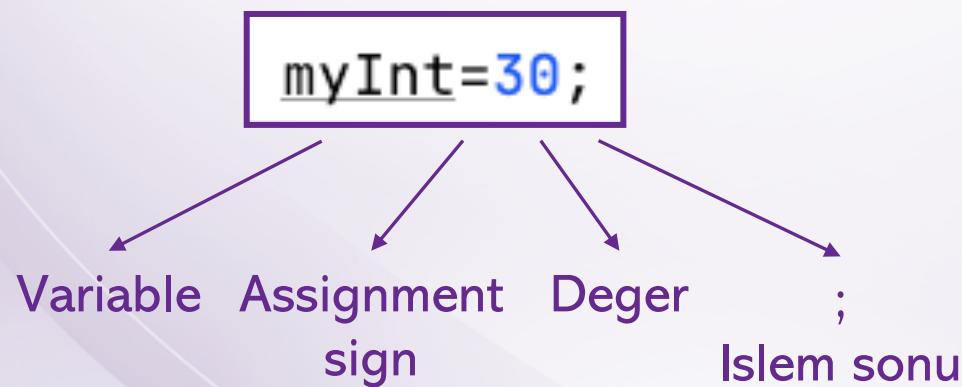
Java'da declaration ve assignment farklı satırlarda yapılabilir.

Ancak bir değer ataması olmadan variable'in kullanılması mümkün degildir.

# Variable Assignment

Deklare edilmiş bir variable'a değer atamaya assignment denir.

Declaration ve assignment farklı iki işlemidir. İkisi aynı satırda yapılabilcegi gibi, farklı satırlarda da yapılabilir.



# Variable Assignment

Declaration ve assignment asagidaki sekillerde yapilabilir.

1- Declaration ve assignment ayni satirda yapilabilir

```
int not=80 ;
String isim="John Doe";
boolean ogrenciMi=true;
double notOrt=89.3;
```

2- Once declaration, sonra assignment yapilabilir

```
int not;
not= 90;
not= (not + 80)/2;
```

NOT : Declaration sadece 1 kere yapilir, assignment ise istendigi kadar yapilabilir.



# Variable Assignment

3- Ayni data turundeki birden fazla variable ayni satirda deklare edilip, sonra tek tek assignment yapilabilir

```
int not1,not2,ortNot;  
  
not1= 80;  
not2= 90;  
ortNot= (not1 + not2)/2;
```

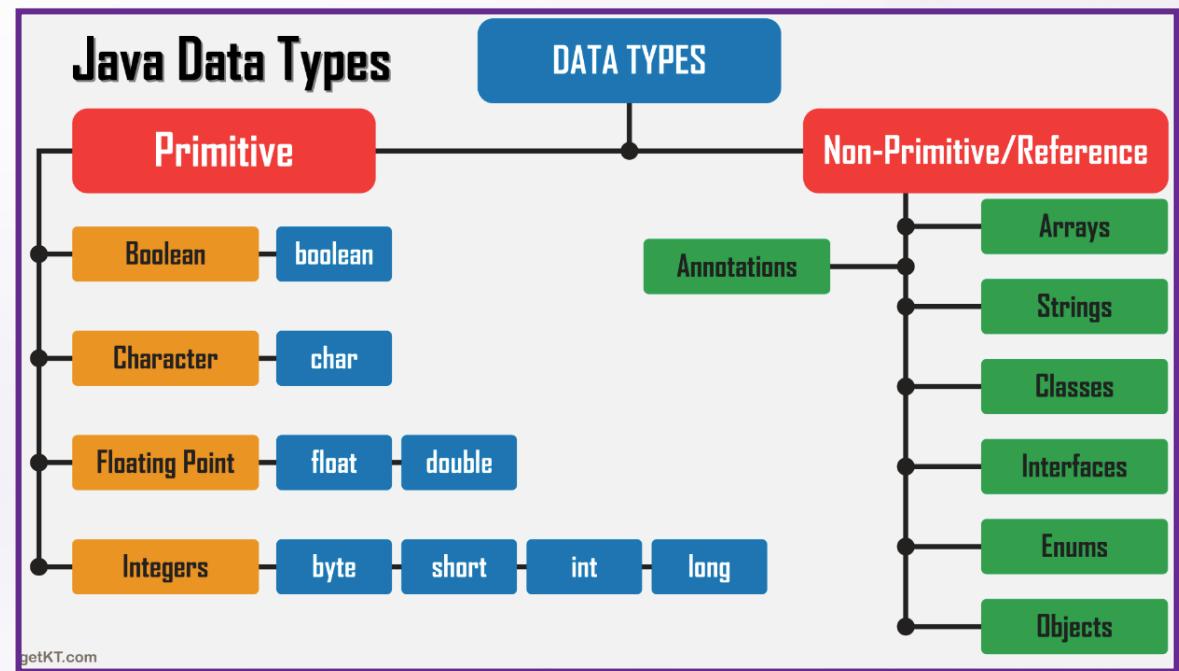
4- Ayni data turunde birden fazla variable tek declaration ile olusturulabilir.

```
int not1=80,not2=90,ortNot= (not1 + not2)/2;
```

# Data Turleri

Java'da temelde iki data turu kullanılır.

- 1- Primitive Data Turleri
- 2- Non-Primitive Data Turleri



Biz baslangicta primitive data turleri ve String kullanarak ilerleyecegiz, daha sonra diger non-primitive data turlerini ogrenecegiz.



# Primitive Data Turleri

Java'da 8 primitive data turu kullanılır.

Primitive data turleri sadece değer store edebilirler ve hafızada kapladıkları alan her data turu için sabittir.

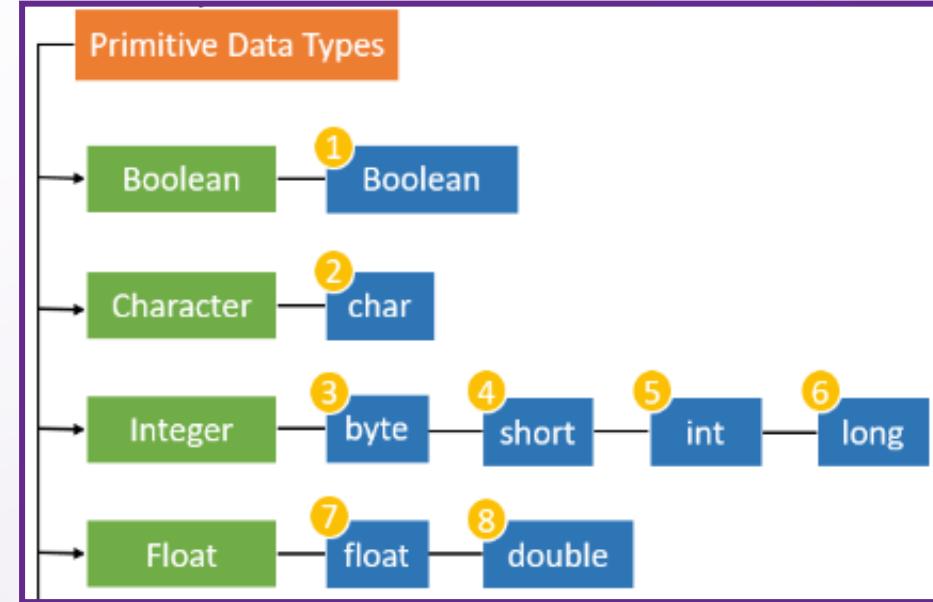
- 1- Boolean : Mantıksal data sonuçlarını store etmek için kullanılır.

boolean data turundeki bir variable sadece 2 değer barındırabilir, **true / false**

Bilgisayar true için 1, false için 0 değerini tutar, dolayısıyla hafızada sadece **1 bit** yer kaplar

- 2- char : Tek bir karakter barındırır. İçerisinde harf, sayı veya özel karakter olabilir. char data turunun en belirgin farklılığı 'c' (tek tırnak) kullanmasıdır. Bir data " kullanırsa char olmalıdır.  
hafızada **16 bit** yer kaplar

```
char harf='A',sayi= '4',karakter='#';
```



# Primitive Data Turleri

Tam sayı barındıran primitive data turleri

Data Turu	Hafiza Boyut	minimum deger	maximim deger
3- byte	8 bit	$-2^7 = -128$	$2^7 - 1 = 127$
4- short	16 bit	$-2^{15} = -32.768$	$2^{15} - 1 = 32.767$
5- int	32 bit	$-2^{31} = -2.147.483.648$	$2^{31} - 1 = 2.147.483.647$
6- long	64 bit	$-2^{63} = -9.223.372.036.854.755.808$	$2^{63} - 1 = 9.223.372.036.854.755.807$

Bir variable icin hangi data turunu kullanacagımız, uygulamamızın hafiza kullanımı için önemlidir.

Ornegin, üniversitedeki öğrencilerin yaslarını barındıran bir variable icin byte yeterlidir.

Yasları short, int veya long olarak da store edebiliriz ancak bu durumda kullanılabilecek hafıza miktarı katlanarak artacaktır.

# Primitive Data Turleri

Ondalikli sayi barindiran primitive data turleri

Data Turu	Hafiza Boyut	min-max deger	Ondalikli basamak sayisi
7- float	32 bit	$\pm 3.40282347E+38F$	6-7 basamak
8- double	64 bit	$\pm 1.79769313486231570E+308$	15-16 basamak

Ondalikli sayilar icin hafiza durumu ve ondalik kismin uzunluguna gore data turu secilebilir.

Deger atamasi yaptigimizda Java'nin float ile double'i ayirt edebilmesi icin, float sayilarin yaninda **f** veya **F** kullanmamiz gerekir.

```
float a=20f;
float b=6f;
System.out.println(a/b); // 3.3333333
```

```
double c=20;
double d=6;
System.out.println(c/d); // 3.333333333333335
```



# Non-Primitive Data Türleri

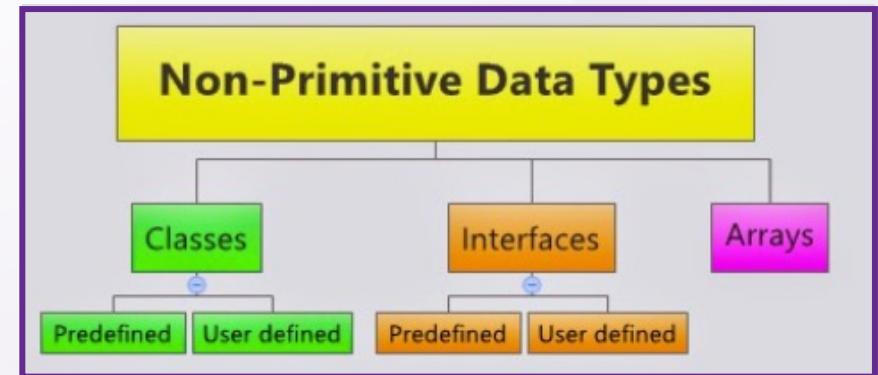
Primitive data türleri Java tarafından oluşturulmuştur ve biz yeni bir PRIMITIVE DATA TURU oluşturamayız.

Non-Primitive data türleri ise Java tarafından sınırlanılmamışlardır. Java da bazi non-primitive data türleri oluşturmuştur, biz de yeni non-primitive data türleri oluşturabiliriz.

Non-Primitive data türlerinin geneli için **object** tabiri kullanılabilir, cunku tüm non-primitive data türlerinin kendilerine ait bir class tarafından oluşturulan objelerdir.

Class'lar OOP konsept çerçevesinde bizim obje oluşturduğumuz, kalıplar olduğundan, non-primitive data türleri bu class'lardan oluşturulan objelerdir.

Class'lar method'lar da içerdiginden, non-primitive data türleri, oluşturuldukları class'lardaki method'lari kullanabilirler.





# Non-Primitive Data Turleri

Non-Primitive data turlerinden, simdilik String'i kullanacagiz. Ilterleyen derslerde Java'nin olusturdugu tum non-primitive data turlerini gorecegiz.

```
String isim= "John Doe";
```

String'i variable'lara deger atamak icin **""** kullaniriz.

```
String isim= "John Doe";
```

isim.|

- m **split**(String regex)
- m **toLowerCase**()
- m **substring**(int beginIndex)
- m **getBytes**(StandardCharsets.UTF\_8)
- m **getBytes**(String charsetName)
- m **getBytes**(Charset charset)
- m **getBytes**()
- m **getBytes**(int srcBegin, int srcEnd,
- m **toLowerCase**(Locale.ROOT)
- m **toLowerCase**(Locale locale)
- m **toUpperCase**(Locale.ROOT)
- m **toUpperCase**(Locale locale)
- m **toUpperCase**()
- m **split**(String regex, int limit)
- m **substring**(int beginIndex, int endIndex)
- m **charAt**(int index)

Non-primitive data turunde olusturulmus herhangi bir variable'in **ismini** yazip **.**'ya basarsaniz, o variable ile kullanabilecegimiz method'lari gorebilirsiniz.

# Primitives & Non-Primitives

İki data turu arasında 5 temel farklilik sayabiliriz.

Primitives	Non-Primitives
Tüm Java tarafından oluşturulmuştur.	Java tarafından oluşturulanlar olduğu gibi biz de oluşturabiliriz
Sadece değer icerirler, variable ile kullanılabilecek hazır method'ları yoktur.	İçerikleri değerin yanında oluşturdukları class'dan gelen hazır method'ları da barındırırlar.
Bir değer atamadan oluşturulabilir ama kullanabilmek için mutlaka değer atanmalıdır.	Değer atanmadan <b>null</b> olarak işaretlenebilirler.
Data turu isimleri <b>kucuk</b> harfle baslar (int, char vb.)	Data turu isimleri <b>buyuk</b> harfle baslar. (String vb..)
Primitive data turundeki variable'ların hafızada kapladıkları alan sabittir. Değeri küçük de olsa, büyük de olsa hafızada belirlenen miktarda alan ayrıılır.	Hafızada kapladıkları alan sabit degildir. Data turu ve içerdigi datanın büyüklüğine göre hafızada yer kaplarlar. (bir kelime veya binlerce kelime içeren String'lerin boyutları farklı olacaktır)

# Naming Convention (Isim Verme Kurallari)

Variable'lara isim verirken istedigimiz ismi secebiliriz ancak asagidaki kurallara uyulmasi gereklidir.

1- Variable isimleri buyuk-kucuk harf duyarlidir (**case sensitive**).

Not, NOT, not, nOT ... birbirinden farklidir.

2- Variable isimlerinde **harf**, **rakam**, **\_** ve **\$** kullanilabilir,  
bosluk veya \* gibi ozel karakterler kullanilamaz.

3- Variable isimleri harf ile baslamlidir, rakam ile baslayamaz.  
**\_** ve **\$** ile baslayabilir ama kullanilmasi tavsiye edilmez.

4- Variable isimleri olarak keyword (Java'da tanimli anahtar kelimeler) kullanilamaz.  
for, int, short, class vb. variable ismi olamaz.

5- Variable isimleri kucuk harfle baslar, birden fazla kelime iceriyorsa **camelCase** kullanilir, yani sonraki her kelimenin ilk harfi **buyuk harf**, diger harfleri **kucuk harf** yapilir.



Wise Quarter  
first class IT courses

# Java Team 127 Ders-02

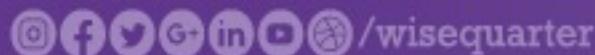
Scanner  
Data Casting



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





Wise Quarter  
first class IT courses

# Onceki Dersten Aklimizda Kalanlar

1-

# Hafiza(Memory) Kullanimi

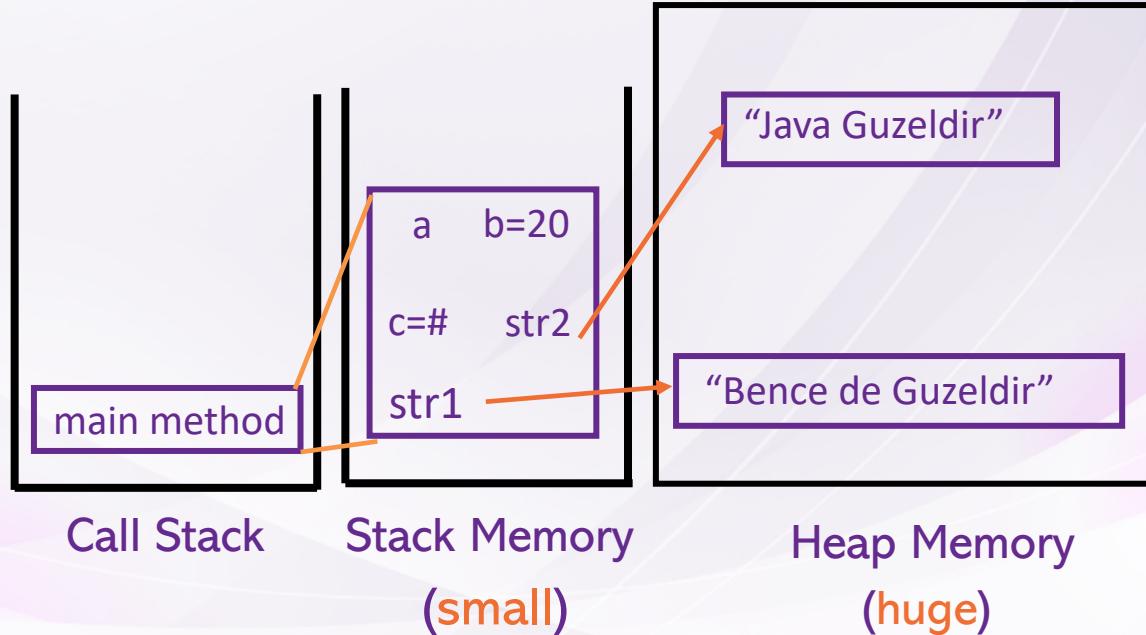


Java'da her method calistiginda, o methoda ait bir stack ve ona ait bir stack memory alani olusturulur.

Ayrica her method'un kullandigi heap memory vardir.

Stack memory'de primitive data turundeki variable'lar ve degerleri ile non-primitive data turundeki variable'larin referanslari olurken, heap memory'de non-primitive variable'larin degerleri store edilir.

```
public static void main(String[] args) {  
  
    int a;  
  
    int b=20;  
  
    char c= '#';  
  
    String str1;  
  
    String str2="Java Guzeldir";  
  
    str1="Bence de Guzeldir";  
  
}
```





# Kullanicidan Deger Alma (Scanner)

Kodlarimizi yazdigimiz IntelliJ veya benzeri ide'lere disardan bilgi almak icin Java Util kutuphanesinden Scanner Class'ina ihtiyacimiz vardir.

## 1. Adim

Scanner Class'inda var olan hazir method'lari kullanabilmek icin Scanner class'indan bir obje olusturmaliyiz.

```
Scanner scan= new Scanner(System.in);
```

## 2. Adim

Scanner calisinda kullanicidan bir bilgi bekleyecektir, kullanicinin kendisinden ne istendigini bilmesi icin bir aciklama yazdiralim.

```
System.out.println("Lutfen bir tamsayi giriniz");
```





# Kullanicidan Deger Alma (Scanner)

## 3. Adim

Kullanicinin girdigi degeri alabilmesi icin Scanner class'indan uygun method'u kullanalim.

Kullanicidan tamsayi istersek **nextInt()** kullanmamiz uygun olacaktir.

**nextInt()** bize kullanicinin girdigi tamsayiyi getirecektir, bunu programimizda kullanabilmek icin uygun data turundeki bir variable'a atayalim.

```
int girilensayi=scan.nextInt();
```

scan.nextInt()	
m next()	String
m next(String pattern)	String
m next(Pattern pattern)	String
m nextBigDecimal()	BigDecimal
m nextBoolean()	boolean
m nextBigInteger()	BigInteger
m nextBigInteger(int radix)	BigInteger
m nextByte()	byte
m nextByte(int radix)	byte
m nextDouble()	double
m nextFloat()	float
m nextInt()	int
m nextInt(int radix)	int
m nextLine()	String
m nextLong()	long
m nextLong(int radix)	long

Bu adimlar sonucunda kullanicinin girdigi deger girilenSayi variable'i olarak kodumuza eklenmis oldu.

# Sorular (Variables ve Scanner)

**Soru 1-** Kullanicidan uc farkli data turunde deger alip, girilen degerleri aciklamalariyla yazdirin.

**Soru 2-** Kullanicidan bir double, bir de int sayi alip bunların toplamini ve carpimini yazdirin.

**Soru 3-** Kullanicidan ismini, soyismini ve yasini alip, asagidaki formmatta yazdirin.

Isminiz : John

Soyisminiz : Doe

Yasiniz : 44

Kaydiniz basariyla tamamlanmistir.

**Soru 4-** Kullanicidan bir dikdortgenin 2 kenar uzunlugunu alip, dikdortgenin alanini yazdirin.

**Soru 5-** Kullanicidan ismini, soyismini ve yasini alip asagidaki formmatta yazdirin.

girilen bilgiler : J Doe, 44

**Soru 6-** Kullanicidan bir cemberin yaricapini alip, cevresini ve alanini yazdirin.

**Soru 7 (Interview)-** Kullanicidan iki sayı alip ikisinin degerlerini degistirin(swap).

**Soru 8 (Interview)-** Kullanicidan iki sayı alip, ucuncu bir degisken kullanmadan ikisinin degerlerini degistirin(swap).



# Socrative Quiz

1) <https://b.socrative.com/login/student/>  
adresine gidin

(google'da Socrative student aranınca cıktı)

2) Room Name **BULUTLUOZ** yazın

3) Isminizi yazın

4) Done butonuna basin

Sure : 15 Dakika

# Data Casting (Data Turunu Degistirme)



Java'da bir data turundeki datayı başka bir data turune cevirmeye **data casting** denir. Ancak her data turu birbirine cevrilemez.

Benzer özelliklerdeki data turundekidataları birbirine kolayca cevirebilirken, bazi casting işlemleri için ekstra kod yazmamız gereklidir, bazi casting işlemleri ise imkansızdır.

```
int sayı= "John Doe";  
String str= false;
```

```
String isim="John Doe";  
int sayı= isim;  
  
boolean dogruMu=false;  
String str= dogruMu;
```

```
double dbl=23.4;  
int sayı= dbl;  
  
int in= 12;  
double db= in;
```

Java'da bir kodun altı kırmızı çizili oluyorsa, orada java'nın çözemediği bir sorun vardır ve siz o sorunu çözmedikçe Java çalışmazacaktır. (Sadece o class değil diğer class'lar da çalışmaz)

Java'da hem primitive, hem de non-primitive data turleri için data casting yapmak mümkün ancak biz simdilik primitive data turleri için data casting konusunu irdeleyelim.

# Implicit Data Casting (Auto-Widening)

Daha kucuk kapsamlı bir data turundeki degeri, daha genis kapsamlı data turundeki variable'a atama yapmak istedigimizde, Java bu islemi otomatik olarak yapacaktir



```
byte a = 12;  
  
int b = a;  
  
double c = b;
```



# Explicit Data Casting

Daha genis kapasiteye sahip bir data turundeki bir degeri, daha dar kapsamli bir variable'a atamak istedigimizde, Java bunu otomatik olarak yapmayacaktir.



```
int a = 12;  
int c = 567;  
  
byte b = a;  
byte d = c;
```

```
int a = 12;  
int c = 567;  
  
byte b = (byte) a; // 12  
byte d = (byte) c; // 55
```

Atanan deger'in data turu genis kapsamli oldugundan deger dar kapsamli variable'in sinirlari icinde olabilecegi gibi, sinirlarindan buyuk de olabilir.

Bu durumda Java, data kaybi veya degisikligi ihtimalinin farkinda oldugumuzun bilmek ister.

Sorumluluğu almak icin cast etmek istedigimiz degerin onune (cast etmek istedigimiz data turunu) yazarsak, java bu casting'i data turu sinirlarina gore yapar.



# Char Data Turu ve ASCII Table

Char data turu sadece 1 karakter icerir, ancak degerlerin ascii degerlerini tuttugundan, matematiksel islemlerde ascii kodlarina gore islemlere dahil olur.

ASCII control characters		ASCII printable characters								Extended ASCII characters							
00	NULL (Null character)	32	space	64	@	96	`	128	Ç	160	á	192	ł	224	ó		
01	SOH (Start of Header)	33	!	65	A	97	a	129	Ü	161	í	193	ł	225	ó		
02	STX (Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	ł	226	ó		
03	ETX (End of Text)	35	#	67	C	99	c	131	â	163	ú	195	ł	227	ó		
04	EOT (End of Trans.)	36	\$	68	D	100	d	132	ä	164	ñ	196	—	228	ö		
05	ENQ (Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	+	229	ö		
06	ACK (Acknowledgement)	38	&	70	F	102	f	134	â	166	º	198	å	230	µ		
07	BEL (Bell)	39	'	71	G	103	g	135	ç	167	º	199	À	231	þ		
08	BS (Backspace)	40	(	72	H	104	h	136	ê	168	¸	200	Ł	232	Þ		
09	HT (Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	Ł	233	Ú		
10	LF (Line feed)	42	*	74	J	106	j	138	è	170	¬	202	Ł	234	Ó		
11	VT (Vertical Tab)	43	+	75	K	107	k	139	í	171	½	203	Ł	235	Ù		
12	FF (Form feed)	44	,	76	L	108	l	140	î	172	¼	204	Ł	236	Ý		
13	CR (Carriage return)	45	-	77	M	109	m	141	í	173	ı	205	=	237	Ý		
14	SO (Shift Out)	46	.	78	N	110	n	142	Ã	174	«	206	‡	238	—		
15	SI (Shift In)	47	/	79	O	111	o	143	Ã	175	»	207	¤	239	·		
16	DLE (Data link escape)	48	0	80	P	112	p	144	É	176	„	208	ð	240	≡		
17	DC1 (Device control 1)	49	1	81	Q	113	q	145	æ	177	„	209	ð	241	±		
18	DC2 (Device control 2)	50	2	82	R	114	r	146	Æ	178	„	210	È	242	–		
19	DC3 (Device control 3)	51	3	83	S	115	s	147	ô	179	—	211	È	243	¾		
20	DC4 (Device control 4)	52	4	84	T	116	t	148	ö	180	—	212	È	244	¶		
21	NAK (Negative acknowl.)	53	5	85	U	117	u	149	ò	181	À	213	ı	245	§		
22	SYN (Synchronous idle)	54	6	86	V	118	v	150	û	182	Ã	214	ı	246	+		
23	ETB (End of trans. block)	55	7	87	W	119	w	151	ú	183	Ã	215	ı	247	·		
24	CAN (Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	ı	248	°		
25	EM (End of medium)	57	9	89	Y	121	y	153	Ó	185	—	217	ı	249	..		
26	SUB (Substitute)	58	:	90	Z	122	z	154	Ü	186	—	218	—	250	·		
27	ESC (Escape)	59	;	91	[	123	{	155	ø	187	—	219	—	251	ı		
28	FS (File separator)	60	<	92	\	124		156	£	188	—	220	—	252	³		
29	GS (Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	—	253	²		
30	RS (Record separator)	62	>	94	^	126	~	158	×	190	¥	222	—	254	■		
31	US (Unit separator)	63	?	95	—			159	f	191	—	223	—	255	nbsp		
127	DEL (Delete)																

```
char harf= 'a';
int sayi= 100;

System.out.println( harf + sayi); // 197
System.out.println( harf+1); // 98

char yениharf=(char)(harf+1);
System.out.println(yениharf); // b
```

# Sorular (Data Casting)

**Soru 1-** Int olarak verilen 3 degerin ortalamasini double olarak yazdiran bir kod yazin

**Soru 2-** Kullanicidan bir harf alin ve alfabebe o harften sonraki 3 harfi yazdirin.

**Soru 3-** Kullanicidan bir sayi alin, kullanici kac degerini girerse girsin, o sayiyi -128 ile 127 arasindaki bir sayiya donusturup yazdirin.

**Soru 4-** Kullanicidan iki double sayi alin, ilk sayiyi ikinci sayiya bolun ve bolum isleminin sonucununun tamsayi kismini yazdirin.

**Soru 5-** Kullanicidan bir double, bir tamsayi alin, double sayiyi ikinci sayiya bolun ve bolum isleminin sonucununun tamsayi kismini yazdirin.



# Wrapper Classes

Java primitive data turleri, kod yazarken mutlaka kullanacagimiz data turleridir. Ancak primitive data turleri sadece deger tasiyabilirler, **class olmadiklari icin** hazir method'lara sahip degillerdir.

Wrapper class'lar primitive data turlerini iceren **class'lardir**. Bu class'lardan olusturulan objeler primitive data turleri ile kullanabilirler.

```
int sayi=10;  
Integer sayiW= 20;  
  
sayiW=sayi;  
sayi= sayiW+5;
```

Wrapper class'lardan objelere primitive data turundeki degerler atanabilir. Ayrca bu class'lar bir çok faydalı method bulundururlar.

Integer.

max(int a, int b)	int
MAX_VALUE ( = 0x7fffffff)	int
bitCount(int i)	int
getInteger(String nm)	Integer
getInteger(String nm, int val)	Integer
compare(int x, int y)	int
compareUnsigned(int x, int y)	int
decode(String nm)	Integer
divideUnsigned(int dividend, int divisor)	int
getInteger(String nm, Integer val)	Integer



# Wrapper Classes

Wrapper class'lar casting, max-min degerler, karsilastirma gibi bircok hazır method'lara sahiptirler.

```
int sayi=10;
Integer sayiW= 20;
System.out.println(Integer.MAX_VALUE); // 2147483647
System.out.println(Integer.max( a: 34, b: 465)); // 465

boolean kontrol=true;
Boolean kont=false;
String knt="false";
boolean sonuc = Boolean.valueOf(knt);

char chr='*';
Character ch='p';
char chr2=101;
System.out.println(Character.valueOf(chr2)); // e
System.out.println(Character.isDigit( ch: '5')); // true
System.out.println(Character.isAlphabetic( codePoint: '9')); // false
System.out.println(Character.isAlphabetic( codePoint: 'a')); //true
```

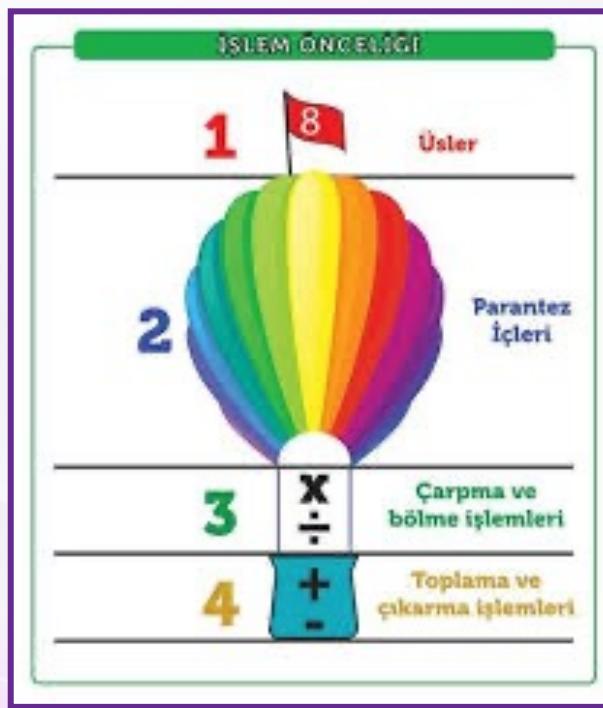


# Java'da Matematiksel İşlemler

Java Matematik işlemlerini sorunsuz yapar, Ancak biz işlemleri yazarken matematik kurallarına uygun olarak yazmazsa ummadığımız sonuçlarla karşılaşabiliriz.

$$24 + ( 5 * 2^3 - 3^3 )^2 - 13$$

$$14 - 5 * 2 + 3 * 4 - 8$$



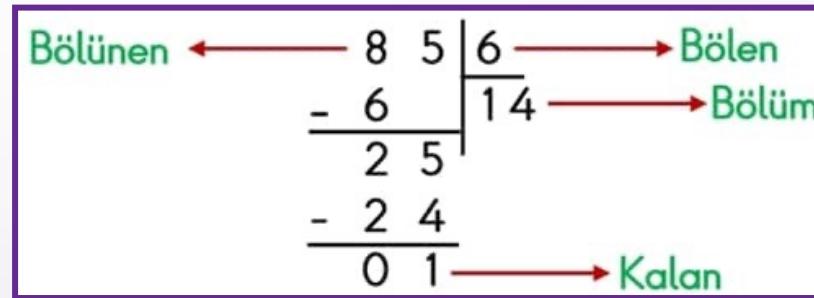
$$24 / 6 * 2 - 7 * 4 + 9$$

$$8 * 5 + 2 * ( 12 / 4 ) - 19$$



# Modulus (% Kalan Bulma)

Java'da Modulus işlemi, bir bolme işlemindeki kalan sayiyi bize verir.



Modulus işlemi sayesinde

- Cift sayilar ( sayı %2 )
- Bir sayinin birler basamagini bulma ( sayı %10 )
- Bir sayı (ornegin 5) ile tam bolunebilen sayıları bulma ( sayı % 5 )

mumkun olmaktadır.

# Modulus Soru

Soru 1- Kullanicidan 4 basamakli pozitif bir tamsayi alip rakamlar toplamini bulalim

**Ipucu 1:** Sayi % 10 => Bize son basamagi verir

$$1469 \% 10 = 9$$

**Ipucu 2:** Int Sayi /10 => Bize son basamak haric sayiyi verir

`int sayi=1469;`

`sayi = sayi / 10 =>`

`sayi'ya 46 degerini atar`



# Increment (Deger Artirma)

Toplama veya carpma yaparak bir variable'in degerini artirabiliriz.

Increment isleminin kalici olmasi icin 3 farkli sekilde assignment yapilabilir.

```
int sayi = 10 ;
```

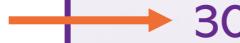
```
sayi= sayi +3 ;
```



13

```
int sayi = 10 ;
```

```
sayi *= 3 ;
```



30

```
int sayi = 10 ;
```

```
sayi++ ;
```



11



# Decrement (Deger Azaltma)

Cikarma veya bolme yaparak bir variable'in degerini azaltabiliriz.

Decrement isleminin kalici olmasi icin 3 farkli sekilde assignment yapilabilir.

```
int sayi = 10 ;  
  
sayi = sayi - 2 ; → 8
```

```
int sayi = 10 ;  
  
sayi -= 4 ; → 6
```

```
int sayi = 10 ;  
  
sayi -- ; → 9
```



# Dunku Dersten Aklimizda Kalanlar

- 1- Data Casting : Java'da bir data turune ait variable'a baska data turune ait deger atanmasina denir.
- 2- Java'da her data turu baska data turune ATANAMAZ
- 3- Bazi data turleri hic bir data turunden deger kabul etmez,
- 4- Bazi data turleri arasında otomatik olarak java casting yapar, bazi data turleri ise casting yapmak icin bizim onayimizi bekler.
- 5- Benzer data turundeki variable ve degerler arasında casting yapmak istedigimizde
  - daha kapsamlı data turundeki variable'a, daha dusuk kapsamlı data turunden deger atamasi yaparsak, Java Auto Widening yapar.
  - daha kucuk kapsamlı data turundeki variable'a, daha genis kapsamdaki data turunden bir deger atamak istersek, Java bunu otomatik olarak yapmaz, bizden manuel onay ister. BU onayı yapmak icin degerin onune (cast yapılacak data turu) yazılır. Explicit Narrowing
- 6- Explicit Narrowing yaptigimizda
  - deger, variable'in data turu sinirlari icerisindeyse direk atama yapar
  - deger, variable'in data turunun sinirlarindan buyukse, data kayiplari veya data degisiklikleri olabilir



Wise Quarter  
first class IT courses

# Java

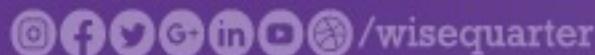
## Team 120 Ders-04

### Data Casting Wrapper Classes Increment – Decrement



+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





Wise Quarter  
first class IT courses

# Java

## Team 120 Ders-05

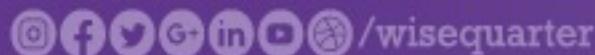
Matematiksel İşlemler  
Increment – Decrement



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





Wise Quarter  
first class IT courses

# Java

## Team 120 Ders-06

### Increment – Decrement Concatenation Operatorler



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Pre – Post Increment

```
int sayi = 10 ;  
  
sayi++ ;
```

```
int sayi = 10 ;  
  
sayi -- ;
```

Sayı değerini kalıcı olarak 1 artırırken **sayi++**, 1 azaltırken **sayi--** kullanabileceğimizi gormustuk.

**++** ve **--** sayidan önce de kullanılabilir, sonuc yine aynı olacaktır.

```
int sayi = 10 ;  
  
++sayi ;
```

```
int sayi = 10 ;  
  
--sayi ;
```

Pre-Increment veya Pre-Decrement ile Post-Increment ve Post-Decrement arasındaki fark, bu işlem farklı bir işlem ile birlikte kullanılırsa ortaya çıkar.

# Pre – Post Increment

Post-Increment'te, increment diger islemden **sonra** yapilir .

```
int sayi = 10 ;  
  
System.out.println(sayi++); ; → Once sayiyi yazdirir (10) , sonra degeri artirir (11)  
  
System.out.println(sayi); → Ust satirda deger (11) oldu, (11) yazdirir.
```

Pre-Increment'te, increment diger islemden **once** yapilir .

```
int sayi = 10 ;  
  
System.out.println(++sayi); ; → Once sayiyi artirir (11) , sonra yazdirir (11)  
  
System.out.println(sayi); → Ust satirda deger (11) oldu, (11) yazdirir.
```

# Pre – Post Increment

Post-Increment'te, increment diger islemden **sonra** yapilir .

```
int a = 10 ;  
int b= a++;  
  
System.out.println(a);  
System.out.println(b);
```

Once b'ye atama yapar ( $b=10$ ) ,  
sonra a degerini artirir ( $a=11$ )  
(11)  
(10)

Pre-Increment'te, increment diger islemden **once** yapilir .

```
int a = 10 ;  
int b= ++a;  
  
System.out.println(a);  
System.out.println(b);
```

Once artirma yapar ( $a=11$ ) ,  
sonra b'ye atama yapar( $b=11$ )  
(11)  
(11)



# Pre – Post Increment Soru

Soru : Asagidaki kod calistirilirsa konsolda gorunecek sonuclar neler olur?

```
int a=10;  
  
System.out.println("a'nin degeri : " + ++a);  
  
int b= a++;  
  
System.out.println("b'nin degeri : " + b);  
  
int c= b++ + a ;  
  
System.out.println("c'nin degeri : " + c);  
  
System.out.println("Son toplam : " +(a+b+c));
```



# Concatenation (String Birlestirme )

Bir String'i, baska bir String veya primitive deger ile + isareti kullanarak isleme sokarsak Java bu degiskenleri birlestirerek yeni bir String olusturur

```
String a = "Hello";
String b = "World";
System.out.println(a+b);
System.out.println(a+" "+b);
```

HelloWorld  
Hello World

**Not :** Eger matematiksel bir islemin icinde String kullanilirsa, matematikteki oncelikler dikkate alınarak islem yapılır. Sira String ile toplamaya geldiginde toplama yerine Concatenation uygulanır

```
String a = "Hello";
int b = 2;
int c = 3;
```

```
System.out.println(a+b+c);
```

Hello23

```
System.out.println(c+b+a);
```

5Hello

```
System.out.println(a+(b+c));
```

Hello5

```
System.out.println(a+b*c);
```

Hello6



# Concatenation (String Birlestirme )

Soru : Sadece verilen variable'lari kullanarak istenen String'leri elde edelim.

```
String s1= "Java";
String s2= " ";
String s3= "kolay";
String s4= "";

int a= 3;
int b= 4;
```

12 Java kolay  
7 Java kolay  
34Java kolay  
Java12kolay  
Java34kolay  
Java7kolay



# Relational (Karsilastirma) Operators

## 1- Esitlik (Cift esitlik isareti) : ==

Java'da, matematikten farkli olarak = isareti assignment(atama) islemi yapar, esitligi kontrol etmez

Java'da, iki degerin esit olup olmadigini kontrol etmek icin == kullanilir ve sonuc olarak bize true veya false doner.

```
int a=10;
int b=15;

System.out.println(a==b);

System.out.println(a==b-5);

boolean c;

System.out.println(c=15==b);

c= 15*a==10*b;

System.out.println(c);
```



# Relational (Karsilastirma) Operators

## 2- Esit Degildir : !=

Java'da, herhangi bir mantiksal degerin basina konulan !=, o mantiksal ifadenin degerini tersine cevirir.

!true → false

!(5==5) → false

5 !=5 → false

```
int a=10;
int b=15;

System.out.println(a!=b);

System.out.println(a!=b-5);

boolean c;

System.out.println(c=15!=b);

c= 15*a !=10*b;

System.out.println(c);
```



# Logical (Mantıksal) Operators

## 1- And (ve) Operatoru `&&` , `&`

Mantıktaki AND operatorunun Java'da 2 tane karsılığı vardır. İşlevleri aynı olmakla birlikte iç işleyisi ve hız açısından aralarında küçük bir fark vardır.

`&&` operatoru birlestirdiği 2 boolean ifadenin ikisi de true ise sonucu true yapar, bunun dışındaki tüm durumlarda sonucu false yapar. (`&&` operatoru mükemmeliyetcidir.)

```
int a=10;
int b=15;

System.out.println(a>b  && b>0);

System.out.println(a<=b-5 && a>b-8);

boolean c;

System.out.println(c=15>=b  && a<0);

c= a>=b && 3*a<4*b;

System.out.println(c);
```

`&&` operatoru carpmaya benzetilir.  
`Sonucun 1 olması için`  
tüm çarpılan sayılar `1` olmalıdır.  
`1` tane bile sıfır olsa sonuc `0` olur.`out`

```
1 * 1 * 1 * 1 = 1
1 * 0 * 1 * 0 = 0
1 * 0 * 0 * 0 = 0
0 * 1 * 1 * 1 = 0
```

# Logical (Mantıksal) Operators

## && ile &' in farki

**&&** operatoru birbirine bagli mantıksal ifadeleri incelerken, **ilk false** degeri ile karsilastiginda, sonucun **false** olacagini algilar ve geriye kalan mantıksal ifadeleri incelemeden hemen sonucu **false** olarak atar.

**&** operatoru ise birbirine bagli mantıksal ifadeleri incelerken, herbir mantıksal ifadenin sonucuna gore karar vermez, islemin sonucuna kadar gider. Tum islem bittikten sonra sonuca atama yapar.

**&&** Operatoru tum mantıksal ifadeleri kontrol etmeden sonuca gidebildigi icin daha hizlidir.

```
int a=10;
int b=15;

System.out.println(a<b && b<10 && b>=a && a<0);

System.out.println(a<b & b<10 & b>=a & a<0);
```



# Logical (Mantıksal) Operators

## 2- OR (veya) Operatoru ||

Mantıktaki OR operatoru ile aynı şekilde kullanılır.

|| operatoru birlestirdiği 2 boolean ifadenin ikisi de false ise sonucu false yapar, bunun disindaki tum durumlarda sonucu true yapar. (OR operatoru iyimserdir.)

```
int a=10;
int b=15;

System.out.println(a>b || b>0);

System.out.println(a<=b-5 || a>b-8);

boolean c;

System.out.println(c=15>=b || a<0);

c= a>=b || 3*a<4*b;

System.out.println(c);
```

|| operatoru toplamaya benzetilir.  
*Sonucun 0 olması için*  
tüm toplanan sayılar 0 olmalıdır.  
1 tane bile bir olsa sonuc 0 olmaz

```
1 + 1 + 1 + 1 != 0
1 + 0 + 0 + 0 != 0
0 + 0 + 0 + 0 == 0
0 + 1 + 1 + 1 != 0
```



Wise Quarter  
first class IT courses

# Java

## Team 120 Ders-07

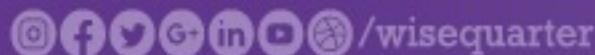
If Statements  
If Else Statements



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# If Statements

If Statements Java'da kod yazarken mutlaka ihtiyac duyacagimiz temel kod bloklarindan biridir.

Gunluk dilimizde oldugu gibi, bir şart ve ona bagli bir sonucu ifade eder.

Eger **iyi calisirsan**, **sinavi gecersin** **Calismazsan sonucu bilmiyoruz.**

Sart cumlesi belirtec

**boolean şart**

Sart saglanirsa sonuc

Kod alt satira gecer

```
int a= 10;  
int b= 20;  
if (a>b){ System.out.println("a b'den buyuk");}
```



# If Statements

Basit if cumleleri, kod'un geriye kalani ile baglantili degildir.

Belirlenen boolean sarti kontrol eder, o sart saglanirsa **if body** calisir, sart saglanmazsa **if body** calismaz

```
int a= 10;
int b= 20;

if (a>b){
    System.out.println("a b'den buyuk");
}

if (a<100){
    System.out.println("a 100'den kucuk");
}

if (b>0){
    System.out.println("b 0'dan buyuk");
}
```



# If Statements

If cumlesindeki boolean şart daha onceden de tanımlanabilir.

```
int a= 10;
int b= 20;

boolean sonuc=a>b;
if (sonuc){
    System.out.println("a b'den buyuk");
}

sonuc= a<100;
if (sonuc){
    System.out.println("a 100'den kucuk");
}

sonuc= b>0;
if (sonuc){
    System.out.println("b 0'dan buyuk");
}
```



# If Statements Sorular

**Soru 1-** Kullanicidan bir sayi isteyin, sayiyi kontrol edip 5 ile bolunebiliyorsa  
“Sayi 5'in tam kati” yazdirin.

**Soru 2-** Kullanicidan bir harf alin, harf ile baslayan bir ay varsa yazdirin.  
NOT: Buyuk harf, kucuk harf hassasiyeti olmasin.  
Kullanici o veya O yazdiginda output Ocak olsun.

**Soru 3-** Kullanicidan bir sayi alin, sayi 3 ile bolunuyorsa ”Uc ile bolunebilen  
sayi”, 5 ile bolunebiliyorsa “Bes ile bolunebilen sayi” yazdirin.

**Soru 4-** Kullanicidan bir ucgenin 3 kenar uzunlugunu alin, ucgen eskenar ise  
“Eskenar ucgen” yazdirin.

**Soru 5-** Kullanicidan notunu alin 50 veya daha buyukse ”Sinifi Gectin”, 50'den  
kucukse “Maalesef kaldin” yazdirin.



# If Else Statements

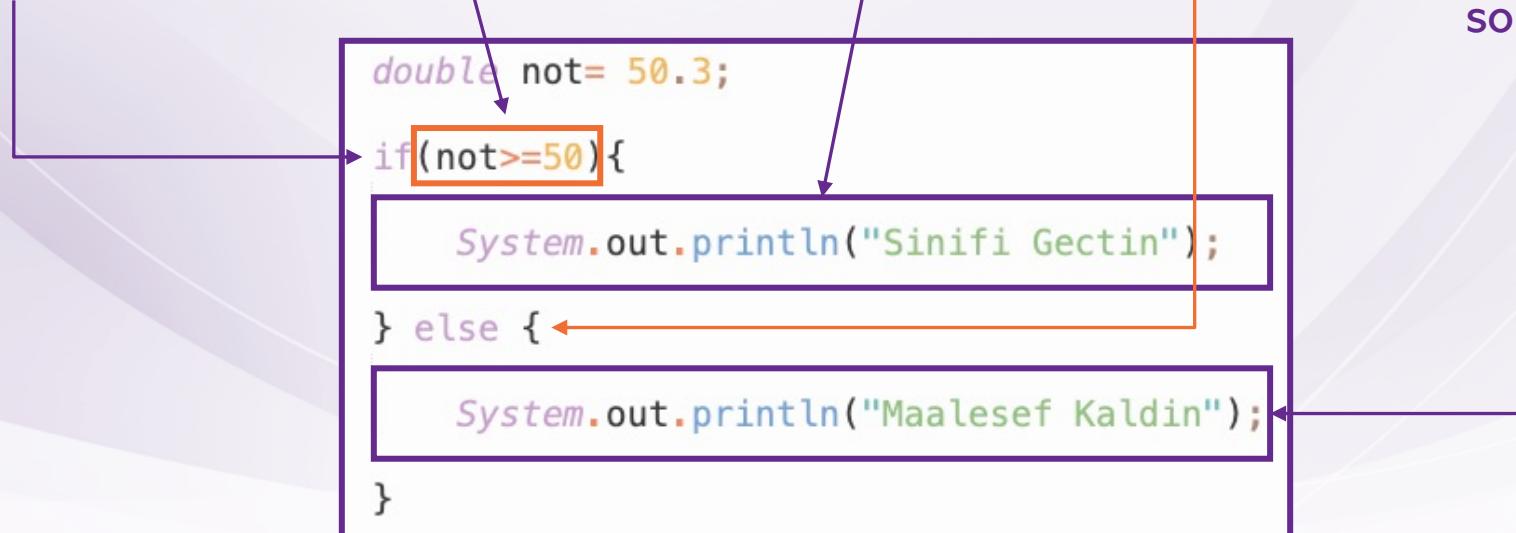
Basit if cümleleri kodun geri kalani ile ilgilenmiyor.

Sorularda şartın sağlanması veya sağlanmaması durumunda yapılacaklar belli ise basit if cümlesi yeterli olmayacağından.

Eğer sayı çiftse, "Çift Sayı" yazdır yoksa, "Tek Sayı" yazdır

Şart cümlesi belirteç boolean şart Sart sağlanırsa sonuc belirteç

Sart sağlanmazsa sonuc



# If Else Statements Sorular

**Soru 1-** Kullanicidan bir ucgenin 3 kenar uzunlugunu alin, ucgen eskenar ise “Eskenar ucgen” yazdirin, degilse “Eskenar degil” yazdirin.

**Soru 2-** Kullanicidan notunu alin 50 veya daha buyukse ”Sinifi Gectin”, 50’den kucukse ”Maalesef kaldin” yazdirin.

**Soru 3-** Kullanicidan yasini isteyin, 65 yas ve uzeri ise ”Emekli olabilirsin” yazdirin, yoksa emekli olmasi icin calismasi gereken yil sayisini yazdirin.

**Soru 4-** Kullanicidan bir karakter girmesini isteyin, girilen karakterin buyuk harf olup olmadigini yazdirin.

**Soru 5-** Kullanicidan bir harf isteyin, girilen karakter kucuk harf ise onu buyuk harf olarak yazdirin, yoksa girilen harfi yazdirin



Wise Quarter  
first class IT courses

# Java Team 120 Ders-08

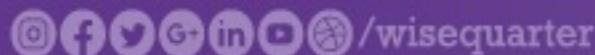
If Else Statements  
Nested If Else Statements



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# If Else If ... Statements

Bazen karsilastigimiz bir durumda secenek sayisi 2'den fazla olur. Bu durumda bir tane if else cumlesi sorunu cozmez.

Ornek :

Ogrencinin notu 85 ve ustu ise AA,

(85 ve ustu degilse) 65 ve ustu ise BB,

(65 ve ustu de degilse) 50 ve ustu ise CC,

(geriye kalanlar) DD

```
double not= 50.3;  
  
if(not>=85){  
    System.out.println("Notunuz AA");  
}  
else if(not>=65){  
    System.out.println("Notunuz BB");  
}  
else if(not>=50) {  
    System.out.println("Notunuz CC");  
}  
else {  
    System.out.println("Notunuz DD");  
}
```



# If Else Statements Sorular

- Soru 1-** Kullanicidan cinsiyetini ve yasini alin, Kadın, 60 yas ve uzeri , Erkek 65 yas ve uzeri emekli olabilir. Cinsiyet ve yasini dikkate alarak “Emekli olabilirsın” veya “Emekli olmak icin .. Yil daha calisman gerekir” yazdirin.
- Soru 2-** Kullanicinin kilo(kg) ve boyunu(cm) isteyip vucut kitle endeksini hesaplayin ( $kilo * 10000 / (boy * boy)$ ) vucut kitle endeksi 30'dan buyukse obez, 25-30 arasi ise kilolu, 20-25 arasi ise normal, 20'den kucukse zayıf yazdirin.
- Soru 3-** Kullanicidan aldiği ürün adedi ve ve liste fiyatını alın, kullanıcıya musteri kartı olup olmadığını sorun. Musteri kartı varsa 10 urunden fazla alırsa %20, yoksa %15 indirim yapın, Musteri kartı yoksa 10 urunden fazla alırsa %15, yoksa %10 indirim yapın
- Soru 4-** Kullanicidan mesafeyi kilometre olarak alın ve cevirmek istedigi birimi sorun, istedigi birim metre veya santimetre ise cevirip yazdirin, yoksa “istediginiz birim sisteme kayitli degil” yazdirin.

# If Else Statements

## Soru ) Interview Question

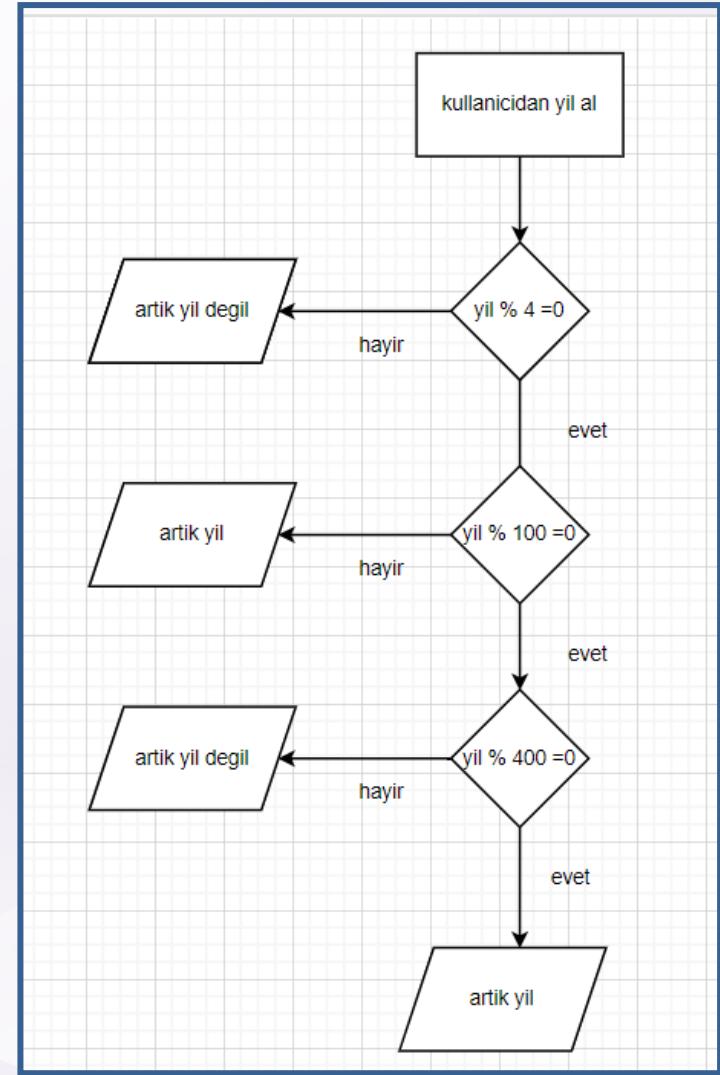
Kullanicidan artik yil olup olmadigini kontrol etmek icin yil girmesini isteyin.

Kural 1: 4 ile bolunemeyen yillar artik yil degildir

Kural 2: 4 ile bolunup 100 ile bolunemeyen yillar artik yildir

Kural 3: 4'un kati olmasina ragmen 100 ile bolunebilen yillardan sadece 400'un kati olan yillar artik yildir

<https://app.diagrams.net/>





# Nested If Statements

Eger kontrol edilecek degisken birden fazla ise ic ice loop'lar olusturmamiz gerekir.

**Ornegin :** Kullanicidan cinsiyetini ve yasini alin, Kadın, 60 yas ve uzeri , Erkek 65 yas ve uzeri emekli olabilir. Cinsiyet ve yasini dikkate alarak “Emekli olabilirsin” veya “Emekli olmak icin .. Yil daha calisman gerekir” yazdirin.

Buna benzer sorularda, degiskenlerden bir tanesini secip ona gore ana yapiyi kurmali, ondan sonra diger degiskene gore detay olusturulmalidir.

```
String cinsiyet= "Kadin";
int yas= 61;

if(cinsiyet.equals("Kadin")){
    // 60'dan buyukse emekli yoksa degil

} else if(cinsiyet.equals("Erkek")){
    // 65'den buyukse emekli yoksa degil

}else {
    // cinsiyet bilgisi hatali
```



# Nested If Statements Sorular

- Soru 1-** Kullanicidan cinsiyetini ve yasini alin, Kadın, 60 yas ve uzeri , Erkek 65 yas ve uzeri emekli olabilir. Cinsiyet ve yasini dikkate alarak “Emekli olabilirsın” veya “Emekli olmak icin .. Yıl daha calisman gerekir” yazdirin.
- Soru 2-** Kullanicidan aldiği ürün adedi ve ve liste fiyatını alın, kullanıcıya musteri kartı olup olmadığını sorun. Musteri kartı varsa 10 urunden fazla alırsa %20, yoksa %15 indirim yapın, Musteri kartı yoksa 10 urunden fazla alırsa %15, yoksa %10 indirim yapın
- Soru 3-** Kullanicidan bir sayı alın sayı tek ise negatif veya pozitif tek sayı olduğunu yazdırın, sayı çift sayı ise 10'un tam kati olup olmadığını yazdırın.
- Soru 4-** Kullanicidan günü ismini girmesini isteyin, girilen gün hafta içi bir gun ise “Simdi calisma zamani tatile .. gun var” şeklinde hafta sonu tatiline kaç gun kaldığını yazdırın, girilen gun hafta sonu ise “Simdi dinlenme zamani” yazdırın.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-09

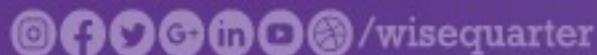
Nested If Else Statements  
Ternary Operators  
Switch Statements



The future at your fingertips

+1 912 888 1630

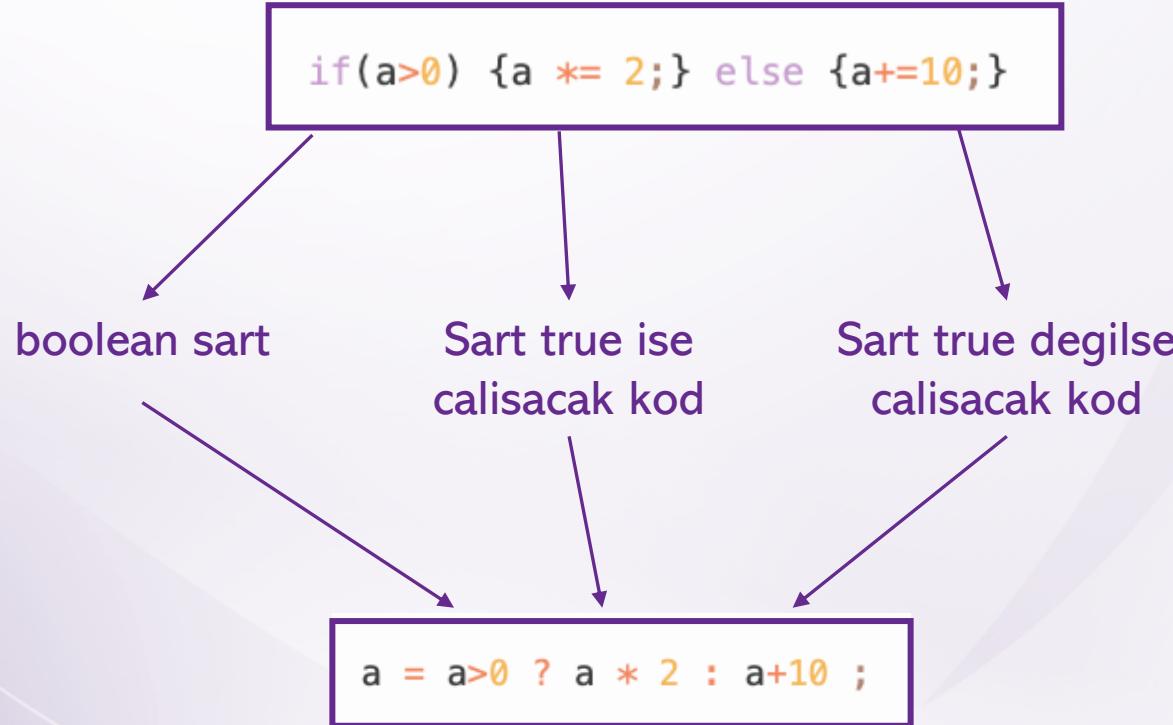
[www.wisequarter.com](http://www.wisequarter.com)





# Ternary Operator

Ternary, if-else statements ile yapabilecegimiz basit islemleri, daha basit bir formda kodlama imkani verir.



If-else statements'da if ve else body'lerinde kompleks kodlar yazabiliriz,  
Ancak ternary'de sadece deger veya deger hesaplayacagimiz basit kodlar yazabiliriz.



# Ternary Operator

Ternary,sadece deger dondurdugu icin, ya yazdirmali veya bir variable'a atamalisiniz.

```
a>0 ? a * 2 : a+10 ;
```

```
System.out.println( a > 0 ? a * 2 : a + 10);
```

```
a= a > 0 ? a * 2 : a + 10;
```



# Ternary Operator

Bir Ternary Ifadenin sonucunu yazdirdigimizda, şart sağlanırsa veya sağlanmazsa yazdırılacak datanın turu onemli olmaz

```
int a = 10;  
  
System.out.println(a > 0 ? "girilen sayı pozitif" : a + 10);
```

Ancak, ternary ifade'nin sonucunu bir variable'a atama yapacaksak, şart sağlanırsa veya sağlanmazsa elde edilecek sonucun aynı data turunde olması gereklidir.

```
int a = 10;  
  
a= a > 0 ? "girilen sayı pozitif" : a + 10;
```

```
a= a > 0 ? a * 2 : a + 10;
```

# Ternary Operator Sorular

**Soru 1-** Kullanicidan bir sayi isteyin, sayiyi kontrol edip 5 ile bolunebiliyorsa  
“Sayi 5'in tam kati” yazdirin.

**Soru 2-** Kullanicidan bir ucgenin 3 kenar uzunlugunu alin, ucgen eskenar ise  
“Eskenar ucgen” yazdirin, degilse “Eskenar degil” yazdirin.

**Soru 3-** Kullanicidan bir harf isteyin, girilen karakter kucuk harf ise onu buyuk  
harf olarak yazdirin, yoksa girilen harfi yazdirin

**Soru 4-** Kullanicidan notunu alin 50 veya daha buyukse ”Sinifi Gectin”, 50'den  
kucukse ”Maalesef kaldin” yazdirin.

**Soru 5-** Kullanicidan iki sayi alin ve buyuk olmayan sayiyi yazdirin

**Soru 6-** Kullanicidan bir sayi alin ve mutlak degerini yazdirin



# Ternary Operator

Asagidaki kod'lar calistiginda konsolda ne yazdiracagini bulun

```
int a = 10;

System.out.println(a>0 ? "Sayi Pozitif" : "Sayi Pozitif degil");

System.out.println(a>20 ? a*a : a++);

System.out.println(a<100 || a<0 ? 3*a+1 : 2 + a /5 );

int x=10;
int y=15;

int z = a>0 ? y++ : --x;

System.out.println(x +" , "+y+" , "+ z);
```

# Nested Ternary Operator

Ternary operatoru basit islemlerde kullanilmak uzere dizayn edilse de bazen kompleks islemleri de ternary ile yapmak isteyebilirsiniz (Tavsiye edilmez).

Ornek : Kullanicidan bir tamsayi alin.

Sayi pozitifse, cift sayi veya cift sayi degil seceneklerinden uygun olani yazdirin

Sayi pozitif degilse, 3 basamaklı veya 3 basamaklı degil seceneklerinden uygun olani yazdirin

```
System.out.println(a>0 ? Sayi pozitifse calisacak kod : Sayi pozitif degilse calisacak kod);
```

```
a%2==0 ; "sayi cift sayi" : "sayi cift sayi degil"
```

```
a<=-100 && a>-1000 ? "3 basamakli" : "3 basamakli degil"
```



# Nested Ternary Operator

Asagidaki kod'lar calistiginda konsolda ne yazdiracagini bulun

```
int a = 10;  
int b = 20;
```

```
System.out.println( a > 5 ? a > 0 ? 100 : 50 : a < 20 ? a + 5 : a - 5);
```

```
System.out.println( b < a ? b > 0 ? b+a : b-a : a < 10 ? a * 5 : b/a);
```

```
System.out.println( a == b ? a > b ? a : b : a < b ? a + b : a - b);
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-10

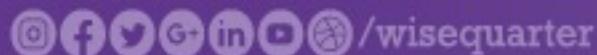
Switch Statements  
String Manipulations



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





# Switch Statements

If else statement ile cozdugumuz sorularda olasi durumların sayisi arttıkça, if else if... yapısı kurgulaması ve anlasılması zor hale gelebilir.

Ornegin kullanıcının rakam olarak girdiği gün numarının ismini yazmamız için 8 if else gereklidir, bu durum sonrasında kodu inceleyenler için uzun ve zor olabilir.

```
Scanner scan = new Scanner(System.in);
System.out.println("Lütfen bir rakam giriniz");
int rakam= scan.nextInt();

if (rakam==1){System.out.println("Pazartesi");}
else if (rakam==2){System.out.println("Sali");}
else if (rakam==3){System.out.println("Carsamba");}
else if (rakam==4){System.out.println("Persembe");}
else if (rakam==5){System.out.println("Cuma");}
else if (rakam==6){System.out.println("Cumartesi");}
else if (rakam==7){System.out.println("Pazar");}
else {System.out.println("Gun sayisi gecersiz");}
```



# Switch Statements

Bu tur sorulari daha anlasilir bir kod ile cozmek icin switch – case yapisi kullanabiliriz

```
switch (rakam){  
    case 1: System.out.println("Pazartesi");  
    break;  
    case 2: System.out.println("Sali");  
    break;  
    case 3: System.out.println("Carsamba");  
    break;  
    case 4: System.out.println("Persembe");  
    break;  
    case 5: System.out.println("Cuma");  
    break;  
    case 6: System.out.println("Cumartesi");  
    break;  
    case 7: System.out.println("Pazar");  
    break;  
    default: System.out.println("Gun sayisi gecersiz");  
}
```

# Switch Statements

Switch Statement kullanımında dikkat edilecek konular.

- 1- Switch Statement'da switch parantezinde long, double, float ve boolean kullanılamaz.
- 2- Switch Statement'da switch parantezinde yazılan degere uygun case calisir ve **break** gorunceye veya switch case bitinceye kadar calismaya devam eder.

```
switch (rakam){  
    case 1: System.out.println("Pazartesi");  
    break;  
    case 2: System.out.println("Sali");  
    break;  
    case 3: System.out.println("Carsamba");  
    break;  
    case 4: System.out.println("Persembe");  
    break;  
    case 5: System.out.println("Cuma");  
    break;  
    case 6: System.out.println("Cumartesi");  
    break;  
    case 7: System.out.println("Pazar");  
    break;  
    default: System.out.println("Gun sayisi gecersiz");  
}
```

break; komutunu her case'den sonra kullanmak zorunda değiliz, ancak bu durumda kodun break gorunceye kadar devam edeceğini unutmamamız gereklidir.

case'leri grüplendirmek için bu yöntem kullanılabilir.

- 3- switch parantezine yazılan değer hiç bir case ile uyusmazsa **default;** satırı devreye girer. (if-else if-if... lerin sonundaki else gibi)



# Switch Statements

**Soru 1-** Kullanicidan bir rakam alip, rakami yaziyla yazdirin.

**Soru 2-** Kullanicidan 2 basamakli bir sayi alip, girilen sayiyi yazi ile yazdirin

**Soru 3-** Kullanicidan ay numarasini alip ay ismini yazdirin

**Soru 4-** Kullanicidan ISTQB kisaltmasindan harfin anlamini ogrenmek istedigini alin ve girilen harfin karsiligini yazdirin.

I : International S : Software T : Testing Q : Qualifications B: Board

**Soru 5-** Kullanicidan gun numarasini alip hafta ici veya hafta sonu yazdirin

**Soru 6-** Kullanicidan ay numarasini alip mevsimi yazdirin.



# String Manipulations

Verilen bir String'i **hazir method'ları** kullanarak **degistirmeye** denir.

String manipulation yapilirken, degisikligin kalici olmasi isteniyorsa, atama yapilmalidir.

```
String str= "Java candir";  
  
System.out.println(str.toUpperCase()); // JAVA CANDIR  
  
System.out.println(str); // Java candir
```

String class'inin ozelliginden dolayi (ileride anlatilacak – immutable class), atama yapilmadan calistirilan method'lar variable'da kalici degisiklik yapmazlar.

```
String str= "Java candir";  
  
str=str.toUpperCase();  
  
System.out.println(str); // JAVA CANDIR
```

# String Manipulations

1. str.toUpperCase( );

Verilen String'i buyuk / kucuk harfe cevirir

2. str.toLowerCase( );

```
String str= "Java candir";  
  
str=str.toUpperCase();  
  
System.out.println(str); // JAVA CANDIR  
  
System.out.println(str.toLowerCase()); // Java candir
```

Eger bu degisimi yaparken ingilizce disinda bir dili esas almak isterseniz Locale secennegi kullanilir.

```
str="JAVA CANDIR";  
System.out.println(str.toLowerCase(Locale.GERMAN)); // java candir  
System.out.println(str.toLowerCase(Locale.forLanguageTag("Tr"))); // java candır
```

# String Manipulations

3. str.equals( baskaStr ); Verilen iki String'in metinlerini karsilastirir. İki String birbiriyle ayni metinleri iceriyorsa **true**, herhangi bir farklilik varsa **false** dondurur.

```
String str1= "Fatih";
String str2= "fatih";
String str3= new String( original: "Fatih" ); // Fatih

System.out.println(str1.equals(str3)); // true
System.out.println(str1.equals(str2)); // false
```

**NOT :** Diger primitive data turlerinde kullandigimiz == (double equal sign)'nin iki String'i karsilastirmak icin kullanilmasi tavsiye edilmez.

lleride detayini gorecegiz( **String Pool** ).

== karsilastirirken hem metne hem de stack memory'deki referansa baktigi icin tamamen ayni metne sahip iki String'i karsilastirirken **bazen true, bazen false** donecektir.

```
String str1= "Fatih";
String str2= "Fatih";
String str3= new String( original: "Fatih" ); // Fatih

System.out.println(str1==str2); // true
System.out.println(str1==str3); // false
```



# String Manipulations

## 4. str.equalsIgnoreCase( baskaStr );

Verilen iki String'in metinlerini karsilastirir. Case-sensitive olmadan birbiriyle ayni metinleri iceriyorsa **true**, herhangi bir farklilik varsa **false** dondurur.

```
String isim1 = "Kadir";
String isim2 = "kadir";
String isim3 = "Kadir ";

System.out.println(isim1.equals(isim2)); // false
System.out.println(isim1.equalsIgnoreCase(isim2)); // true

System.out.println(isim1.equals(isim3)); // false
System.out.println(isim1.equalsIgnoreCase(isim3)); // false
```

**NOT :** equalsIgnoreCase( ); sadece buyuk / kucuk harf farkliliklarini ignore eder, farkli bir karakter bulunmasi durumunda **her zaman false** donecektir (bosluk da bir karakterdir)



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-11

### String Manipulations



+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Onceki Gunden Aklimizda Kalanlar

- 1- Switch statements : If else ile yapabilecegimiz sorularda, secenek cogaldikca kodun anlasilir olmasi zorlasir. Secenekler cogaldiginda if-else yerine switch statements kullanmayi tercih ederiz.
- 2- double, float, boolean ve long data turundeki degiskenler switch statements'da kullanilamaz
- 3- switch parantezinde yazilan degisken'in degerleri case'lerde yazilir. Java verilen degere karsilik gelen case'den baslar, break gorunceye veya switch statement bitinceye kadar calismaya devam eder.
- 4- eger birden fazla case icin calistirilacak kodlar ayni ise break kullanmadan bu case'leri alt alta yazip, sonuncu case'de calisacak kodlari yazabiliriz.
- 5- default, if-else'lerin sonundaki else gibidir. Hicbir case'e uymayan TUM DEGERLER icin default calisir.
- 6- String Manipulations : Verilen bir String'in hazir methodlar kullanilarak degistirilmesi veya String'in bazi bilgilerinin elde edilmesi icin yapılan tum islemlere denir.
- 7- İki String'in esitligini karsilastirmak icin == yerine equals( ) kullanilir.
- 8- == hem degerlere, hem de referans'a bakar. equals( ) ise sadece metne odaklanir.
- 9- equals( ) ile karsilastirilan metinlerde true donmesi icin metnin tamamiyla ayni olması gereklidir. Harf farkliliği, buyuk-kucuk harf farkliliği gibi durumlarda false doner.
- 10- Eger case sensitive olmadan metinlerin esitligini kontrol etmek istersek ignorecase

# String Manipulations

5. str.charAt( istenenIndex );

Verilen bir String'in istenen index'indeki char karakteri bize döndürür.

```
String str= "Java Candır";  
  
System.out.println(str.charAt(0)); // J  
System.out.println(str.charAt(3)); // a  
System.out.println(str.charAt(10)); // r
```

**NOT 1:** Java'da index 0'dan baslar. İlk harfe ulaşmak için str.charAt(0); yazmalısınız

**NOT 2:** Index 0'dan basladığı için son index toplamKaraktersayısı -1 olacaktır.

Yukarıdaki örnekte karakter sayısı 11 iken, son harfe ulaşmak için charAt(10); kullanılmalıdır.

**NOT 3:** Son index'den daha büyük bir index yazdığında java hata verir

```
System.out.println(str.charAt(20));
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException Create breakpoint : String index out of range: 20  
at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:47)  
at java.base/java.lang.String.charAt(String.java:693)  
at day08_switchStatements_StringManipulations.C10_charAt.main(C10_charAt.java:17)
```

# String Manipulations

## 6. str.length( );

Verilen bir String'deki karakter sayisini bize döndürür.

```
String str= " Uzunkavaklaraltindayataruyumazoglu";  
  
System.out.println(str.length()); // 35
```

**NOT 1:** Index 0'dan basladigi icin son index str.length() -1 olacaktir. Yukarıdaki ornekte length(uzunluk) 11 oldugundan, son index  $35-1 = 34$  olacaktir.

**NOT 2:** Her hangi bir karakter istenirken index sayisi degil, sondan kacinci harf oldugu veriliyorsa (`length - sondanKacinciKarakter`) index olarak kullanilabilir. Ornegin sondan 3.karakter isteniyorsa,

```
System.out.println(str.charAt(str.length()-3)); // g
```

# String Manipulations

## 7. str.substring( istenenParametre );

Verilen bir String'in istedigimiz bir bolumunu bize döndürür. İstedigimiz bolume uygun olarak 1 parametre veya 2 parametrelili 2 farkli kullanimi vardir

str.substring( tekParametre );

Parametre olarak girilen index'den String'in sonuna kadar olan bolumunu bize döndürür.

```
String str= "Java Guzeldir";  
  
System.out.println(str.substring( beginIndex: 2)); // va Guzeldir  
  
System.out.println(str.substring( beginIndex: 10)); // dir
```

Eger sondan istenen kadar karakteri elde etmek istersek length( )-istenenkarakterSayisi kullanilir.

```
System.out.println(str.substring( beginIndex: str.length()-3)); // dir  
  
System.out.println(str.substring( beginIndex: str.length()-1)); // r
```

# String Manipulations

7. str.substring( istenenParametre );

str.substring( baslangicIndex, bitisIndex );

Parametre olarak girilen iki index'den baslangic index'i dahil, bitis index'i haric bolumunu bize döndürür.

```
String str= "Java Guzeldir.";  
  
System.out.println(str.substring(1,3)); // av  
  
System.out.println(str.substring(5,10)); // Guzel  
  
System.out.println(str.substring(0,12)); // Java Guzeldi
```

Eger parametre olarak sondan belirlenen index'i istersek length( ) kullanilabilir.

```
System.out.println(str.substring(0, str.length()-3)); // Java Guzeld
```

# String Manipulations

## 7. str.substring( istenenParametre );

charAt(istenenIndex); method'u bize istenen indexdeki karakteri dondurur fakat char oldugu icin sonrasinda String method'ini kullanamayiz, bunun yerine substring kullanabilir

```
// sadece 5.index'deki harfi yazdiralim
System.out.println(str.substring(5,6));

// sadece 2.indexteki harfi buyuk harf olarak yazdiralim
System.out.println(str.substring(2,3).toUpperCase());
```

Eger parametre olarak ayni index'i baslangic ve bitis olarak secerek bize hiclik döndürür.

```
System.out.println(str.substring(3,3));
// hiclik yazdirir, konsolda birsey gorunmez
```

Eger parametre olarak girilen bitis index'i, baslangictan kucuk olursa hata olusur.

```
System.out.println(str.substring(5,2));
// RTE 5.index'den sonra 2.index'i bulamaz
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException Create breakpoint : begin 5, end 2, length 14
```

# String Manipulations

## 8. str.concatenation( baskaString );

Istedigimiz String'in sonuna baska bir String ekler.

Daha once String variable'lar ile + (toplama) islemi yaptigimizda, Java'nin bunu matematiksel toplama olarak degil birlestirme (concat) olarak algiladigini soylemistik.

+ islemini method ile yapmak istersek concat( ) kullanabiliriz.

```
System.out.println(a+d+b+d+s+t); // Java Guzeldir 54
System.out.println(a.concat(d).concat(b).concat(d).concat(str: s+t+c)); // Java Guzeldir 9
```



# String Manipulations

## 9. str.contains ( baskaString );

Bir String'in başka bir String'i icerip icermedigini kontrol eder, boolean sonuc döndürür.

Aranan String'in kaç tane olduğunu tespit edemez, sadece **var** veya **yok** cevabi verir.

```
String str= "Java cok guzel, cok.";  
  
System.out.println(str.contains("Java")); // true  
  
System.out.println(str.contains("java")); // false  
  
System.out.println(str.contains("cok")); // true  
  
System.out.println(str.contains("a")); // true  
  
System.out.println(str.contains(" ")); // true  
  
System.out.println(str.contains(""))); // true
```

**NOT :** contains( ) parametre olarak **char** kabul etmez, ananan charSequence yani String olmalıdır

# String Manipulations

## 10. str.startsWith ( baskaString );

Bir String'in başka bir String ile baslayip, baslamadigini kontrol eder, boolean sonuc döndürür.

2 kullanım sekli vardir. **Tek parametreli** olursa str'in baş kismini kontrol eder

```
String str="Java çok guzel,cok.";  
  
System.out.println(str.startsWith("J")); // true  
System.out.println(str.startsWith("Java")); // true  
System.out.println(str.startsWith("Java çok guzel,cok.")); // true  
System.out.println(str.startsWith(""))); // true
```

**2 parametre** olursa, Java'ya baslangic olarak hangi index'i kullanmasini istedigimizi de verebiliriz.

Ornegin, 5.index ve sonrasi "cok" ile mi basliyor diye kontrol edebiliriz.

```
System.out.println(str.startsWith( prefix: "cok", toffset: 5)); // true  
System.out.println(str.startsWith( prefix: "guzel", toffset: 10)); // false
```

# String Manipulations

## 11. str.endsWith ( baskaString );

Bir String'in başka bir String ile bitip, bitmediğini kontrol eder, boolean sonuc döndürür.

```
String str="Java çok güzel,cok.";  
  
System.out.println(str.endsWith("cok")); // false  
System.out.println(str.endsWith("cok.")); // true  
System.out.println(str.endsWith(""))); // true
```

SORU : kullanıcidan bir mail alın

- mail @ icermiyorsa "gecersiz mail"
- mail @gmail.com icermiyorsa, "mail gmail olmalı"
- mail @gmail.com ile bitmiyorsa, "mailde yazım hatası var"

yazdırın.



# String Manipulations

## 12. str.indexOf ( baskaString/char );

Bir String icerisinde aradigimiz bir String veya char degerin ilk kullanım index'ini döndürür.

2 parametre kullanırsak aramaya hangi index'den baslayacagini da söyleyebiliriz.

```
String str="Java cok guzel,cok.";  
  
System.out.println(str.indexOf('a'));  
// bulduğu ilk a'nın index'ini verir : 1  
  
System.out.println(str.indexOf( ch: 'a', fromIndex: 1));  
// 1.index ve sonrasında a arar : 1  
  
System.out.println(str.indexOf( ch: 'a', fromIndex: 2)); // 3  
  
System.out.println( str.indexOf("cok")); // 5  
  
System.out.println(str.indexOf( str: "cok", fromIndex: 6)); // 15
```

# String Manipulations

## 12. str.indexOf ( baskaString/char );

indexOf( ) method'u bize int index döndürür.

String icerisinde olmayan bir metin aradigimizda Java'nin bunu bize bir integer ile anlatmasi gereklidir. O ve pozitif sayilar index olarak kullanilabilecegi icin, Java aradigimiz metin aranan String'de olmadiginda bize **-1** döndürerek, durumu rapor eder.

```
String str="Java cok guzel,cok.";  
  
System.out.println(str.indexOf("Soner")); // -1  
System.out.println(str.indexOf('t'));
```

# String Manipulations

12. str.indexOf ( baskaString/char );

**Soru 1 :** Kullanicidan bir String ve aranacak metin alin. String'in aranan metni icerip icermedigini indexOf( ) method'u kullanarak yazdirin.

**Soru 2 :** Kullanicidan bir String ve aranacak metin alin. Aranan metnin String icerisinde kullanimini kontrol ederek asagidaki cumlelerden uygun olanini yazdirin.

- String aranan metni icermiyor
- Aranan metin String'de sadece 1 kere kullanilmis
- Aranan metin String'de sadece 1'den fazla kullanilmis

# String Manipulations

## 13. str.lastIndexOf(arananString );

Aranan String veya char'in verilen metindeki en son kullanımını bulur ve index'ini döndürür.

```
String str= "Java çok güzel, çok";  
  
System.out.println(str.indexOf("cok")); // 5  
System.out.println(str.lastIndexOf(str: "cok")); // 16  
  
System.out.println(str.indexOf('o')); // 6  
System.out.println(str.lastIndexOf(ch: 'o')); // 17
```

str.lastIndexOf(arananString, sonIndex ); şeklinde kullanırsak aramaya sonIndex olarak girilen index'den başlar ve basa doğru devam eder.

Her iki kullanımda da arananString/char'i bulamazsa -1 döndürür.

```
System.out.println(str.lastIndexOf(str: "cok" , fromIndex: 10)); // 5  
// 10.index ve öncesinde arama yapar  
System.out.println(str.lastIndexOf(ch: 'x')); // -1  
System.out.println(str.lastIndexOf(str: "x" , fromIndex: 10)); // -1
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-12

### String Manipulations



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# String Manipulations

13. str.lastIndexOf( baskaString/char );

**Soru 1 :** Kullanicidan bir String ve aranacak metin alin. String'in aranan metni icerip icermeydigini lastIndexOf( ) method'u kullanarak yazdirin.

**Soru 2 :** Kullanicidan bir String ve aranacak metin alin. Aranan metnin String icerisinde kullanimini kontrol ederek asagidaki cumlelerden uygun olanini yazdirin

- String aranan metni icermiyor
- Aranan metin String'de sadece 1 kere kullanilmis
- Aranan metin String'de sadece 1'den fazla kullanilmis

# String Manipulations

## 14. str.isEmpty( );

Verilen bir String'in bos olup olmadigini boolean olarak döndürür.

```
String str= "Java ogren, 70000 euro offer al";  
  
System.out.println(str.isEmpty()); // false  
  
String str2="";  
  
System.out.println(str2.isEmpty()); // true
```

String'in bos olmasi(length=0) ile sadece space'lerden olusmasi farklidir.  
Space'lerden olusan bir String'in uzunlugu 0 olmayacagi icin isEmpty( ) bize false döndürür.

Bir String'in sadece space'lerden olusmus oldugunu kontrol icin str.isBlank( ) kullanilabilir.

```
String str3= " ";  
System.out.println(str3.isEmpty()); // false  
System.out.println(str3.isBlank()); // true
```

# String Manipulations

## Null Pointer

Null pointer bir deger degil isaretcidir.



```
String isim1=null;  
  
String isim2;  
  
String isim3="";
```

Yandaki 3 isim variable'nin durumlari birbirinden tamamen farklıdır.

İsim3'e bir deger atanmistir. Bu degeri yazdirabilir veya method'larda kullanilabilir.

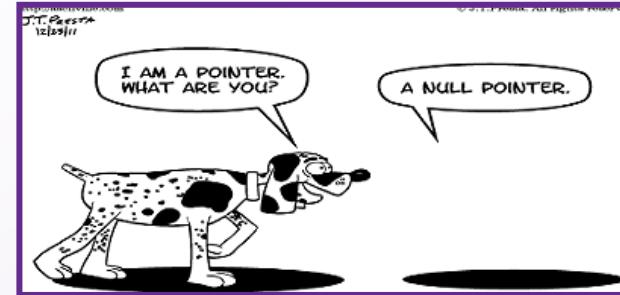
```
System.out.println(isim3);  
//hiclik yazdirir, konsolda birsey gorunmez  
  
System.out.println(isim3.length()); // 0
```



# String Manipulations

## Null Pointer

```
String isim1=null;  
  
String isim2;  
  
String isim3="";
```



İsim1 ve isim2'nin durumları biraz daha benzerdir.

İkisi de oluşturulmuş ve ikisine de **deger atanmamıştır**.

İsim1 null pointer ile işaretlendiği için Java isim1'in kullanım sorumluluğunu bize bırakır.

İsim2'yi kullanmanıza ise (**deger atanmadığı surece**) izin vermez.

```
System.out.println( isim2 ); // CTE  
System.out.println( isim2.length() ); // CTE  
  
System.out.println(isim1); // null  
System.out.println(isim1.length()); // NullPointerException
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-13

### String Manipulations For Loops



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# String Manipulations

15. str.replace ( degisecekString , yeniDeger );

Bir String'in icinde bulunan degisecekString'lerin tumunu yeniDeger yapar.

Parametre olarak char da kullanilabilir, bu durumda degisecekChar'larin tumunu yeniChar yapar.

```
String str= "Java ogren, isi kap";  
  
System.out.println(str.replace( oldChar: 'J', newChar: 'j'));  
// java ogren, isi kap  
  
System.out.println(str); // Java ogren, isi kap  
  
str=str.replace( target: "isi", replacement: "offer'i");  
// String'deki degisikligin kalici olmasi icin atama yapmaliyiz  
  
System.out.println(str); // Java ogren, offer'i kap
```

degisecekString ile yeniDeger'in ayni uzunlukta olmasi şart değildir. Daha uzun veya daha kısa olabilir.

degisecekString'i tamamen silmek istiyorsak yeniDeger'i “” (hiclik) secebiliyoruz.

# String Manipulations

16. str.replaceFirst ( degisecekString , yeniDeger );

Bir String'in icinde bulunan degisecekString'lerden ilkini yeniDeger yapar.

degisecekString String olabilecegi gibi regex de olabilir.

```
String str = "Herkesin github'i olmali";  
  
System.out.println(str.replaceFirst( regex: "e" , replacement: "a"));  
// Harkesin github'i olmali  
  
System.out.println(str.replaceFirst( regex: "\\\w" , replacement: "1"));  
// 1erkesin github'i olmali
```

## Regex (Regular Expressions)

\s : space

\S : space olmayan hersey

\s+ : yanyana birden fazla space

\d : digits

\D : digit olmayan hersey

\w : harf, rakam veya \_

\W : harf, rakam veya \_ olmayan hersey

# String Manipulations

## 16. str.replaceAll ( Regex , yeniDeger );

Bir String'in icinde bulunan degisecekRegex'e uyan tum karakterleri yeniDeger yapar.

str.replace'den farki tek tek harfleri veya metinleri degistirmek yerine, parametre olarak girilen regex'in kapsadigi tum karakterleri degistirmesidir.

Tum rakamlar, tum space'ler, rakam olmayan tum karakterler vb...

```
str="J1a2va3 G4u5z6e7l8d9i0r.";

// ornegin yukarida metin'de tum rakamlardan tek seferde kurtulalim
str=str.replaceAll( regex: "\d", replacement: "");

System.out.println(str); // Java Guzeldir.

// eger birden fazla bosluk olan yerleri tek space yapmak istersek
str="Java      Guzel bir programlama      dili";

str=str.replaceAll( regex: "\s+", replacement: " ");

System.out.println(str); // Java Guzel bir programlama dili
```

# String Manipulations

## 18. str.repeat ( tekrarSayisi);

Bir String'i tekrarSayisi kadar tekrar ettirir.

```
String str= "Java Candir.";  
  
System.out.println(str.repeat( count: 4));  
// Java Candir.Java Candir.Java Candir.Java Candir.
```

## 19. str.trim ( );

Bir String'in basında ve sonunda (**varsa**) bulunan space'leri siler.

```
str= "    Ali kos    ";  
  
str=str.trim();  
  
System.out.println(str); // Ali kos
```

# String Manipulations

**Soru 1 :** Kullanicidan bir cumle alin

- cumlede ev geciyorsa, "home home sweet home" yazdirin
- cumlede is geciyorsa, "calismak guzeldir"
- ikisini de iceriyorsa "Hem ev lazim hem is"
- hicbirini icermiyorsa "cok calisman lazim" yazdirin

**Soru 2 :** Kullanicinin belirli bir formatta verdigi String fiyatları toplayip yazdirin.

input1 : "15.30 €" , input2 : "11.40 €"

output : 26.70 €

**Soru 3 :** Kullanicidan alınan metindeki istenmeyen rakam ve ozel karakterleri silip, sadece ilk harfi buyuk diger harfler kucuk harf yapan bir program yazın.

input : java1 ogRe2@nMek3 #ne +Gu=zel

output : Java ogrenmek ne guzel.

# String Manipulations

**Soru 4 :** Kullanicidan bir sifre isteyip, asagidaki şartlari kontrol edin ve kullaniciya duzeltmesi gereken tum eksikleri soyleyin, eger tum şartlari saglarsa, "sifre basariyla kaydedildi" yazdirin

- ilk harf kucuk harf olmali
- son karakter rakam olmali
- sifre bosluk icermemeli
- uzunlugu en az 10 karakter olmali

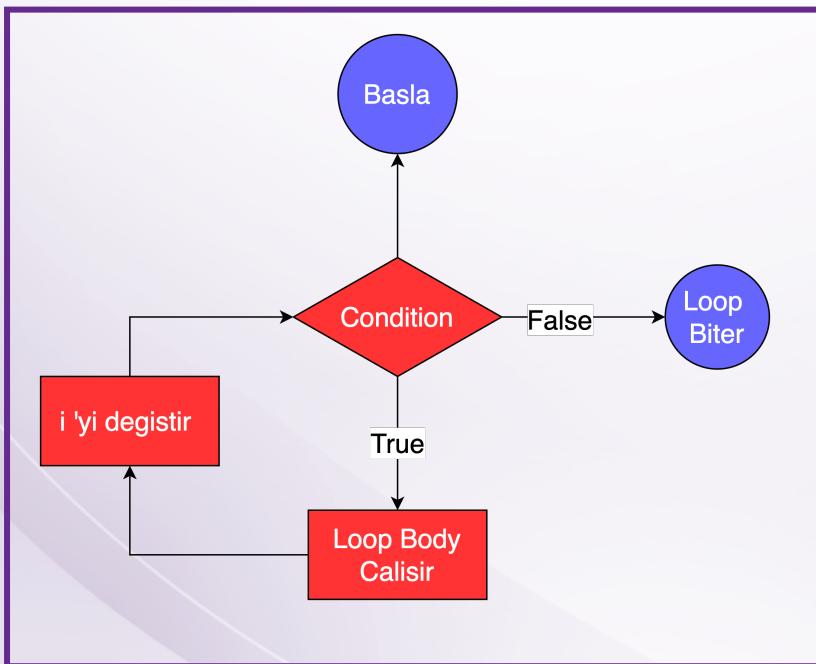
**Soru 5 :** Kullanicidan isim ve soyismini ayri ayri alin.

- ismi daha uzun ise, isim ve soyismi ilk harf buyuk kalanlar kucuk seklinde yazdirin
- soyisim daha uzun ise ismi ilk harf buyuk digerleri kucuk, soyismi buyuk harflerle yazdirin.

**Soru 6 :** Kullanicidan alınan bir String alın, String'in uzunluğu çift sayı ise tam ortasına :) ekleyin, String'in uzunluğu tek sayı ise ortadaki harfi silin ve yerine :( yazdırın.

# For Loops

For Loop, belirli sayıda çalıştırılması gereken bir döngüyü verimli bir şekilde yazmanızı olanak tanıyan bir tekrar kontrol yapısıdır.



For döngüsü, bir görevin kaç kez tekrarlanacağını bildiğinizde kullanışlıdır.

```
for (int i = 0; i < 10 ; i++) {  
    /* şart sağlandığında çalışacak kod */  
    System.out.print(i + " ");  
}
```



# For Loops

NOT 1 : Condition i'nin tum degerleri icin hep true oluyorsa

```
for (int i = 0; i >-10 ; i++) {  
    System.out.print(i + " ");  
}
```

Sonsuz loop olusur

NOT 2 : Condition i'nin ilk degeri icin bile false oluyorsa

```
for (int i = 0; i >10 ; i++) {  
    System.out.print(i + " ");  
}
```

For loop calisir ancak loop body calismaz



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-14

For Loops

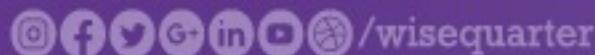
Nested For Loops



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# For Loops

**Soru 1-** 1'den 100'e kadar sayilari aynı satırda aralarında bir boşluk bırakarak yazdırın.

**Soru 2-** Kullanıcıdan pozitif bir tamsayı alın, 1'den girilen sayıya kadar(girilen sayı dahil) 7 ile bolunebilen sayıları yazdırın.

**Soru 3-** Kullanıcıdan başlangıç ve bitiş değeri olarak pozitif sayılar alın, sınırlar dahil olarak aralarındaki tüm sayıların toplamını yazdırın. Bitiş değeri başlangıç değerinden küçükse, uyarı yazdırıp işlemi sonlandırın

**Soru 4-** Kullanıcıdan başlangıç ve bitiş değeri olarak pozitif sayılar alın, sınırlar dahil olarak aralarındaki tüm sayıların toplamını yazdırın. Bitiş değeri başlangıç değerinden küçük olsa da program çalışın

**Soru 5-** Kullanıcıdan 20'den küçük bir sayı alıp, bu sayının faktöryel değerini hesaplayın.

**Soru 6-** Kullanıcıdan 20'den küçük bir sayı alıp, bu sayının faktöryel değerini hesaplayın. Konsolda faktöryel hesabının yapılmasını da yazdırın.

$$\text{Or : } 6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$$

# For Loops

**Soru 7-** Kullanicidan pozitif bir tamsayi alip, rakamlar toplamini yazdirin.

**Soru 8 (interview)-** Kullanicidan pozitif bir sayi alin, 1'den baslayarak tum tamsayilari yazdirin, sira

- 3 ile bolunebilen bir sayiya gelirse, sayi yerine **fizz**
- 5 ile bolunebilen bir sayiya gelirse sayi yerine **buzz**
- hem 3 hem 5 ile bolunebilen bir sayiya gelirse sayi yerine **fizzBuzz** yazdirin

**Soru 9 (interview)-** Kullanicidan bir String isteyin ve String'i tersten yazdirin.

**Soru 10 (interview)-** Kullanicidan bir String isteyin ve String'i tersine cevirip kaydedin.

**Soru 11-** Kullanicidan pozitif bir tamsayi isteyip, sayinin asal sayi olup olmadigini kontrol edin ve sonucu yazdirin.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-15

### Nested For Loops

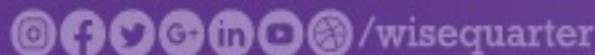
### Method Olusturma ve Kullanma



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# Nested For Loops

For Loop, belirli sayıda çalıştırılması gereken bir döngüyü verimli bir şekilde yazmanıza olanak tanıyan bir tekrar kontrol yapısıdır.

Bazen verilen görevi yapmak için tek bir loop yeterli olmaz, Örneğin bir carpım tablosu hazırlamak veya bir futbol liginde oynanacak maçları planlamak için tek bir loop yeterli olmaz.

Nested for-Loop ihtiyacı iki şekilde karşımıza çıkabilir.

1 -	1	2	3	4	1*1	1*2	1*3	1*4
	2	4	6	8	2*1	2*2	2*3	2*4
	3	6	9	12	3*1	3*2	3*3	3*4
	4	8	12	16	4*1	4*2	4*3	4*4

1.Sayı bir değer alındığında, 2. sayı baştan sona tüm değerleri alıyor

1.Sayı bir değer arttığında, 2. sayı baştan sona tüm değerleri yeniden alıyor



# For Loops

2 - 1

1. satir 1'den 1'e kadar yazdir

1 2

2. satir 1'den 2'e kadar yazdir

1 2 3

3. satir 1'den 3'e kadar yazdir

1 2 3 4.

4. satir 1'den 4'e kadar yazdir

Bu tarz sorularda iç dongu 1'den dış loop'un iç degerine kadar gidiyor.

Soru – Aşağıdaki şekilleri yazdırın

\*

\* \* \* \* \*

\* \* \* \* \*

\*\*

\* \* \* \* \*

\* \* \* \*

\*\*\*

\* \* \* \* \*

\* \* \*

\*\*\*\*

\* \* \* \* \*

\* \*



# Method Olusturma ve Kullanma

Method'lar istedigimiz islemleri bizim adimiza yapan kod bloklaridir.



Belirli bir isi yapmak icin tasarlanmis robotlar gibidirler. Baslangicta tasarlamaasi ve sorunsuz calismasi emek ve zaman ister ancak calismaya baslayinca, yaptigi islemi sorunsuz olarak tekrar tekrar yaptirabiliriz.

Method'lar da robotlar gibi calismasini istedigimizde calisir. Java calismaya main method'dan baslar ve bir Method Call (Method Cagirma) gorurse o method'u bulur ve calistirir.

# Method Olusturma ve Kullanma

Nicin bir islemi main method icerisinde yapmak yerine method olusturmamizi tercih ederiz ?

## 1- Projemiz icerisinde tekrar tekrar

kullanicigimiz bir islem icin her seferinde  
yeniden kod yazmak yerine bir kere yazip  
ihtiyacimiz oldukca kullanmak (OOP  
Concept)

Ornegin faktoryel hesaplamak zor bir islem  
degildir, ama her ihtiyacimiz oldugunda  
yeniden faktoryel hesaplamak icin kod  
yasmak yerine bir kere method olarak yazip  
ne zaman lazim olsa kullanmak daha pratik  
olacaktir.



## 2- Calistigimiz class'i ve main method'u basit bir yapida tutup, sectigimiz uygun isme sahip method'larla kodumuzu daha anlasilabilir hale getirmek.



# Method Olusturma ve Kullanma

Bir okul projesi yaptigimizi dusunelim.

Ogretmen ekleme, ogrenci  
kayit gibi islemler de  
binlerce kez  
tekrarlanacaktır.

```
===== YILDIZ KOLEJI =====
===== ANA MENU =====
```

- 1- Okul Bilgileri Goruntule
- 2- Ogretmen Menu
- 3- Ogrenci Menu
- Q- ÇIKIŞ

```
===== YILDIZ KOLEJI =====
===== OGRENCI MENU =====
```

- 1- Ogrenci Listesi Yazdir
- 2- Soyisimden Ogrenci Bulma
- 3- Sinif ve Sube Ile Ogrenci Bulma
- 4- Bilgilerini Girerek Ogrenci Ekleme
- 5- Kimlik No Ile Kayit Silme
- A- ANAMENU
- Q- ÇIKIŞ

```
===== YILDIZ KOLEJI =====
===== OGRETMETEN MENU =====
```

- 1- Ogretmenler Listesi Yazdir
- 2- Soyisimden Ogretmen Bulma
- 3- Branstan Ogretmen Bulma
- 4- Bilgilerini Girerek Ogretmen Ekleme
- 5- Kimlik No Ile Kayit Silme
- A- ANAMENU
- Q- ÇIKIŞ

Tum bu kodlari tek bir class'da ve main method'da yapmak uygulamamizi kontrol edilemez ve anlasilamaz bir hale getirecektir.

Gunumuzde Instagram, facebook gibi sosyal media uygulamalarin, bankacilik, alisveris siteleri gibi ticari programlarin veya e-devlet gibi bir ulkenin tum insanlarini ve yapılan tum resmi islemleri kapsayan uygulamalarin olusturulmasi ve kullaniminin pratik olarak yapisilmesi icin MUTLAKA method'lara ihtiyacimiz olacaktir.



# Method Olusturma ve Kullanma

Bir method'un sonuc olarak bize bir deger dondurmeyi saglar. Matemetik islemlerindeki sonuc gibidir.

Ornegin String method'larini incelerken, hem ne is yaptigina, hem de bize sonuc olarak hangi data turunden bir sonuc dondurdugune bakiyorduk.

```
str.to|  
  m toLowerCase() String  
  m toUpperCase() String  
  m toLowerCase(Locale.ROOT) String  
  m toLowerCase(Locale locale) String  
  m toUpperCase(Locale locale) String  
  m toUpperCase(Locale.ROOT) String  
  m toCharArray() char[]  
  m toString() String  
  m compareTo(String anotherString) int  
  m compareToIgnoreCase(String str) int  
Press ↵ to insert, → to replace Next Tip ::
```

# Method Olusturma ve Kullanma

Method'lar bize bir sonuc döndürüp döndürmedigine gore 2'ye ayrılır.

1- Bazi method'lar gorevlerini yapar ama bize herhangi bir data turunde sonuc dondurmezler. Bu tur method'larin return type'i void olur.

Ornegin ogrenci kaydi yapan bir method dusunelim, amac , kayit yapan kisiye bir sonuc dondurmekten ziyade ogrenciyi kayit yapmaktir. Belki kayit islemi tamamlandi diye bir sonuc yazdirilabilir ama bu yazdirma islemi asil amac degildir.



“Kayit basariyla yapildi” yazan ama kayit yapmayan bir method calisti Kabul edilemez.

Bu tur method'lari fatura yatirmaya yolladigimiz cocugumuz gibi dusunebiliriz. Amac faturayı yatırmaktır, bize bir makbuz getirmesi degildir.

# Method Olusturma ve Kullanma

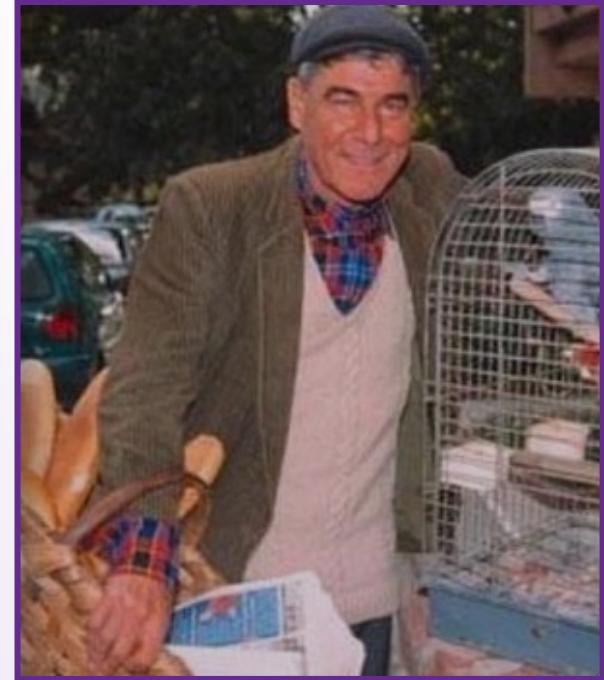
## Method Turleri

### 2- Bize sonuc döndüren method'lar.

Cogu zaman method'lar bize bir sonuc döndürmesi icin olusturulur.

Markete alisverise giden kapici gibidir, bizim istedigimiz urunu getirir. Onun getirmesi yetmez biz de kapicinin getirdigi urunu ondan almaliyiz.

```
String str= "Java Guzeldir.";  
  
str.toUpperCase(); // bize String dondurur
```



Bu method calistiginda konsolda bir sey de goremeyez, str da degismez.

Bize sonuc döndüren method'lar ya direk yazdirilmali veya data turune uygun bir variable'a atanmalidir.

```
System.out.println(str.toUpperCase());  
  
str= str.toUpperCase();
```



# Wise Quarter

first class IT courses

## JavaTeam 120

### Ders-16

Method **Olusturma ve Kullanma**

Method **Overloading**



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

@ /wisequarter



# Onceki Gunden Aklimizda Kalanlar

1- Method'lar robotlara benzer, yapmasi main method icinde olusturmaktan biraz daha zahmetli olabilir ama programlamada olmazsa olmazlardandir.

2- Method olusturmanin 2 temel faydasi vardir

- tekrar tekrar kullanabilme kolayligi
- main method'umuzu veya class'imizi basit ve anlasilabilir hale getirme

3- Genel kullanimda sadece 1 kere kullanacagimiz islemler icin method olusturma tercih edilmeyebilir ama bize surekli lazim olacak islemler icin mutlaka method olusturmaliyiz. Ornegin; bir ogrencinin not ortalamasini bulmak zor degildir, ama binlerce ogrencisi olan bir okulda not ortalamasini herseferinde manuel olarak yapmak mantikli degildir.

4- Method'larin urettigi sonuca islem sonucu demeyiz. Bunun yerine "bize su sonucu döndürür" deriz.

5- bir method'un dondurdugu sonucun data turunu method deklarasyonunda goruz.

6- method'lar sonuc dondurme durumuna gore 2'ye ayrilirlar

- islemi yapip bize bir sey dondurmayen(ogrenci kaydi, bankaya para yatirmak vb) veya islemi yapip sonucu sadece konsolda yazdiran(elektrik parasini yatirip, makbuz almak gibi) method'larin return type'i void olur
- islemi yapip, islem sonucunu bize donduren method'larin, return type'i dondurukleri dataya uygun olur (markete gonderdigimiz kapicinin, bizim istedigimiz urunu getirmesi gerektigi gibi)



# Method Olusturma ve Kullanma

void mi yoksa return type'li method mu tercih edilmelidir ?

Bu tercih bize verilen gereksinim (requirements)'a gore bizim belirleyecegimiz bir durumdur.

Ancak return type'i olan method'lar daha avantajlidir. Void bir method'a sonuc dondurtemeyiz ama sonuc donen bir method'u System.out.println( ) icinde kullanip sonunu yazdirabiliriz.

```
String str= "Java Guzeldir.";  
  
str.toUpperCase(); // bize String dondurur  
  
System.out.println(str.toUpperCase());  
  
str= str.toUpperCase();
```

# Method Olusturma ve Kullanma

Bir method olusturmak istedigimizde kullanilacak syntax soyledir.

```
public static void toplama(int sayi1, int sayi2){  
    System.out.println( sayi1 + sayi2 );  
}
```

1- access modifier : method'a proje icerisinden nerelerden ulasabilecegini belirler.

**public** : Proje icerisinde tum class'lardan kullanilabilir.

**protected** : Sadece icinde bulundugu package ve child class'lardan kullanilir

**default** : Sadece icinde bulundugu paket(package)'den kullanilir

**private**: Sadece bulundugu class'da kullanilabilir

Access Levels					
Modifier	Class	Package	Subclass	World	
public	Y	Y	Y	Y	
protected	Y	Y	Y	N	
no modifier	Y	Y	N	N	
private	Y	N	N	N	



# Method Olusturma ve Kullanma

Bir method olusturmak istedigimizde kullanilacak syntax soyledir.

```
public static void toplama(int sayi1, int sayi2){  
    System.out.println( sayi1 + sayi2 );  
}
```

**2- static :** Access modifier olmadigi halde method ve variable'lar icin erisimi duzenler.

static olarak isaretlenmis method'lar, method disinda bulunan variable ve method'lardan sadece static olarak isaretlenmis olanlara direkt ulasabilir.

main method static olarak isaretlendiginden (**simdilik**) main method'dan cagiracagimiz method'lari da static yapacagiz.

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



# Method Olusturma ve Kullanma

3- return type : Method'un hangi data turunden bir sonuc urettigini belirtir.

Gorevi sadece birsey yazdirmak olan method'larin return type'i void olarak belirlenir.

Method'un gorevi bize bir sonuc dondurmek ise, dondurecegi dataya uygun bir return type secilmeli,

method'un sonunda ise return keyword'u ile beklenen data turunden bir deger dondurulmelidir.

```
public static void sayiTposta(int sayi1, int sayi2){  
    System.out.println(sayi1+sayi2);  
}  
  
public static int sayiTpostaDondur(int sayi1, int sayi2){  
    return sayi1+sayi2;  
}  
  
public static void main(String[] args) {  
  
    sayiTposta(5,10); // 15 yazdirir  
  
    int sonuc= sayiTpostaDondur(20,30);  
    // 50 dondurup sonuc'a assign eder  
  
    // istersek sonucu yazdirabiliriz  
    System.out.println(sonuc); // 50
```

Return type'i void olan method'lar cagrildiginda, sadece **yazirma islemi** yapabilir, Void olmayan method'lar ise bize bir deger dondurur ve **biz de o degeri kaydederiz**,

Sonunu bir variable'a atadiktan sonra istedigimiz zaman yazdirmak mumkun olacaktir.



# Method Olusturma ve Kullanma

```
public static void toplama(int sayi1, int sayi2){  
  
    System.out.println( sayi1 + sayi2 );  
}
```

**4- method ismi :** Method ismi olarak istedigimiz ismi secebiliriz, ancak method'un islevi ile isminin uyumlu olması tercih edilir.

Method isimleri kucuk harfle baslar ve camelCase kuralina uygun olur.

**5- parametre :** ( ) icerisine yazilan variable'lardir. Bir method cagrildigi zaman (method call) parametrelerine uygun argument'ler ile cagrilmalidir.

Java, herhangi bir method call ile karsilastiginda once method call'daki argument'ler ile method'daki parametre'leri karsilastirir uyumlu degilse CTE verir.

```
public static int sayiToplama(int sayi1, int sayi2){  
    return sayi1+sayi2;  
}  
  
public static void main(String[] args) {  
  
    int sonuc= sayiToplama(20,30);  
}
```



# Method Olusturma ve Kullanma

6- method body : Suslu parantezler arasında kalan ve kodlarimizi yazdigimiz bolumdur.

Soru : Method nerede olusturulabilir ?

Cevap : Class icerisinde, main method veya diger var olan method'larin disinda olmalidir.

```
public class asd {  
  
    public static int sayiToplama(int sayi1, int sayi2) {  
        return sayi1+sayi2;  
    }  
  
    public static void main(String[] args) {  
  
        int sonuc= sayiToplama(20,30);  
  
    }  
}
```

Method'larin main method'dan once veya sonra olmasinin farki yoktur.

Method'lar cagrilmadan calismaz, cagriliyor da nerede olursa olsun Java bulup calistirir.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-17

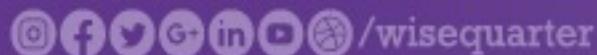
Method **Olusturma ve Kullanma**  
**Method Overloading**



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# Method Olusturma ve Kullanma

**Soru 1-** Kullanicidan input olarak verilen bir String, baslangic ve bitis indexlerine gore baslangic index'i dahil, bitis index'i haric olacak sekilde aradaki harfleri yazdiran bir method olusturun.

- kullanici baslangic degeri olarak, bitis degerinden buyuk bir sayi girerse, hata mesaji verin
- kullanici str'da olan index'lerden daha buyuk bir index girerse hata mesaji yazdirin.

**Soru 2-** Kullanicidan main method icinde ayri ayri isim ve soyismini alin Isim ve soyismi ilk harfleri buyuk diger harfler kucuk olacak sekilde duzenleyip, isim bosluk soyisim seklinde bize donduren bir method olusturun  
**input :** isim : Ali soyisim :YILMAZ.    **output :** Ali Yilmaz

**Soru 3-** Kullanicidan main method icinde pozitif bir tamsayi alin. Girilen sayinin asal sayi olup olmadigini kontrol edip, sonuc olarak “asal sayi” veya “asal sayi degil” sonuclarini donduren bir method olusturun.

**Soru 4-** Kullanicidan main method icinde bir tamsayi alin. Girilen sayinin pozitif tam bolenleri sayisini bulup bize donduren bir method olusturun.

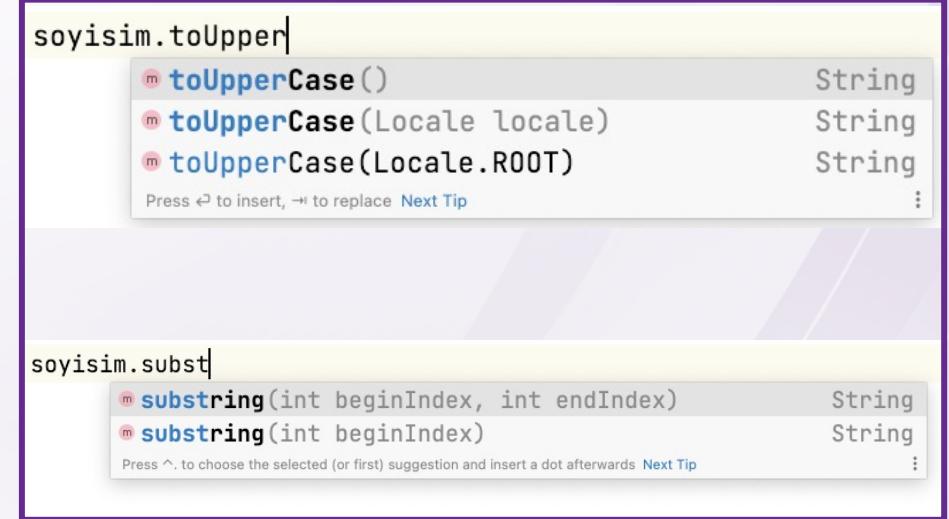
# Method Overloading

**Method overloading** : Bir class'da ayni isimde fakat farkli method signature'ina sahip methodların bulunmasıdır.

**Method overloading'in amacı** ayni islevi farklı parametrelerle, farklı şekilde gerçekleştirebilmektir.

Ornegin, String'deki substring method'unda

- 1 parametre girersek o index'den sona kadar olan metni verir
- 2 parametre girersek, 1.index dahil, 2.index haric olmak üzere aradaki metni verir.



Boylece kullanıcıya ihtiyacına uygun method'u kullanma imkani vermiş oluruz.

# Method Overloading

Bir class'da aynı isimde birden fazla method oluşturabilmek için signature'larini degistirmek gereklidir..

**Method signature :** Method ismi, parametre sayisi ve parametrelerin dizilisi demektir.

Method overloading için ismin  
aynı olması gereklidir,  
signature degistirmek için iki  
yontem kalır.

- 1- parametre sayisini degistirmek
- 2- parametrelerin data turunu veya  
data turu farklı olan parametrelerin  
yerlerini degistirmek.  
(aynı data turundeki parametrelerin  
yerini degistirmek signature'i  
degistirmez)

```
System.out.println(carpim(2,3)); // int int 6
System.out.println(carpim(2,3.4)); // double double 6.8
System.out.println(carpim(3,4,5)); // double double double 60.0
}

public static double carpim(double sayi1, double sayi2){
    return sayi1*sayi2;
}
public static int carpim(int sayi1, int sayi2){
    return sayi1*sayi2;
}

public static double carpim(double sayi1, double sayi2,double sayi3){
    return sayi1*sayi2*sayi3;
}
```

# Method Overloading

Bir class'da ayni isimde birden fazla method oldugunda Java hangisini kullanacagini karar vermek icin

1- Oncelikle method ismi ve parametre sayisina bakar.

2- Ayni isim ve parametre sayisinda birden fazla method varsa, argument ve parametrelerin uyumuna bakar.

- Argumentlerle parametrelerin %100 uyustugu method varsa onu kullanir.

- %100 uyumlu parametre bulamazsa casting ile calisacak method'lara bakar  
- casting ile calisacak method birden fazla ise en az casting yapacagini secer.

```
System.out.println(carpim(2,3)); // int int 6
System.out.println(carpim(2,3.4)); // double double 6.8
System.out.println(carpim(3,4,5)); // double double double 60.0
}

public static double carpim(double sayi1, double sayi2){

    return sayi1*sayi2;
}
public static int carpim(int sayi1, int sayi2){

    return sayi1*sayi2;
}

public static double carpim(double sayi1, double sayi2,double sayi3){

    return sayi1*sayi2*sayi3;
}
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-18

### While Loop

### Do While Loop

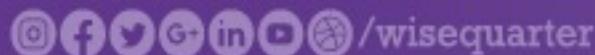
### Scope



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# While Loop

For Loop, belirli sayıda çalıştırılması gereken bir döngüyü verimli bir şekilde yazmanıza olanak tanıyan bir tekrar kontrol yapısıdır.

For loop kullanırken ihtiyacımız olan

- baslangic degeri,
- bitis sarti (condition)
- artis/azalis yontemi

bilgilerine while loop icin de ihtiyac duyuyoruz,  
ama java bunlari otomatik yazmadigi icin  
manuel olarak yazmamiz gerekir.

```
int s=10;  
  
while(s<100){  
    // calisacak kodlar  
    s++;  
}
```

```
for (int i = 0; i <100 ; i++) {  
    // calisacak kodlar  
}
```

Baslangic degerini ve artis degerinin manuel  
yazilmasi, while loop'u baslangicta kullanissiz gibi  
gosterebilir.

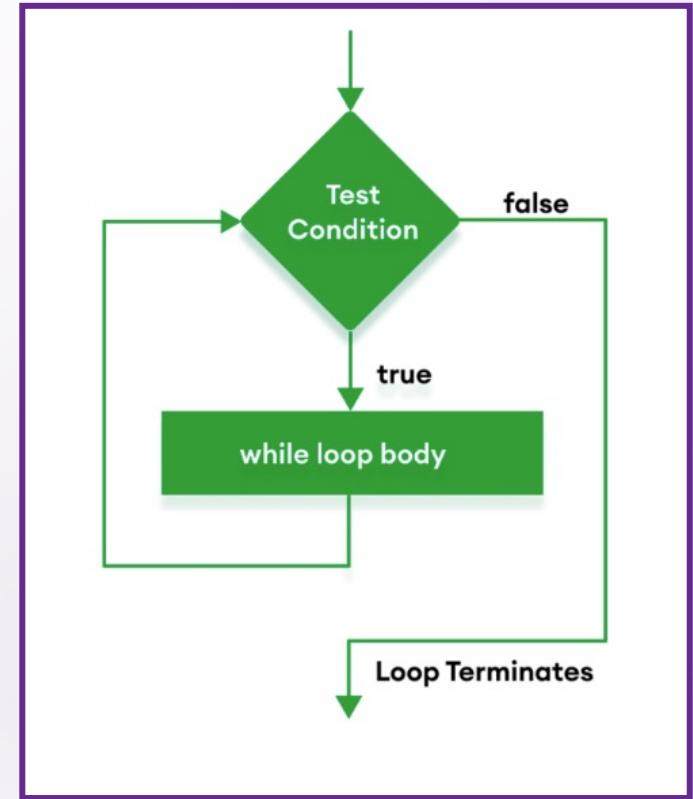
Ozellikle artis/azalis miktarı yazilmazsa kod sonsuz  
loop'a girecek ve bize sorun cikaracaktir.

# While Loop

Ancak while loop bazi durumlarda for loop'dan avantajli olacaktir.

Bir loop'un kac kere calisacagi belli degilse, veya bitis sarti loop degiskene degil, baska bir degiskene bagli ise while loop daha kullanisli olacaktir.

Ornegin kullanicidan sifre istiyorsak ve yanlis giris oldugu muddetce tekrar istememiz gerekiyorsa adim sayisini bilmemiz mumkun olmadigindan while loop tercih edilebilir.



Veya kullanici istedigi muddetce kodumuzun ayni islemi yapmasini istiyorsak, kodun durmasini kullanicinin girecegi "**cikis icin O'a basiniz**" gibi bir degere baglayip, kodu tekrar tekrar calistirabiliriz.



# While Loop

Soru : Kullanicidan toplanmak üzere sayilar isteyin toplam 500 olur veya gecerce toplami yazdirin.

```
Scanner scan= new Scanner(System.in);
int sayi=0;
int toplam=0;

while (toplam<=500){
    System.out.println("Lutfen toplamak üzere sayı girin");
    sayi= scan.nextInt();
    toplam +=sayi;
}

System.out.println("girilen sayıların toplam : "+ toplam);
```

# While Loop

**Soru :** Kullanicidan Kullanicidan sifre isteyin asagidaki şartlari saglamayan sifrelerde hatalari yazdirip, kullanicinin yeni sifre girmesi isteyin Gecerli bir sifre yazilincaya kadar bu islemi tekrar edin gecerli sifre yazilinca “sifreniz basari ile kaydedildi” yazdirin

sartlar :

- sifrenin ilk karakteri kucuk harf olmali
- sifrenin son karakteri sayı olmali

```
Scanner scan = new Scanner(System.in);
boolean sifreDogrumu=false;
String sifre="";
char ilkHarf;
char sonHarf;

while(!sifreDogrumu){ // sifreDogrumu==false

    System.out.println("Lutfen sifre giriniz");
    sifre= scan.nextLine();
    ilkHarf=sifre.charAt(0);
    sonHarf=sifre.charAt(sifre.length()-1);

    if (ilkHarf<'a' || ilkHarf>'z'){
        System.out.println("sifrenin ilk harfi kucuk harf olmali");

    }else if(sonHarf<'0' || sonHarf>'9'){
        System.out.println("sifrenin son karakteri rakam olmali");

    }else{
        System.out.println("Sifre basari ile kaydedildi");
        sifreDogrumu=true;
    }
}
```

# While Loop

**Soru 1-** While loop kullanarak 2 basamakli 7 ile bolunebilen pozitif tamsayıları yazdırın.

**Soru 2-** While loop kullanarak kullanıcıdan alınan sayının rakamlar toplamını bulun.

**Soru 3-** While loop kullanarak verilen bir String'i terse çevirip, bu halini bize donduren bir method oluşturun.

**Soru 4-** Kullanıcıdan toplanmak üzere pozitif tamsayılar isteyin Kullanıcıya bitirmek istediginde 0'a basmasını söyleyin

Kullanıcı bitirmek istediginde toplam kaç adet pozitif tam sayı girdiginive bunların toplamının kaç olduğunu yazdırın

Kullanıcı negatif sayı girerse "negatif sayı kullanamazsınız" yazdırın bu negatif sayiyi sayı adedine ve toplama eklemeyin

**Soru 5-** Kullanıcıdan bir sayı ve hesaplamak istediği ussunu isteyin. While loop kullanarak verilen sayının istenilen ussunu hesaplayıp yazdırın bir method oluşturun.



# Do While Loop

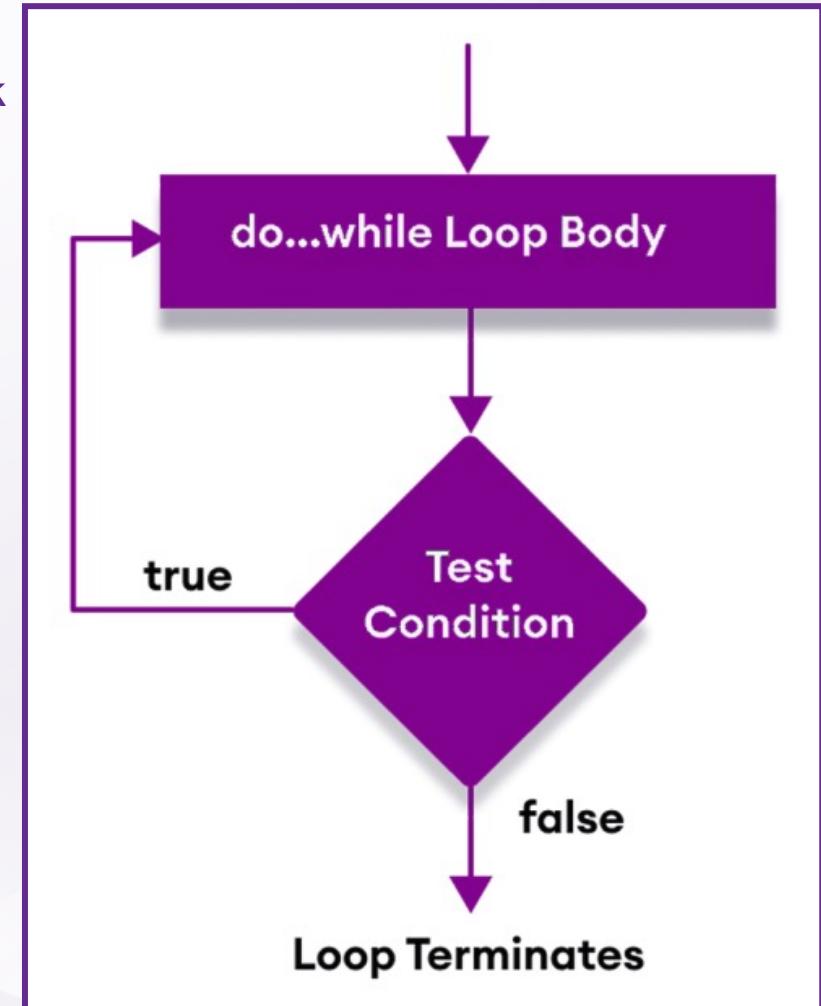
do-while loop, belirtilen koşul doğru olana kadar programın bir bölümünü tekrar tekrar calistirmak için kullanılır.

Java do-while döngüsüne **çıkış kontrol** döngüsü denir.

While döngüsü ve for döngüsünden farklı olarak do-while, döngü gövdesinin sonundaki koşulu kontrol eder.

Java do-while döngüsü, döngü gövdesinden sonra koşul kontrol edildiğinden en az bir kez calisir.

Tekrar sayısı belirli değilse ve döngüyü **en az bir kez çalıştırmanız** gerekiyorsa, bir do-while döngüsü kullanmanız önerilir.



# Do While Loop

**Soru 1-** 'k' harfinden 't' harfine kadar harfleri yazdirin.

**Soru 2-** Kullanicidan bir sifre girmesini isteyin. Girilen sifreyi asagidaki sartlara gore kontrol edin ve sifredeki hatalari yazdirin.

Kullanici gecerli bir sifre girinceye kadar bu islemi tekrar edin ve gecerli sifre girdiginde “Sifreniz Kabul edilmistir” yazdirin.

- Sifre kucuk harf icermelidir
- Sifre buyuk harf icermelidir
- Sifre ozel karakter icermelidir
- Sifre en az 8 karakter olmalidir.

**Soru 3-** Kullanicidan bir pozitif sayi isteyin, sayinin tam kare olup olmadığını bulunuz, tamkare ise true değilse false yazdiriniz.

Ornek : input : 16, output: 4



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-19

Scope

Arrays



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

Scope bir variable'in erisilebildigi alandir.

Scope, variable'in olusturuldugu yer goz onunde bulundurularak 2 ana gruba ayrilir.

### 1- Local variable'lar

Local variable'lar bir method veya kod blogu icerisinde olusturulan variable'lardir.

Local variable'larin scope'u icerisinde olusturulduklari kod blogudur ve o blogun disinda kullanilamazlar.

### 2- Class level variable'lar

Class level variable'lar method ve kod bloklarinin disinda olusturulurlar ve scope'lari tum class'dir.

Ancak static keyword kullanilip kullanilmamasina gore erisimleri ve kullanimlari farkli olur.

# Scope

```
public class Scope {  
  
    static String hastaneIsim;  
    String persIsim;  
    String persSoyisim;  
    String persNo;  
    int cihazNo;  
    String servis;  
  
    public static void main(String[] args) {  
  
        int sayi=10;  
  
        for (int i = 0; i <20 ; i++) {  
            System.out.println(i);  
        }  
    }  
  
    public void method1(){  
        String str="Java";  
    }  
}
```

# Scope

## 1- Local variables :

Local variable'lar bir method veya kod bloğu içerisinde oluşturulan variable'lardır.

Local variable'ların scope'u içerisinde oluşturdukları kod blogudur ve o blok içerisinde kullanılabılırler.

Ancak scope'lari disinda kullanılamazlar. Kullanmak isterseniz CTE olusur.

Tum method'larda kullanmak istediginiz variable'lari class level'da oluşturmalisiniz.

```
public static void main(String[] args) {  
  
    int sayi=10;  
  
    System.out.println(str);  
  
    sayi++;  
    System.out.println(sayi);  
}
```

```
public void method1(){  
    String str="Java";  
    str=str.toUpperCase();  
    System.out.println(str);  
    System.out.println(sayi);  
}
```

# Scope

## 1- Local variables :

Local variable'lar deklare edilirken değer atanma mecburiyeti yoktur.

Java variable'i olusturdugumuzu ama değer atamasını ileriki satırlarda yapacağımızı kabul eder ve CTE vermez.

Ancak local variable'lara değer ataması yapmadan kullanmaya kalkışırsanız Java **olmayan değeri** kullanamayacağı için CTE verir.

```
public static void main(String[] args) {  
  
    int sayi;  
  
    sayi++;  
  
}
```

```
public void method1(){  
    String str;  
  
    System.out.println(str);  
}
```



# Scope

## 1- Local variables :

### Loop variables

Bir loop içerisinde olusturulan variable'larin scope'u olusturuldukları loop'tur, yani o loop disindan kullanilamazlar.

```
public static void main(String[] args) {  
  
    for (int i = 0; i < 10 ; i++) {  
        int sayi= 20;  
        System.out.println(i+ sayi);  
    }  
  
    sayi++;  
  
    System.out.println(i);  
}
```

**NOT :** Her ne kadar bir method içerisinde olsa da loop variable'larinin scope'u içinde bulundukları **method degil**, içerisinde olusturuldukları loop'tur.



# Scope / Class Level Variables



Doktor



Hemsire



Labaratuvar ve personel



Hasta



Tibbi cihazlar

# Scope / Class Level Variables

## 2- Class level variables

Class level variables variable'lar instance ve static olmak üzere ikiye ayrılırlar.

Class level'da oluşturulacak bir variable'in static veya instance yapılmasına o variable'in class'dan oluşturulacak objeler ile ilişkisine bakılarak karar verilir.



Hemsire

Hastane adı, hastane adresi, telefonu gibi bilgiler tüm objeler için ortaktır ve her bir obje için ayrı ayrı atama yapılmasına gerek yoktur.

Ancak, personel adı, personel adresi, telefonu veya cihaz no vs.. gibi bilgiler objelere özeldir ve her obje için birbirinden farklı olabilir.



Tibbi cihazlar

# Scope / Class Level Variables

## 2- Class level variables

Yandaki sema incelenirse hangi variable'larin **tum objeler icin ortak oldugu** dolayisiyla **static** olmasi gerektigi

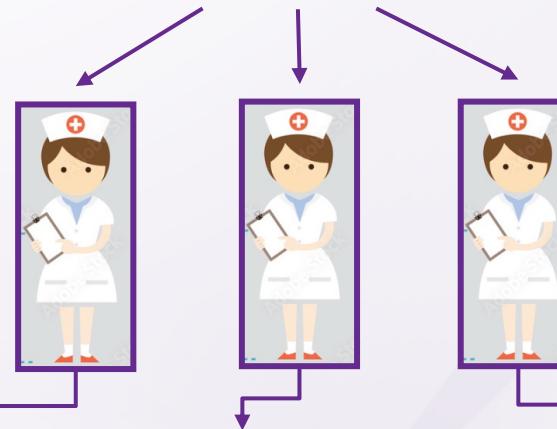
hangi variable'larin ise **tum objeler icin objeye ozel** oldugu, dolayisiyla **static olmamasi** gerektigi asikardir.



H.Ismi : Yildiz

H.adres: Ankara/Cankaya

H.Telefon : 2445566



Hemsire1

P1.Ismi : Yildiz

P1.adres: Cankaya

P1.Tel : 4164352

Hemsire2

P2.Ismi : Ayse

P2.adres: Sincan

P2.Tel : 6151232

Hemsire3

P3.Ismi : Fatma

P3.adres: Altindag

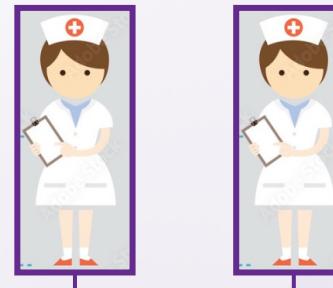
P3.Tel : 7141536

# Scope / Class Level Variables

## 2- Class level variables

Static (**class**) variables : Class'a ait **tek bir variable** olusturulur, tum objeler icin bu variable'in degeri ortaktir.

Instance (**object**) variables : Class level'da olusturulur, ancak her bir object olusturuldugunda Java ilk atanan degerlere sahip yeni bir variable olusturup objeye baglar. Dolayisiyla Java instance bir variable icin **obje sayisinda kopya variable'lar** olusturur.



H.Ismi : Yildiz	H.adres: Ankara/Cankaya	H.Telefon : 2445566
Hemsire2 P2.Ismi : Ayse P2.adres: Sincan P2.telefon : 6151232	Hemsire3 P3.Ismi : Fatma P3.adres: Altindag P3.telefon : 7141536	

# Scope / Class Level Variables

## 2- Class level variables

Kural 1 : Static veya instance variable'lara deger atama mecburiyeti yoktur.

Biz deger atamasi yaparsak, o degeri kullanir. Deger atamazsak Java variable'lar icin default olarak tanimlanan degerleri assign eder ve onlari kullanir.

Default degerler :

String : null

Sayisal primitive'ler : 0

Char : hiclik

Boolean : false

```
static String hastaneIsim;
static String hastaneTel="03123454354";
String persIsim;
String persSoyisim="Soyisim belirtildi";
boolean izindeMi;
int yas;

public static void main(String[] args) {

    System.out.println(hastaneIsim); // null
    System.out.println(hastaneTel); // 03123454354

    Scope per1=new Scope();
    System.out.println(per1.persIsim); // null
    System.out.println(per1.persSoyisim); // Soyisim belirtildi
    System.out.println(per1.izindeMi); // false
    System.out.println(per1.yas); // 0
}
```

# Scope / Class Level Variables

**Kural 2 :** Static variable'lar, static olduklari icin tum class'dan direk kullanilabilirler,  
(tum static method'lар ve static olmayan method'lardan)

Instance variable'lar static olmadiklari icin static method'lardan direk kullanilamazlar.

Instance variable'lara static method'lardan ulasmak ve/veya kullanmak icin obje olusturmamiz gerekir.

Instance variable'lari static olmayan method'lardan direk kullanabiliriz.

```
static String hastaneIsim;
static String hastaneTel="03123454354";
String persIsim;
String persSoyisim="Soyisim belirtildi";
boolean izindeMi;
int yas;

public static void main(String[] args) {

    System.out.println(hastaneIsim); // null
    System.out.println(hastaneTel); // 03123454354

    Scope per1=new Scope();
    System.out.println(per1.persIsim); // null
    System.out.println(per1.persSoyisim); // Soyisim belirtildi
    System.out.println(per1.izindeMi); // false
    System.out.println(per1.yas); // 0
}
```

# Scope / Class Level Variables

## 2- Class level variables

Kural 3 : Static variable'lara, baska class'lardan erismek icin `classAdi.staticVariableAdi` yazmamiz yeterlidir.

Instance variable'lara baska class'dan ulasmak ve/veya kullanmak icin obje olusturmamiz gereklidir.

```
public class Scope {  
  
    static String hastaneIsim;  
    static String hastaneTel="03123454354";  
    String persIsim;  
    String persSoyisim="Soyisim belirtildi";  
    boolean izindeMi;  
    int yas;
```

```
public class Runner {  
    public static void main(String[] args) {  
  
        System.out.println(Scope.hastaneIsim); // null  
        System.out.println(Scope.hastaneTel); // 03123454354  
  
        Scope per1=new Scope();  
        System.out.println(per1.persIsim); // null  
        System.out.println(per1.persSoyisim); // Soyisim belirtildi  
        System.out.println(per1.izindeMi); // false  
        System.out.println(per1.yas); // 0  
    }  
}
```



Scope : Class icerisinde olusturulan variable'larin kapsamini (nereden erisebilecegini) belirler

Temel olarak 4 Scope'dan bahsedebiliriz

Class Level'da olusturulan variable'lar class'in tamaminda gecerlidir, ancak direk erisim icin static keyword belirleyicidir

1- static olarak tanimlanan variable'lara tum method'lardan ulasilabilir

2- static olarak tanimlanmayan (instance) variable'lara sadece static olmayan method'lardan ulasilabilir

Local olarak olusturulan variable'lar sadece tanimlandiklari scope'da gecerlidirler.  
(Herkes oturdugu mahallede taninir)

3- bir method'da olusturulan variable'lara sadece o method'dan ulasilabilir

4- Loop icerisinde olusturulan variable'a loop disindan erisilemez

```
2 ► public class ScopeNedir {  
3 →     static int sayi=5;  
4 →     String ders="Java";  
5 ►     public static void main(String[] args) {  
6         sayi=100;  
7         ders="Java Course";  
8         int mainsayi=20;  
9         ders2="API";  
10        for (int i = 0; i < 10; i++) {  
11            System.out.println(i);  
12            String ders3="SQL";  
13        }  
14        System.out.println(i);  
15        ders3="API";  
16    }  
17    public static void staticMethod(){  
18        sayi=110;  
19        System.out.println(ders);  
20        mainSayi=10;  
21        System.out.println(ders2);  
22    }  
23    public void staticOlmayanMethod(){  
24        System.out.println(sayi);  
25        ders="Java Course";  
26        System.out.println(mainSayi);  
27        String ders2="Selenium";  
28    }
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-20

### Arrays



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Scope / Ozet

Soru : Yandaki class için aşağıdaki soruları yanıtlayın.

1- hangi satırlarda local variable'lar vardır ?

2- class level'da oluşturulan variable'ların scope'ları ve değerleri nelerdir ?

3- Hangi satırlarda CTE vardır ve düzeltmesi gereklidir ?

```
3 public class Scope {  
4  
5     static String str;  
6     String tel="03123454354";  
7  
8     public static void main(String[] args) {  
9  
10        Scope obj=new Scope();  
11        System.out.println(tel);  
12        obj.str="Java ne guzel";  
13        int sayi=15;  
14        method2(sayi);  
15        method1();  
16    }  
17  
18    public void method1(){  
19        tel="03124324343";  
20        String isim= "John Doe";  
21        boolean dogruMu;  
22        int sayi;  
23    }  
24  
25    public static void method2(int sayi){  
26        str=str+".";  
27        tel=tel.substring(1);  
28        int sayi=10;  
29    }  
30}
```

# Arrays

Bugune kadar kullandigimiz data turleri sadece 1 variable'a 1 deger atamasi yapabiliyordu.

```
int sayi=20;  
String str="Java Candir";  
char ilkHarf='A';  
boolean aktifMi=true;
```

Ancak Java gibi kompleks uygulamalar geliştirmeye uygun bir programlama dilinde birden fazla eleman barindirabilen yapılara da ihtiyac vardır.

Ornegin; bir sınıfındaki öğrenci listesini oluşturmak, online satış yapan bir uygulamadaki satılan ürünlerin listesini tutmak için birden çok eleman barındıran yapılar gereklidir.

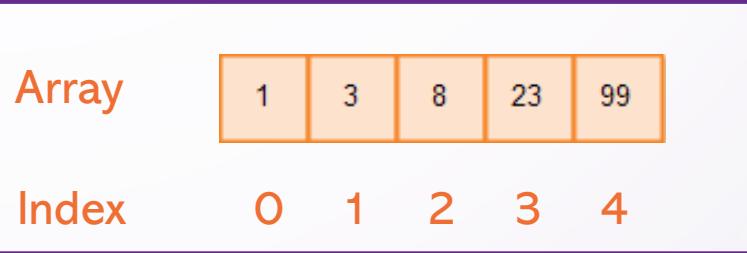
Array(dizi)'ler Java'da **ayni data turunden** birden fazla eleman barindirabilen **objelerdir**.

Array'ler her bir eleman için ayrı bir alan ayıırlar ve bu elemanlara index bilgisi ile ulaşabiliriz.

Array	1	3	8	23	99
Index	0	1	2	3	4



# Arrays



1- Bir array olusturulurken 2 sey deklare edilmek zorundadir.

A- Icine konulacak elementlerin data turu

Bir array icinde farkli data turundan element olamaz.

Array'ler primitive data turundeki datalarin degerlerini, non-primitive data turundeki datalarin ise referanslarini barindirirlar.

```
int [ ] arr = new int [5];
```

B- Icine kac element konulacagi (length)

Olusturulan bir array'in uzunlugu sabittir, degistirilemez.



# Arrays

## 2- Bir array 2 sekilde deklare edilir

- A- `int arr[];` IntelliJ sariya boyar
- B- `int[] arr;`

Deklare ettik ama deger atamadik, java **referansi** olusturur ama length belli olmadigindan **objeyi** olusturamaz.

## 3- Bir array'e 2 sekilde deger atanabilir

- A- Direk degerler atanabilir    `int[] arr={1,3,8,23,99};`    `[1, 3, 8, 23, 99]`

- B- Uzunlugu belirtilerek olusturulur ama elemanlara deger atamasi yapilmaz.

Bu durumda Java belirtilen uzunlukta ve default degerlere sahip bir array olusturur.

`int[] arr= new int[5];`

`[0, 0, 0, 0, 0]`

# Arrays

## 4- Bir array'in **length**'i nasıl bulunur ?

```
int[] arr= {1,3,8,23,99};  
  
System.out.println(arr.length); // 5
```

**NOT:** Array'deki **length** bir method degil uzunlugu tutan bir variable'dir.

Dolayisiyla yaninda ( ) parantez yoktur.

## 5- Bir array'de **istenen index**'deki elemana nasıl ulasılır ?

Array'deki herhangi bir elemana ulasmak veya degistirmek icin index kullanilir.

Array'de olmayan bir index'i kullanmak isterseniz ArraysIndexOutOfBoundsException exception verir.

```
int[] arr= {1,3,8,23,99};  
  
System.out.println(arr[2]); // 8  
arr[2]=10;  
System.out.println(arr[2]); // 10
```

# Arrays

6- Bir array'in **tum elemanları** nasıl yazdırılır ?

For-Loop kullanalım.

```
int[] arr= {1,3,8,23,99};  
  
for (int i = 0; i < arr.length ; i++) {  
    System.out.print(arr[i]+ " ");  
}
```

1 3 8 23 99

7- Bir **array** nasıl yazdırılır ?

Array direkt yazılmaz, direkt yazdırmak isterseniz array'i değil referansını yazdırır.

Array'i yazdırmak için Arrays class'ından `toString()` method'u kullanmalıdır.

```
int[] arr= {1,3,8,23,99};  
  
System.out.println(arr); // [I@2752f6e2
```

```
int[] arr= {1,3,8,23,99};  
  
System.out.println(Arrays.toString(arr)); // [1, 3, 8, 23, 99]
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-21

### Arrays



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# Arrays

- Soru 1-** Verilen bir int array'in tum elemanlarini 2 artirip bize donduren bir method olusturun. Eski array'i yeni haliyle kaydedin.
- Soru 2-** Verilen bir array'deki pozitif tamsayiları toplayip sonucu bize donduren bir method yaziniz.
- Soru 3-** Verilen bir array'deki tum elementleri bir saga kaydirip, sondaki elementi de basa tasiyacak bir method olusturun, array'i yeni haliyle kaydedin.  
Orn : input : [4,5,6,7] array'in son hali. : [7,4,5,6]
- Soru 4-** Verilen bir array'de istenen bir elemanın var olup olmadigini ve varsa kac kere kullanildigini yazdiran bir method olusturun.
- Soru 5-** Kullanicidan array'in boyutunu ve elementlerini alip array'i olusturan ve bize donduren bir method olusturun.
- Soru 6-** Verilen String bir array'deki en uzun ve en kisa kelimeleri yazdiran bir method olusturun.
- Soru 7-** Verilen bir array'e istenen bir elemani ekleyip bize donduren bir method yazin, eski array'e yeni degeri atayin.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-22

### Arrays

### Multi Dimensional Arrays



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Arrays

8- Bir array'in elemanlari nasıl sıralanır ?

```
int[] arr={1,3,9,5,4,6};
```

```
System.out.println(Arrays.toString(arr)); // [1, 3, 9, 5, 4, 6]
```

Array'lerde sıralama yapmak için Arrays class'ından yardım almak gereklidir.

```
Arrays.sort(arr);
```

```
System.out.println(Arrays.toString(arr)); // [1, 3, 4, 5, 6, 9]
```

Arrays.sort( ) method'u array'i Natural Order'a göre sıralar.

Buyukten kucuge sıralamak isterseniz, sort( ) ile sıralayıp, loop ile tersine cevirmek gereklidir.



# Arrays

## 9- Bir array'de istenen elemanın varlığı nasıl kontrol edilir ?

Array'lerde istenen bir elemanın varlığını kontrol etmek için **binarySearch( )** method'u kullanılır.

Ancak, **binarySearch( )** kullanmadan önce **MUTLAKA** **sort( )** ile sıralama yapılmalıdır.

```
int[] arr={1,3,9,5,4,6};

System.out.println(Arrays.toString(arr)); // [1, 3, 9, 5, 4, 6]

Arrays.sort(arr);

System.out.println(Arrays.toString(arr)); // [1, 3, 4, 5, 6, 9]

System.out.println(Arrays.binarySearch(arr, key: 1)); // 0

System.out.println(Arrays.binarySearch(arr, key: 6)); // 4
```



# Arrays

Arrays.sort( ) kullanilmadan da binarySearch( ) calisir ancak sonuclarinin ne olacagi BILINEMEZ.

```
int[] arr={1,3,9,5,4,6};

System.out.println(Arrays.toString(arr)); // [1, 3, 9, 5, 4, 6]

System.out.println(Arrays.binarySearch(arr, key: 1)); // 0

System.out.println(Arrays.binarySearch(arr, key: 6)); // -3

System.out.println(Arrays.binarySearch(arr, key: 8)); // -3
```

binarySearch( ) ile array'de olmayan bir eleman aranrsa, olmasi gereken sirayi – ile VERIR.

```
int[] arr={1,3,9,5,4,6};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr)); // [1, 3, 4, 5, 6, 9]

System.out.println(Arrays.binarySearch(arr, key: -3)); // -1

System.out.println(Arrays.binarySearch(arr, key: 7)); // -6

System.out.println(Arrays.binarySearch(arr, key: 18)); // -7
```



# Arrays

## 10- İki array'in eşitliği nasıl kontrol edilir ?

Arrays.equals(arr1,arr2); her bir index için elemanları kontrol eder, tüm index'lerdeki değerler eşit ise **true**, farklılık varsa **false** döner.

```
int arr1[] = {2, 1, 7, 6};  
int arr2[] = {7, 1, 6, 2};  
System.out.println(Arrays.equals(arr1, arr2));
```

→ **false**

```
int arr3[] = {3, 2, 7, 8, 11};  
int arr4[] = {7, 3, 8, 2, 12};  
Arrays.sort(arr3);  
Arrays.sort(arr4);  
System.out.println(Arrays.equals(arr3, arr4));
```

→ **false**

```
int arr5[] = {4, 2, 6, 8, 11};  
int arr6[] = {11, 4, 8, 2, 6};  
Arrays.sort(arr5);  
Arrays.sort(arr6);  
System.out.println(Arrays.equals(arr5, arr6));
```

→ **true**



# Arrays

## 11- Bir String'i array'e cevirmek

str.split( **StringAyirac** ); bir String'i istedigimiz parcalara ayirarak bir array'e cevirir.

```
String str= "Ali topu at, at Ali at";

String[] arrVirgul=str.split( regex: "," );
System.out.println(Arrays.toString(arrVirgul));
// [Ali topu at, at Ali at]

String[] arrSpace=str.split( regex: " " );
System.out.println(Arrays.toString(arrSpace));
// [Ali, topu, at,, at, Ali, at]

String[] arrHiclik=str.split( regex: "" );
System.out.println(Arrays.toString(arrHiclik));
// [A, l, i, , t, o, p, u, , a, t, , , a, t, , A, l, i, , a, t]
```

# Arrays

ONEMLI NOT : Varolan bir array'e yeni deger atanabilir mi ?

Varolan bir array'e elementleri yazarak yeni deger atamasi yapamayiz

```
int[] arr= {1,3,8,23,99};  
  
arr= new int[3]; // [0, 0, 0]  
  
int[] arr2=new int[4];  
  
arr2=new int[6]; // [0, 0, 0, 0, 0, 0]
```

```
int[] arr= {1,3,8,23,99};  
  
arr= {1,2,3};  
  
int[] arr2=new int[4];  
  
arr2={3,4,5};
```

Ancak new keyword ile yeni bir deger atayabiliriz.

Bu varolan array'in uzunlugunu degistirmek degil, yepeni bir array olusturmak oldugundan Java CTE vermez.



# Arrays

Soru :

What is the result of the following?

```
int[] random = { 6, -4, 12, 0, -10 };  
int x = 12;  
int y = Arrays.binarySearch(random, x);  
System.out.println(y);
```

- A.** 2
- B.** 4
- C.** 6
- D.** The result is undefined.
- E.** An exception is thrown.
- F.** The code does not compile.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-23

### Multi Dimensional Arrays



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Onceki Gunden Aklimizda Kalanlar

- 1- Array : Array'ler ayni data turunden birden fazla element barindiran java yapılarindandır.
- 2- Birden fazla element barindirdigi icin elementlerine direk ulasabilmemiz icin index kullanmamız gereklidir. Bir elemente ulasmak veya yeni deger atamak istersek arr[index] seklinde array ismi ve oindex'ini yazabiliriz.
- 3- sout (arr[3]) → bize bir element yazdiracagi icin direkt yazdirabiliriz, ancak tum array'i yazdirmak istersek sout(Arrays.toString(arr)) seklinde yazdirabiliriz. Eger toString( ) kullanmazsak array'l degil array'in referansini yazdirir.
- 4- Bir array'in tum elementlerine erismek, kontrol etmek, update etmek veya yazdirmak istersek bir forLoop kullanabiliriz. For-Loop içerisindeki arr[i] bize sirasiyla tum elementleri getirir, biz de bu elementler üzerinde istedigimiz islemi yapariz.
- 5- iki array'in esit olabilmesi icin hem ayni elementlere sahip olmalı, hem de ayni indexdeki elementler esit olmalı. Bundan dolayi 2 array'in esitligini kontrol etmeden once sort( ) kullanmakta fayda vardır.
- 6- Bir array'de istenen bir elementin var olup olmadigini Arrays.binarySearch(arr , elm) ile kontrol edebiliriz. Ancak binaryTree yapisindan dolayi binarySearch( ) calistirilmadan once MUTLAKA sort( ) calistirilmalidir. Sort( ) calistirilmadan yapilacak binarySearch( )'un sonucu ONGORULEMEZ. binarySearc( ) element varsa indexini, element yoksa – isareti ile birlikte olsaydi hangi SIRADA olacagini döndürür.



# Multi Dimensional Arrays

Multidimensional array'ler, içerisinde array bulunduran array'lerdir.

```
int[][] arr= {{3,1,2,4},{1,2},{3,4,5},{10},{2,7}};
```

Bu sekilde deklare edilmiş olan arr array'ini tek katlı olarak yazmak istersek, length'i 5 olan bir array goruruz.

```
int[] arr= {arr[0], arr[1], arr[2], arr[3], arr[4]};
```

outer array

Multidimensional array'lerde en distaki array'e **outer array**, içerisindeki array'lere ise **inner arrays** denir.

```
arr[0] => {3,1,2,4}  
arr[1] => {1,2}  
arr[2] => {3,4,5}  
arr[3] => {10}  
arr[4] => {2,7}
```

inner arrays



# Multi Dimensional Arrays

Multidimensional array'lerde, bir element'e ulasmak icin once elementin icinde oldugu inner array'e, sonra o inner array'deki index'i kullanarak da elemente ulasmaliyiz.

```
int[][] arr= {{3,1,2,4},{1,2},{3,4,5},{10},{2,7}};
```

0.      1.      2.      3.      4.

0.1. 2.

Ornegin; 5 elementine ulasmak icin once outer array'den inner array'e ulasalim

Sonra inner array'den 5 elementine ulasalim

```
arr[2][2]
```



# Multi Dimensional Arrays

```
int[][][] arr= {{3,1,2[4]}, {1[2]}, {3,4[5]}, {[10]}, {2[7]}};
```

0.

1.

2.

3.

4.

4 elementi icin → arr [0][3]

2 elementi icin → arr [1][1]

5 elementi icin → arr [2][2]

10 elementi icin → arr [3][0]

7 elementi icin → arr [4][1]

# Multi Dimensional Arrays

Yazdirma islemi yapmak istedigimizde, yazdiracagimizin ne oldugunu bilmemiz gerekir.

```
int[][] arr= {{3,1,2,4},{1,2},{3,4,5},{10},{2,7}};
```

0.      1.      2.      3.      4.

1- Array'in icerisinde bulunan elementlerden birini yazdirmak istersek direk yazdirabiliriz.

```
System.out.println(arr[1][1]); // 2
System.out.println(arr[2][0]); // 3
System.out.println(arr[4][1]); // 7
```

2- Inner array'lerden birini yazdirmak istersek Arrays.toString( ) kullaniriz.

```
System.out.println(Arrays.toString(arr[1])); // [1, 2]
System.out.println(Arrays.toString(arr[2])); // [3, 4, 5]
```

3- Outer array'i yazdirmak istersek Arrays.deepToString( ) kullaniriz.

```
System.out.println(Arrays.deepToString(arr));
// [[3, 1, 2, 4], [1, 2], [3, 4, 5], [10], [2, 7]]
```



# Multi Dimensional Arrays

Multi Dimensional Array'in tum elementlerine ulasmamiz gerektiginde, katman sayisi kadar ic ice for loop olusturmaliyiz

Ornegin 2 katli asagidaki array'in tum elementlerini yazdirmak istedigimizde, ic ice 2 loop kullanmalıyız.

```
int[][] arr= {{3,1,2,4},{1,2},{3,4,5},{10},{2,7}};
```

```
for (int i = 0; i < arr.length ; i++) {  
  
    for (int j = 0; j <arr[i].length ; j++) {  
  
        System.out.print(arr[i][j] +" ");  
    }  
}
```

Output :

```
3 1 2 4 1 2 3 4 5 10 2 7
```

# Multi Dimensional Arrays

**Soru 1-** Verilen 2 katlı bir array'de bulunan çift sayıları toplayıp, sonucu yazdırın bir method oluşturun.

**Soru 2-** Verilen 2 katlı bir array'de aynı index'e sahip elementleri toplayıp, yeni oluşturacağımız tek katlı bir array'e bu toplamları atayın.

input : int[][] arr = {{3,4,5}, {2,3,6,7}};

output: [5, 7, 11]

**Soru 3-** Verilen 2 katlı bir array'de her bir iç array'deki elementleri toplayıp, yeni oluşturacağımız tek katlı bir array'e bu toplamları atayın.

input : int[][] arr = {{3,1,2,4},{1,2},{3,4,5},{10},{2,7}};

output: [10, 3, 12, 10, 9]

**Soru 4-** Verilen 2 katlı bir array'de bulunan tüm sayıların çarpını bize döndüren bir method oluşturun.

**Soru 5-** Verilen 2 katlı bir array'de her bir inner array'in son elementlerinin toplamını yazdırın bir method oluşturun.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-24

### ArrayLists



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# ArrayList

ArrayList, dinamik ve yeniden boyutlandirilabilir bir array'dir.

ArrayList'in dezavantajları uzunlugunu en basta belirlemek zorunda olmamız ve uzunluk sabit olmasıdır. Sabit uzunluk esnek çalışmaya imkan tanımadığından array liste tutmakta çok kullanılmaz.

ArrayList array altyapısını kullanmakla beraber element ekleme ve silme işlemlerine açıktır. Ancak array altyapısı sebebiyle eklemeleri birer birer yapabiliriz. ( veya uzunluğu belirli başka bir ArrayList ekleyebiliriz)

```
List<Integer> sayilar= new ArrayList<>();  
sayilar.add(10);  
sayilar.add(20);  
sayilar.add(30);  
  
System.out.println(sayilar);
```

# ArrayLists Olusturma

1- Ileride gorecegimiz üzere List bir class degil interface'dir. Interface'lerden obje olusturulamadiği için de eşitliğin sağında List<>() kullanılamaz.

```
List<Integer> sayilar1 = new List<>();  
  
List<Integer> sayilar2 = new ArrayList<>();  
  
List<Integer> sayilar3 = new ArrayList<Integer>();
```

- 2- List oluşturabilmek için eşitliğin sağ tarafında List interface'sinin child class'i olan ArrayList<>() veya LinkedList<>() kullanılabilir. Biz simdilik ArrayList'i kullanacağız.
- 3- Eşitliğin sağında <> içerisinde data turu yazılması mecburi değildir ama istersek yazabiliriz.

# ArrayList'lar

## 1- ArrayList'e eleman ekleme

```
public static void main(String[] args) {  
  
    List<Integer> sayilar = new ArrayList<>();  
  
    sayilar.add(10); // [10]  
    sayilar.add(20); // [10, 20]  
  
    sayilar.add(index: 1, element: 25); // [10, 25, 20]  
  
    List<Integer> liste = new ArrayList<>();  
    liste.add(101); // [101]  
    liste.add(102); // [101, 102]  
  
    liste.addAll(sayilar); // [101, 102, 10, 256, 20]  
  
    liste.addAll(index: 1, sayilar);  
    // [101, 10, 25, 20, 102, 10, 256, 20]  
}
```

- List'e elemanlar birer birer eklenir ve ekleme sirasina gore listeye yerlestilir.
- List'te istedigimiz bir index'e eleman ekleyebiliriz. Bu durumda ekledigimiz index ve sonrasindaki elementler birer basamak geriye kaydirilir.
- List'tin sonuna istedigimiz bir list'i de ekleyebiliriz
- Eklenecek list, list'tin sonuna degil istedigimiz bir index'e de eklenebilir. Bu durumda yine o index ve sonrasindaki elementler eklenen listenin uzunlugu kadar geriye kaydirilacaktir.



# ArrayList Method'ları

## 2- list'in boyutunu belirleme

list.size( ) Method'u List'in boyutunu bize dondurur.

```
List<Integer> sayilar = new ArrayList<>();
```

```
System.out.println(sayilar.size()); // 0
```

```
sayilar.add(10);  
sayilar.add(20);
```

```
System.out.println(sayilar.size()); // 2
```

## 3- list'in bos olup olmadigini kontrol etme

list.isEmpty( )

```
List<Integer> sayilar = new ArrayList<>();
```

```
System.out.println(sayilar.isEmpty()); // true
```

```
sayilar.add(10);  
sayilar.add(20);
```

```
System.out.println(sayilar.isEmpty()); // false
```



# ArrayList Method'ları

4- list'de bir elementin olup olmadığını kontrol etme. boyutunu belirleme

`list.contains ( arananObj )`

**NOT :** Olusturulacak list uzun ise elementleri tek tek eklemek yerine bir array olusturup, for-loop ile list'e atanabilir.

```
int[] arr={2,3,4,5,3,6,7,3,8,9,1,2,5,3,8,5};  
  
List<Integer> sayilar=new ArrayList<>();  
  
for (int i = 0; i < arr.length ; i++) {  
    sayilar.add(arr[i]);  
}  
  
System.out.println(sayilar.contains(4)); // true
```



# ArrayList Method'lari

## 5- list'den istedigimiz index'deki element'e ulasma

list.get ( istenenIndex )

```
List<String> liste=new ArrayList<>();  
liste.add("Fatih");  
liste.add("Levent");  
liste.add("Esra");  
liste.add("Seher");  
  
System.out.println(liste.get(2)); // esra  
System.out.println(liste.get(1)); // Levent
```

# ArrayList Method'ları

## 6- İstenen elementi update etme

`list.set( istenenIndex , yeniDeger )`

```
List<String> liste=new ArrayList<>();  
liste.add("Fatih");  
liste.add("Levent");  
liste.add("Esra");  
liste.add("Seher");  
  
liste.set(2,"Yasar");  
System.out.println("set'den sonra "+liste);  
// [Fatih, Levent, Yasar, Seher]
```

`list.set( istenenIndex , yeniDeger )` ile `list.add( istenenIndex, yeniElement)` birbirinden farklıdır.

`Add()`'da element silinmez eklemeden sonra kalanlar geriye kaydırılır, `set()` de ise o indexdeki eski değer silinir, yeniisi yazılır, diğer elementlerin yerlerinde bir değişiklik olmaz

# ArrayList Method'ları

## 7- İstenen elementi veya elementleri silme

**list.remove ( istenenObje )** : İstenen objeyi siler ve bize boolean sonuç döndürür.

Silmek istedigimiz obje list'de yoksa list'de bir degisiklik olmaz, remove( ) bize **false** döndürür.

**list.remove ( istenenIndex )** : İstenen indexdeki elementi siler ve bize sonuç olarak silinen elementi döndürür.

Silmek istedigimiz index list'de yoksa exception olusur.

```
List<String> liste=new ArrayList<>();  
liste.add("Fatih");  
liste.add("Levent");  
liste.add("Esra");  
liste.add("Seher");  
  
System.out.println(liste); // [Fatih, Levent, Esra, Seher]  
  
// remove 1- silme islemi icin obje yazilirsa,  
// objeyi siler ve bize boolean sonuc doner  
System.out.println(liste.remove( o: "Fatih")); // true  
System.out.println(liste); // [Levent, Esra, Seher]  
  
System.out.println(liste.remove( o: "Ahmet")); // false  
  
// remove 2- index ile silersek, bize silinen elementi dondurur  
System.out.println(liste.remove( index: 1)); // Esra  
  
System.out.println(liste); // [Levent, Seher]
```

# ArrayList Method'ları

**NOT :** List Integer değerlerden olusuyorsa, remove(**sayı**) yazdığımızda direkt **index** olarak kabul eder ve o index'deki elementi siler.

Istenen Integer objeyi, obje olarak remove etmek isterseniz, oncelikle o objeyi bir variable olarak oluşturup, **list.remove(variableIsmi)** şeklinde kullanmalısınız.

## 8- List'deki tüm elementleri silme

**list.clear()**

```
List<Integer> sayilar = new ArrayList<>();  
  
sayilar.add(10);  
sayilar.add(15);  
sayilar.add(20);  
sayilar.add(2);  
  
System.out.println(sayilar); // [10, 15, 20, 2]  
  
sayilar.remove(index: 2);  
System.out.println(sayilar); // [10, 15, 2]  
  
Integer silinecek=10;  
sayilar.remove(silinecek);  
System.out.println(sayilar); // [15, 2]  
  
sayilar.clear(); // tüm listeyi temizler  
System.out.println(sayilar); // []
```



# ArrayList Method'ları

9- Bir list'de başka bir list'in tüm elementleri silme

**list.removeAll (silinecekList)**

```
List<String> liste=new ArrayList<>();
liste.add("Fatih");
liste.add("Levent");
liste.add("Esra");
liste.add("Seher");

List<String> silinecekListe=new ArrayList<>();
silinecekListe.add("Fatih");
silinecekListe.add("Levent");
```

```
liste.removeAll(silinecekListe);
System.out.println(liste); // [Esra, Seher]
```

```
System.out.println(liste.equals(silinecekListe)); // false
```

10- İki list'in eşit olduğunu kontrol etme

**list.equals (digerList)**



# ArrayList Method'ları

- 11- Bir list'de istenen elementin  
kullanildigi ilk index'i bulma

`list.indexOf (arananElement)`

- 12- Bir list'de istenen elementin  
kullanildigi son index'i bulma

`list.lastIndexOf (arananElement)`

```
List<String> liste=new ArrayList<>();  
liste.add("Fatih");  
liste.add("Esra");  
liste.add("Levent");  
liste.add("Esra");  
liste.add("Seher");
```

```
System.out.println(liste.indexOf("Esra")); // 1
```

```
System.out.println(liste.lastIndexOf("Esra")); // 3
```



# ArrayList Method'ları

## 13- Bir list'deki elementleri sıralama

Collections.sort(list)

```
List<String> liste=new ArrayList<>();  
liste.add("Eyup");  
liste.add("Yahya");  
liste.add("Esra");  
liste.add("Seher");  
  
System.out.println(liste); // [Eyup, Yahya, Esra, Seher]  
  
Collections.sort(liste);  
  
System.out.println(liste); // [Esra, Eyup, Seher, Yahya]
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-25

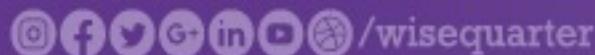
Array Lists  
For-each Loop



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





# ArrayLists

- Soru 1-** Verilen bir array'de tekrar eden elementler icin, mukerrer olanlari silip, tum elemanlardan sadece 1 tane yapip bize dondurecek bir method olusturun.
- Soru 2-** Kullanicidan istedigi kadar isim alip, Q'ya bastiginda girdigi isimleri bize liste olarak dondurecek bir method olusturun.
- Soru 3-** Verilen String bir listede istenmeyen harf iceren elementleri silip, kalan kismini list olarak bize donduren bir method olusturun
- Soru 4-** Verilen pozitif bir n tamsayisini alarak, bize ilk n tane tane Fibonacci sayisini bir list olarak donduren bir method olusturun.
- Soru 5-** Kullanicidan pozitif bir tamsayi alip, o tamsayidan kucuk Fibonacci sayilarini bir liste olarak bize donduren bir method olusturun.
- Soru 6-** Verilen pozitif bir tamsayiyi, tam bolebilen tum pozitif tamsayıları bir liste olarak bize donduren bir method olusturun.



# ArrayList Method'lari

## 14- Bir list'in belirli bir bolumunu alma

`list.subList(baslangicIndex,bitisIndexi)`

```
List<String> liste=new ArrayList<>();  
liste.add("Eyup");  
liste.add("Yahya");  
liste.add("Esra");  
liste.add("Seher");  
  
System.out.println(liste.subList(1, 3)); // [Yahya, Esra]
```



# Array'i ArrayList'e Cevirme

Verilen bir array'i Arrays.asList(array) method'u kullanarak list'e cevirebiliriz.

Ancak bu method'u kullanıssız yapan 2 dezavantaj soz konusudur.

- array'den donusturulen list'de add( ) ve remove( ) gibi uzunlugu etkileyen method'lar kullanıldığında Run Time'da exception olusur

```
Integer[] arr= {2,3,4,5,6};

List<Integer> list= Arrays.asList(arr);

System.out.println(list); // [2, 3, 4, 5, 6]

list.add(10);
list.remove(index: 0);

list.set(1,20);
System.out.println("list'i update ettikten sonra list : " + list);
//[2, 20, 4, 5, 6]

System.out.println("list'i update ettikten sonra array : " + Arrays.toString(arr));
//[2, 20, 4, 5, 6]
```

- asList( ) method'u array ve list'i ozdeslestirir, birinde yapılan degisiklik otomatik olarak digerine de islenir.



# ArrayList'i Array'e Cevirme

Verilen bir list'i Array'e 2 turlu cevirebiliriz.

Array'in data turunu girmek istersek sag tarafta parametre olarak **(new dataTuru[0])** yazmaliyiz

```
List<String> liste=new ArrayList<>();  
liste.add("Eyup");  
liste.add("Yahya");  
liste.add("Esra");  
liste.add("Seher");
```

```
String arr[ ] = liste.toArray(new String[0]);  
System.out.println(Arrays.toString(arr)); // [Eyup, Yahya, Esra, Seher]
```

```
List<Integer> liste2=new ArrayList<>();  
liste2.add(12);  
liste2.add(10);  
liste2.add(5);  
liste2.add(9);
```

```
Integer[] arr2 = liste2.toArray(new Integer[0]);  
System.out.println(Arrays.toString(arr2)); // [12, 10, 5, 9]
```

```
Object[] arr3= liste.toArray();  
Object[] arr4= liste2.toArray();  
System.out.println(Arrays.toString(arr3)); // [Eyup, Yahya, Esra, Seher]  
System.out.println(Arrays.toString(arr4)); // [12, 10, 5, 9]
```

**String[ ] arr = liste.toArray(new String[0]);**

Array'in data turunu Object secersek sag tarafta parametre yazmaya gerek kalmaz

**Object[ ] arr = liste.toArray( );**



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-26

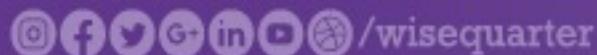
For-each Loop  
Constructors



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





# For-Each Loop

For-Each Loop (**enhanced**) gelismis for loop olarak bilinir ve bir array veya collection'daki **TUM ELEMENTLERİ** index veya siralama olmaksızın bize getirir.

```
int[] arr={2,3,4,6,7,8,1};

for (int i = 0; i <arr.length ; i++) {
    System.out.print(arr[i]+ " ");
}
```

```
int[] arr={2,3,4,6,7,8,1};

for (int each:arr
     ) {
    System.out.print(each + " ");
}
```

**Avantaji :** Index yapısına ihtiyac duymadığından kod yazarken yasanacak hataları azaltır, tüm elementleri eksiksiz olarak bize getirir.

**Dezavantajı :** Index yapısı olmadığından belli elementleri atlama, elementleri bastan sona veya sondan başa sıralı getirme gibi işlemleri yapamaz

# For-Each Loop

**Soru 1-** Verilen bir array'de tekrar eden elementler icin, mukerrer olanlari silip, tum elemanlardan sadece 1 tane yapip eski array'e yeni halini atayip yazdirin.

**Soru 2-** Verilen int array'deki her elementin karelerini alip, karelerinin toplamini yazdiran bir method olusturun.

**Soru 3-** Verilen String bir array'deki her bir elementi kontrol edip,

- Kelimenin uzunlugu cift sayi ise ilk yarisini
  - Kelimenin uzunlugu tek sayi ise ortadaki harf dahil ikinci yarisini
- yeni bir listeye ekleyip yazdirin.

**Soru 4-** Kullanicidan bir cumle ve bir harf alin, harf cumlede kullanilmisa kac kere kullanildigini yazdirin, kullanilmadiysa “harf cumlede kullanilmamis” yazdirin.

**Soru 5-** Verilen iki array'in elementlerini karsilastirip, ikisinde ortak olan elementleri ayri bir liste olarak veren bir program yazin.



# Constructors

Bir class'dan bir obje urettigimizde obje temel olarak olusturuldugu class'daki ozelliklere sahip olur.

```
Car car1=new Car();
System.out.println(car1.fiyat);
System.out.println(car1.toString());
```

toString() String  
fiyat int  
ilanNo int  
marka String  
yil int  
model String

Java'da obje  
olusturuldugunda alacagi ilk  
degerlerin ne olacagini  
**(initialize) constructor'lar ile**  
duzenleriz.

```
public class Car {

    int ilanNo;
    String marka="deger atanmadı";
    String model="deger atanmadı";
    int yil;
    int fiyat;

    public void maxHiz(String yakit){

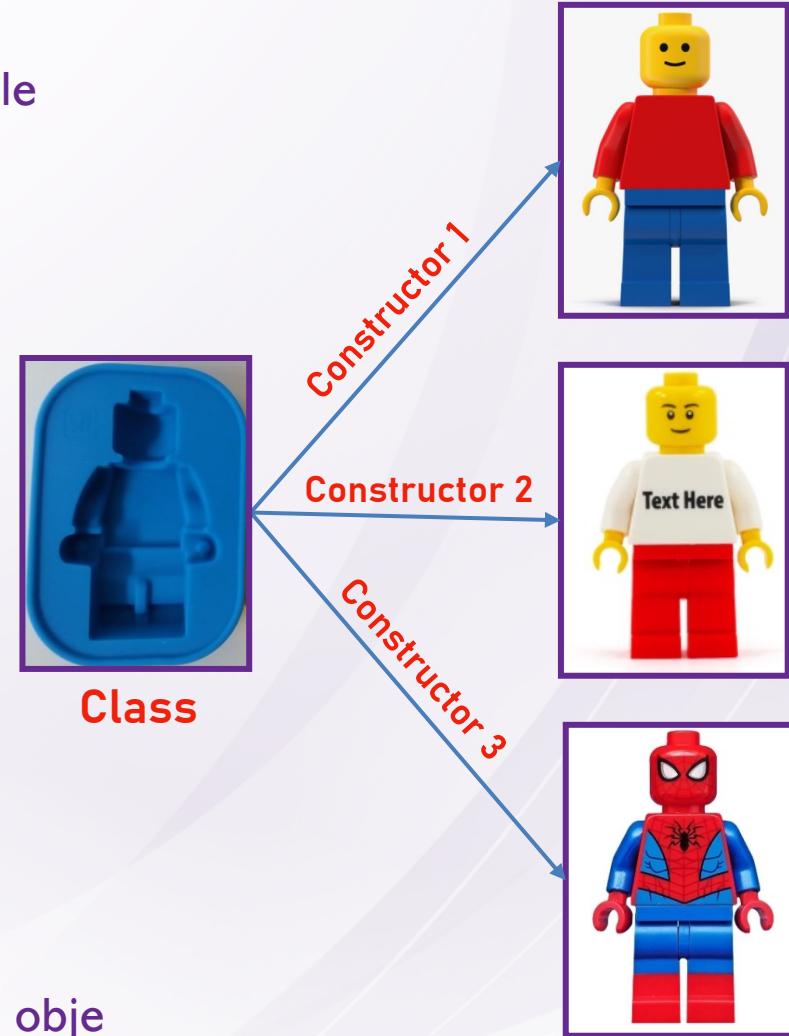
        if (yakit.equalsIgnoreCase( anotherString: "benzin")){
            System.out.println("Benzinli araclar icin max hiz 240 km/h");
        } else if (yakit.equalsIgnoreCase( anotherString: "dizel") ) {
            System.out.println("dizel araclar icin max hiz 260 km/h");
        } else if (yakit.equalsIgnoreCase( anotherString: "elektrikli") ) {
            System.out.println("elektrikli araclar icin max hiz 200 km/h");
        }else{
            System.out.println("Girilen yakit turu gecerli degil");
        }
    }
}
```

# Constructors

Java'da olusturulan bir obje'ye farkli **Constructor** ile ilk deger atamasini yaparak, ayni class'dan farkli ozelliklere sahip objeler olusturulabilir

Java'da obje olusturulan class'daki temel ozellikleri kullanmak istersek, parametresiz bir constructor kullanılır.

```
public class Araba {  
    public Araba() {  
    }  
}
```



İstersek ( ) icerisine yazacagimiz parametreler ile obje olusturulurken ozellikleri belirleyebiliriz.



# Constructors

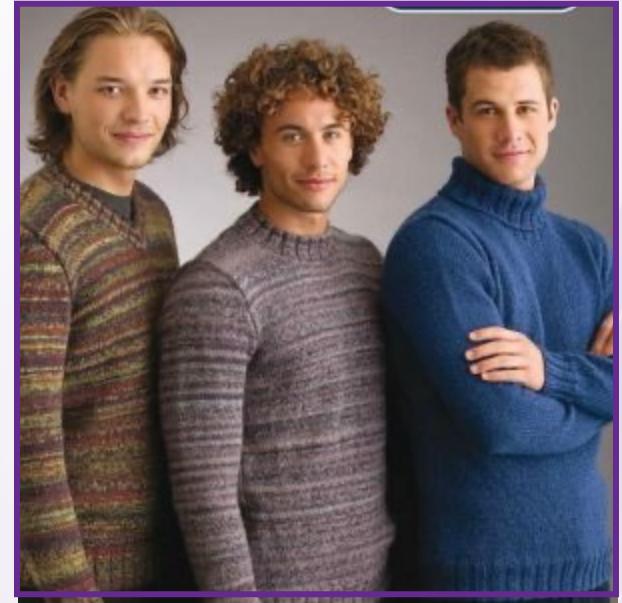
Yildiz triko Katalog

Bana triko uret YildizTriko ( );

Hic bir ozellik belirtilmemis, standart urun yollariz.

Bana mavi triko uret YildizTriko ("mavi" );

Sadece renk tercihi belirtilmis,



Bana mavi, bogazli yaka triko uret YildizTriko ("mavi","bogazli" );

Renk ve yaka tercihi belirtilmis,

Bana mavi, bogazli, medium, 100  
ad. yaka triko uret

YildizTriko ("mavi","bogazli", "medium", 100);

# Constructors

1- Constructor'lar syntax olarak method'lara benzerler ama return type'leri yoktur

2- Isimleri mutlaka Class ismi ile ayni olmalıdır. (Buyuk harfle baslar)

3- Static olarak tanimlanamazlar.

4- Method overloading'deki gibi farklı signature'lara sahip istendiği kadar constructor oluşturulabilir

5- Constructor'lar class'in içerisinde, method'ların ve diğer constructor'ların dışında oluşturulabilirler.

```
public class Araba {  
    public Araba() {  
    }  
  
    public Araba(String Model){  
    }  
  
    public static Araba (String Marka, int yil){  
    }  
}
```



# Default Constructor

Java'da bir obje olusturulup ona deger atanabilmesi icin mutlaka bir constructor calisir.

Java OOP concept geregi, her class obje uretmek icin olusturulmus bir kalip gorevi gorur.

Bir class olusturulduğunda biz hic bir constructor olusturmasak da, Java o class'dan obje olusturulmasına izin verir.

Biz constructor olusturmasak bile class'in obje olusturma gorevini yapmasi icin Java her Class'a default bir constructor koyar.

Default constructor parametresiz ve body'sinde hic bir kod olmayan bir constructor'dir ama gorunmez.

Biz bir class'da parametreli veya parametresiz bir constructor olusturunca Java default cons.'i siler.

```
public class Araba {  
}
```

```
public class Runner {  
  
    public static void main(String[] args) {  
  
        Araba arb= new Araba();  
    }  
}
```



# Constructor Nasıl Kullanılır

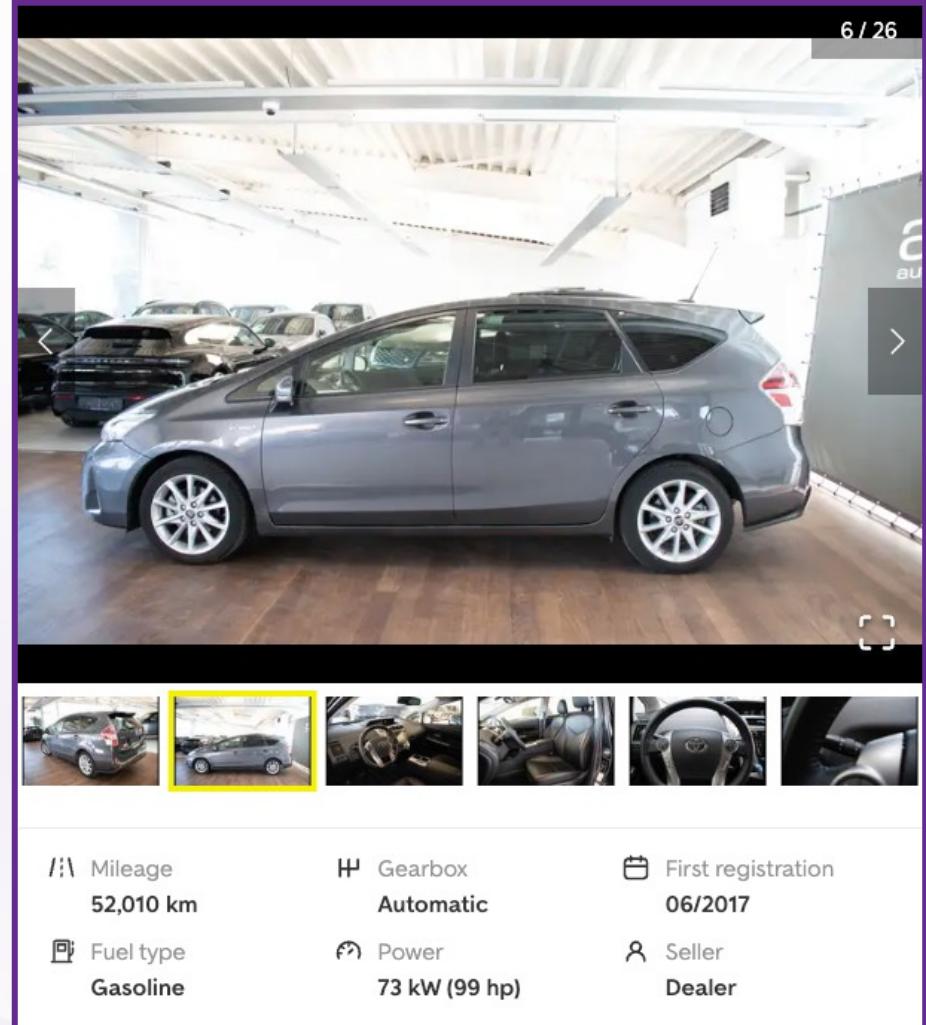
Ornek : Bir Car class'i olusturalim.

Arabalari olustururken standart ozellikler olarak asagidaki variable'lar olmalidir.

- İlan No
- Marka
- Model
- Yıl
- Fiyat

Ayrıca yakita gore hiz ve vites method'lari olmalidir.

CarRunner class'i olusturup icerisinde Car class'indan objeler olusturalim, deger atayip, yazdiralim.





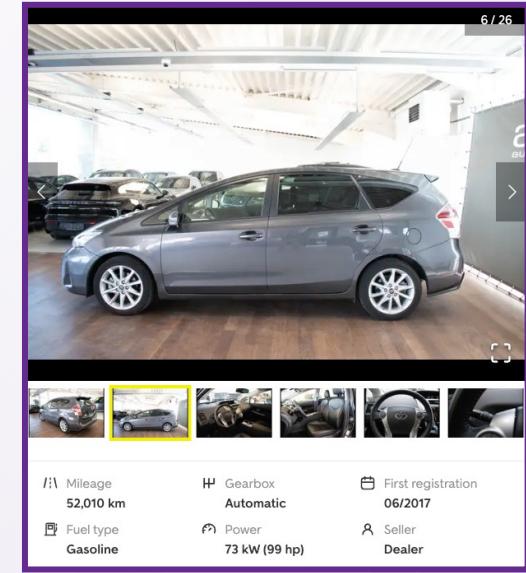
# Objeler Icin toString( ) Kullanimi

Bir class'dan olusturdugumuz objelerin ozelliklerini yazdirmak istedigimizde her bir obje icin instance variable'lari tek tek yazmamiz gerekir .

Bunun yerine IntelliJ'de menudeki Code / Generate seceneklerini kullanarak toString( ) method'u olusturabiliriz.

toString( ) method'u olusturulurken kullanilacak variable'lari secebılır ve/veya method içerisindeki dondurulen String'i istedigimiz gibi degistirebiliriz.

```
public String toString() {  
    return  
        "ilanNo : " + ilanNo +  
        ", marka : " + marka +  
        ", model : " + model +  
        ", yil : " + yil +  
        ", fiyat : " + fiyat ;  
}
```



```
Car car1=new Car();  
  
car1.yil=2017;  
car1.model="prius";  
car1.marka="Toyota";  
car1.ilanNo=1234;  
car1.fiyat=12000;  
  
System.out.println(car1.fiyat+", "+car1.ilanNo+", "+  
    car1.marka+", "+car1.model+", "+  
    car1.yil);  
// 12000,1234,Toyota, prius, 2017
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-27

Parametrized Constructors  
Constructor Call



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Onceki Gunden Aklimizda Kalanlar

- 1- Constructor : Bir class'dan obje olusturup ilk deger atamasi(initialization) icin o class'dan bir constructor calismalidir.
- 2- Constructor method'a benzer ama method degildir, constructor constructor'dir.
- 3- Biz her yeni olusturdugumuz class'da constructor olusturmak zorunda degiliz. Cunku Java olusturulan her yeni class'a obje olusturulabilmesi icin default constructor koyar.
- 4- Default constructor gorunmez, parametresi yoktur ve body'si bostur.
- 5- biz parametresiz veya parametreli bir constructor olusturdugumuzda Java default constructor'i siler. Yani gorunur bir constructor varsa default constructor YOKTUR.
- 6- Bir kod blogunun constructor olabilmesi icin 2 şart saglanmalidir
  - ismi class ismi ile bire-bir ayni olmalıdır(Case sensitive olarak)
  - return type olmamalidir
- 7- Bir class'da farkli constructor'lar olabilir. (Method overloading gibi signature'lari farkli)
- 8- Biz default constructor'i kullanarak bir obje olusturdugumuzda Java once constructor'in varligini kontrol eder sonra objeyi olusturur ve obje olusturulan class'daki tum instance variable'larin bir kopyasini olusturup objeye ilisiklendirir.
- 9- constructor calistiktan sonra obje uzerinde yapılan degisiklikler objenin olusturulduugu class'daki instance variable'lara degil, objeye ilisiklendirilen kopya instance variable'lara kaydedilir.

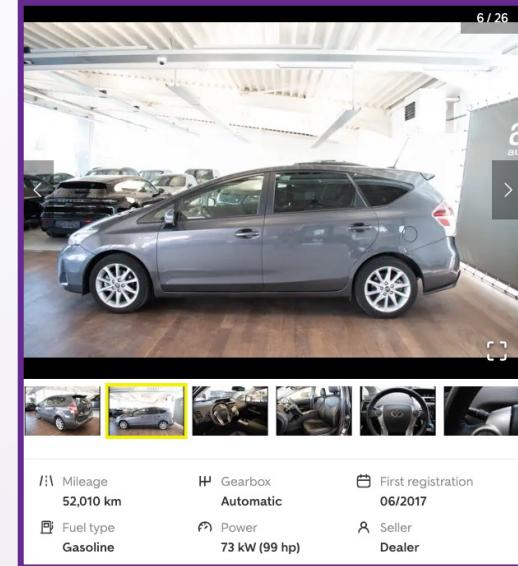
# Parametrized Constructors

Bir class'dan obje olustururken default constructor kullanilirsa, obje'nin ozellikleri, class'da instance olarak atanan degerler olacaktir.

Default constructor ile olusturulan objeye tum ozellikleri tek tek atamamiz gerekir.

```
Car car1=new Car();  
  
car1.yil=2017;  
car1.model="prius";  
car1.marka="Toyota";  
car1.ilanNo=1234;  
car1.fiyat=12000;
```

Tek tek atama yapmak istemezsek, parametreli constructor olusturmamiz ve constructor icerisinde atama yapmamiz gerekir.



```
public class Car {  
  
    int ilanNo;  
    String marka="deger atanmadı";  
    String model="deger atanmadı";  
    int yil;  
    int fiyat;
```



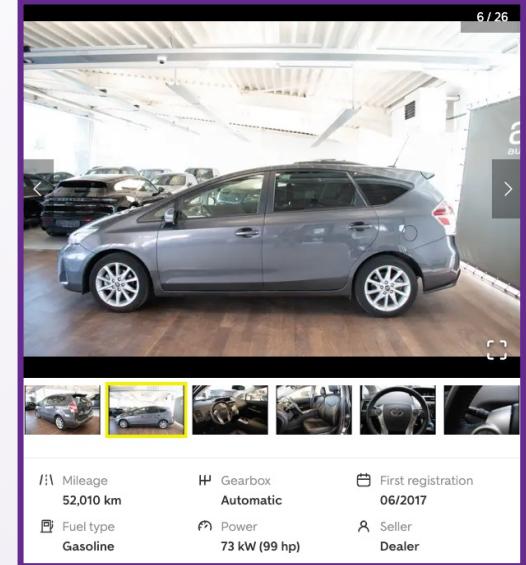
# Parametrized Constructors

Her bir obje icin instance variable'lara deger atamasi yapmamiz gereklidir.

Parametreli constructor'larin farki bu atamanin constructor icerisinde yapilmasidir.

```
public Car(int ilnNo, String mrk, String mdl, int yl, int fyt) {  
    ilanNo = ilnNo;  
    marka = mrk;  
    model = mdl;  
    yil = yl;  
    fiyat = fyt;  
}
```

Parametre isimleri ile instance variable isimleri ayni degilse atamalar direkt olarak yapilabilir.



```
Car car1=new Car();  
  
car1.yil=2017;  
car1.model="prius";  
car1.marka="Toyota";  
car1.ilanNo=1234;  
car1.fiyat=12000;
```

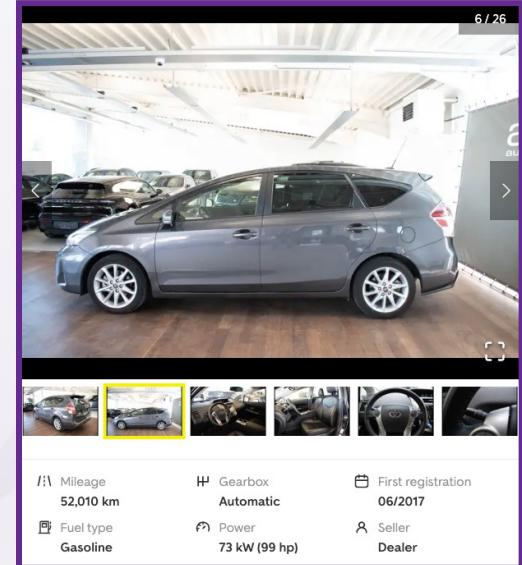
# Parametrized Constructors

Parametre isimleri ile instance variable isimleri ayni olursa atamalar direk yapmak istedigimizde scope problemi yasariz.

```
public Car(int ilanNo, String marka, String model, int yil, int fiyat) {  
    ilanNo = ilanNo;  
    marka = marka;  
    model = model;  
    yil = yil;  
    fiyat = fiyat;  
}
```

Bu problemi asmak icin birbiriyle ayni olan isimlerin hangisinin instance variable oldugunu **this.** keyword'u ile belirtmemiz gereklidir.

```
public Car(int ilanNo, String marka, String model, int yil, int fiyat) {  
    this.ilanNo = ilanNo;  
    this.marka = marka;  
    this.model = model;  
    this.yil = yil;  
    this.fiyat = fiyat;  
}
```





# Parametrized Constructors

```
Car car11= new Car();
// Car class'inda instance variable'lara
// atanan degerlere sahip bir car olusturur
```

```
Car car11= new Car( ilanNo: 1256, marka: "BMW");
// ilan no ve marka istedigimiz degerler olur
// geriye kalan ozellikler Car class'inda
// instance variable'lara atanan degerlere esit olur
```

```
public class Car {
    int ilanNo;
    String marka="deger atanmadı";
    String model="deger atanmadı";
    int yil;
    int fiyat;
```

```
public Car(){}
```

```
public Car(int ilanNo, String marka){
    this.ilanNo = ilanNo;
    this.marka = marka;
}
```

```
public Car(int ilanNo, String marka, String model, int yil, int fiyat) {
    this.ilanNo = ilanNo;
    this.marka = marka;
    this.model = model;
    this.yil = yil;
    this.fiyat = fiyat;
}
```

```
Car car3=new Car( ilanNo: 1236, marka: "Volvo", model: "S90", yil: 2010, fiyat: 13000);
System.out.println(car3);
// ilanNo : 1236, marka : Volvo, model : S90, yil : 2010, fiyat : 13000
```



Soru : Yandaki kod calistirildiginda konsol'da ne gorulur ?

```
public class Deneme {  
    String isim="John Doe";  
    int yas=40;  
  
    public Deneme(String isim, int yas) {  
        this.isim = isim;  
        this.yas = 30;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj=new Deneme(isim: "Fatih", yas: 35);  
  
        System.out.println("isim : " + obj.isim +  
                           " yas : " + obj.yas);  
    }  
}
```



# Parametrized Constructors

Soru : Yandaki kod calistirildiginda  
konsol'da ne gorulur ?

```
public class Deneme {  
    String isim="John Doe";  
    int yas=40;  
  
    public Deneme(){  
        isim="Seher";  
        yas=38;  
    }  
  
    public Deneme(String isim, int yas) {  
        this.isim = isim;  
        this.yas = 30;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme();  
  
        Deneme obj2=new Deneme( isim: "Fatih", yas: 35);  
  
        System.out.println("isim : " + obj1.isim +  
                           " yas : " + obj2.yas);  
    }  
}
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-28

### Constructor Call



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Onceki Gunden Aklimizda Kalanlar

- 1- Constructor : Bir class'dan obje olusturup ilk deger atamasi(initialization) icin o class'dan bir constructor calismalidir.
- 2- Default cons. kullanilarak olusturulan objeler, class'da verilen degerlere sahiptirler, bizim istedigimiz degerleri almasi icin her bir variable icin atama yapmaliyiz.
- 3- Eger obje olusturulurken bizim verdigimiz degerlere sahip olmasini istersek, parametreli cons.'lar kullaniriz.
- 4- Cons. cagirdigimiz kodda yazilan argumentler constructor'daki parametrelere deger olarak gider.
- 5- parametre olarak gonderilen degerlerin objeye atanmasi icin cons. icinde atama yapilmalidir.
  - parametre isimleri ile instance variable isimleri farkli ise instVar = paramIsmi; seklinde atama yapabiliyoruz. this.instVar = paramIsmi; yasmak zorunda degiliz
  - parametre isimleri ile instance variable isimleri ayni ise this.instVar = paramIsmi; yasmak zorundayiz. Aksi takdirde o scope'da ayni isimde variable oldugundan insVar. Kullanilmaz
- 6- Method overloading'de oldugu gibi bir class'in icerisinde signature'l farkli olmak uzere istedigimiz kadar constructor olusturabiliriz.
- 7- Birden fazla cons. oldugunda argument – parametre uyumuna gore cons. calisir.



# Constructor Call

Obje olusturulup ilk deger atamasinin yapilmasi icin MUTLAKA bir constructor calisir.

Bir class'daki constructor calisirken, ayni class'dan veya farkli bir class'dan bir constructor'in daha calismasi gerektiginde constructor call yapilir.

Constructor call inheritance'i anlamamiz icin onemli bir yapı tasidir.

Constructor call icin **this(parametreler)** kullanilir.

**KURALLAR :**

- 1- **this( )** constructor'in ilk satirina yazilmalidir.
- 2- ilk kuraldan dolayi sadece bir constructor call yapilabilir.

```
public class Deneme {  
    String isim="John Doe";  
    int yas=40;  
  
    public Deneme(){  
        isim="Seher";  
    }  
  
    public Deneme(String isim, int yas) {  
  
        this();  
        this.yas=yas;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme();  
  
        Deneme obj2=new Deneme( isim: "Fatih", yas: 35);  
  
        System.out.println("isim : " + obj1.isim +  
                           " yas : " + obj2.yas);  
    }  
}
```



Soru : Yandaki kod calistirildiginda konsol'da ne gorulur ?

isim : Seher, yas : 35

# Constructor Call

```
public class Deneme {  
    String isim="John Doe";  
    int yas=40;  
  
    public Deneme(){  
        isim="Seher";  
    }  
  
    public Deneme(String isim, int yas) {  
  
        this();  
        this.yas=yas;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme();  
  
        Deneme obj2=new Deneme( isim: "Fatih", yas: 35);  
  
        System.out.println("isim : " + obj1.isim +  
                           " yas : " + obj2.yas);  
    }  
}
```



# Constructor Call

Soru : Yandaki kod calistirildiginda  
konsol'da ne gorulur ?

isim : Seher  
isim : Esra  
isim : Seher  
isim : Murat

```
public class Deneme {  
    String isim="John Doe";  
    int yas=40;  
  
    public Deneme(){  
        isim="Seher";  
        System.out.println("isim : " + isim);  
    }  
  
    public Deneme(String isim){  
        this();  
        this.yas=30;  
        System.out.println("isim : " + isim);  
    }  
  
    public Deneme(String isim, int yas) {  
  
        this(isim: "Murat");  
        this.yas=45;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme(isim: "Esra");  
  
        Deneme obj2=new Deneme(isim: "Fatih", yas: 35);  
    }  
}
```



# Constructor Call

Soru : Yandaki kod icin hangileri dogrudur ? (Uygun olan tum cevaplari isaretleyin)

Which are true of the following code? (Choose all that apply)

```
1: public class Rope {  
2:     public static void swing() {  
3:         System.out.print("swing ");  
4:     }  
5:     public void climb() {  
6:         System.out.println("climb ");  
7:     }  
8:     public static void play() {  
9:         swing();  
10:    climb();  
11: }  
12: public static void main(String[] args) {  
13:     Rope rope = new Rope();  
14:     rope.play();  
15:     Rope rope2 = null;  
16:     rope2.play();  
17: }  
18: }
```

- A. The code compiles as is.
- B. There is exactly one compiler error in the code.
- C. There are exactly two compiler errors in the code.
- D. If the lines with compiler errors are removed, the output is climb climb.
- E. If the lines with compiler errors are removed, the output is swing swing.
- F. If the lines with compile errors are removed, the code throws a NullPointerException.

# Constructor Call

Yandaki kod icin sorulari true / false olarak cevaplayın

- 1- 4. satirdaki variable instance variable'dir. ....
- 2- 8. satirdaki variable instance variable'dir. ....
- 3- 17. satirdaki isim variable'i local variable'dir. ....
- 4- 12. satirdaki Deneme parametreli constructor'dir. ....
- 5- 21. satirdaki Deneme parametresiz constructor'dir. ....
- 6- 4.satırda atama yapmazsaç CTE olusur. ....
- 7- 8.satırda atama yapmazsaç 9.satırda CTE olusur. ....
- 8- 18.satırdan sonra 12.satır calisir. ....
- 9- 21.satırdan void yazıldıgi icin hata yapılmıştır. ....
- 10- Kod bu haliyle calismaz. ....

```

3 ► public class Deneme {
4     String isim="John Doe";
5     int yas=40;
6
7     □ public Deneme(){
8         String isim="Olçay";
9         System.out.println("isim : " + isim);
10    }
11
12    □ public Deneme(String isim){
13        this.yas=30;
14        System.out.println("isim : " + isim);
15    }
16
17    □ public Deneme(String isim, int yas) {
18        this( isim: "Murat");
19        this.yas=45;
20    }
21    □ public void Deneme(){
22        System.out.println(isim);
23    }
24
25 ► public static void main(String[] args) {
26
27        Deneme obj1=new Deneme( isim: "Esra");
28    }
29

```

# Static Keyword

Bir Hastane uygulamasi icin Hemsire class'i olusturalim.

Hemsire class'inda tum hemsireler icin ortak olan su variable'lari olusturalim.

- Hastane adi
- Hastane adresi
- Bashekim ismi

Ayrica her hemsire icin su variable'lari olusturalim.

- Hemsire ismi
- Hemsire adresi
- Hemsire Telefon



Hemsire



# Static Keyword

Bir Hastane uygulamasi icin  
Hemsire class'i olusturalim.

Hemsire class'inda tum hemsireler icin  
ortak olan su variable'lari olusturalim.

- Hastane adi
- Hastane adresi
- Bashekim ismi

Ayrica her hemsire icin su  
variable'lari olusturalim.

- Hemsire ismi
- Hemsire adresi
- Hemsire Telefon

Hemsire  
Class'i



Hastane adi  
Hastane adresi      static  
Bashekim ismi



Hemsire1

P1.Ismi : Yildiz  
P1.adres: Cankaya  
P1.Telefon : 4164352

Hemsire2

P2.Ismi : Ayse  
P2.adres: Sincan  
P2.telefon : 6151232

Hemsire3

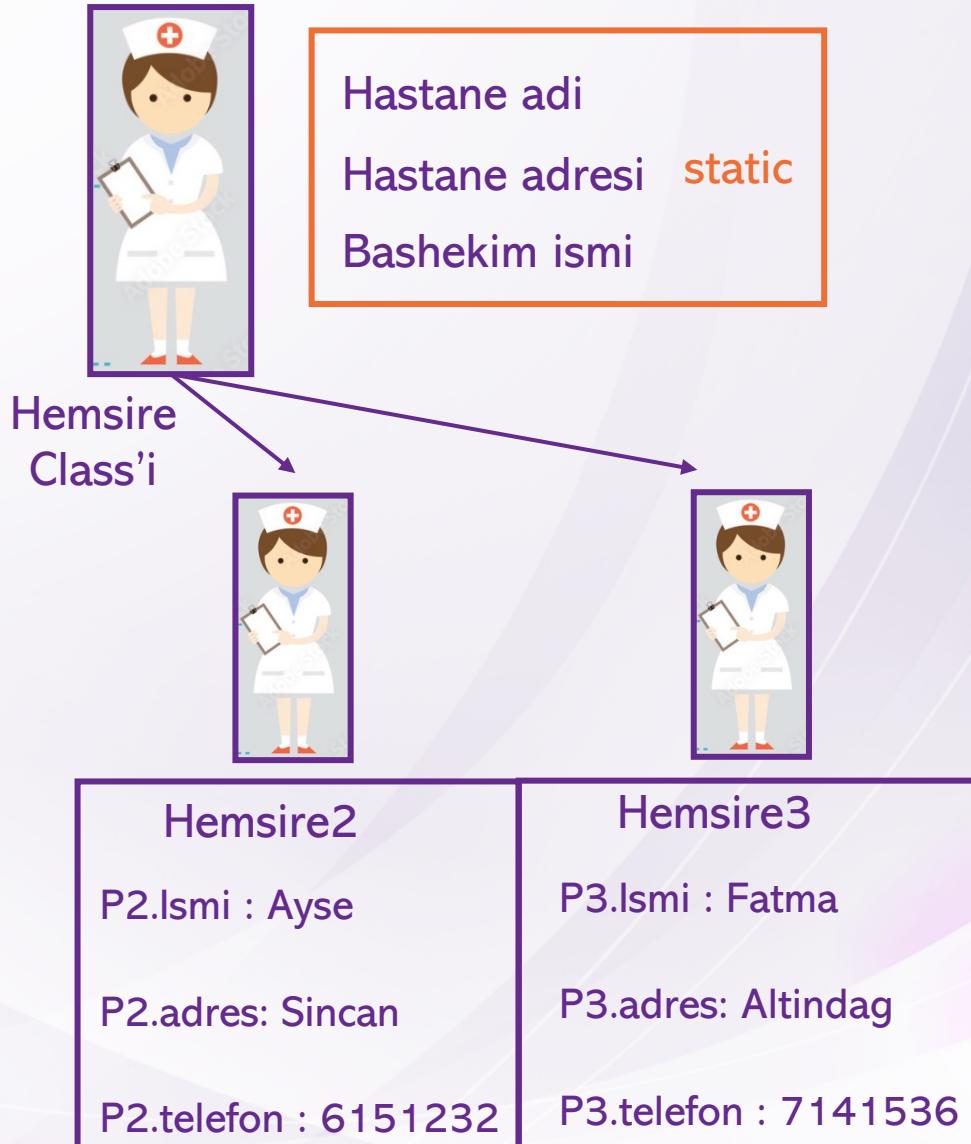
P3.Ismi : Fatma  
P3.adres: Altindag  
P3.telefon : 7141536

# Static Keyword

**static** keyword bir variable veya method'u class baglar.

**static** olarak etiketlenen variable veya method'a ulasmak icin classismi.variableismi yazmaniz yeterli olacaktır.

**static** variable'lar objeye degil class'a baglidir. Static variable'lari obje uzerinden de objelsmi.variableismi seklinde kullanmak istersek java CTE vermez ama static variable'a static yontemlerle ulasmalisiniz diye uyarır.



# Static Variables

- 1- Class calismaya basladiginda static variable'lar olusturulur.
- 2- Static variable sadece 1 tane olusturulur, kac tane obje olusturulursa olusturulsun ayni static variable'i kullanirlar. Herhangi bir satirda static variable degeri degistirilirse bu satirdan itibaren tum objeler icin yeni deger kullanilir.
- 3- Static variable'lar sadece 1 kere olusturulduugu icin memory kullanimi dusuktur.
- 4- Static variable'lar static olan veya olmayan tum method'lardan kullanilabilir.
- 5- Class disindan static variable'a ulasmak icin ClassIsmi.variableIsmi yazmamiz yeterlidir.



Hastane adi  
Hastane adresi static  
Bashekim ismi

Hemsire Class'i



Hemsire2

P2.Ismi : Ayse

P2.adres: Sincan

P2.telefon : 6151232

Hemsire3

P3.Ismi : Fatma

P3.adres: Altindag

P3.telefon : 7141536



Soru : Yandaki kod calistiginda  
konsolda ne yazdirir ?

# Static Variables

```
public class Deneme {  
    static int sayi=20;  
    int yas=40;  
  
    public Deneme(){  
  
    }  
  
    public void Deneme(){  
        sayi++;  
        yas++;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme();  
        Deneme obj2=new Deneme();  
        System.out.println(obj2.sayi+", "+ obj2.yas);  
    }  
}
```



# Static Methods

- 1- Bir method'u static yapmak için return type'dan önce **static keyword** yazılmalıdır.
- 2- Bir static method'dan static olmayan variable kullanamazsınız.
- 3- Bir static method static olan veya olmayan tüm method'lardan çağrılabılır.
- 4- Static olarak etiketlenen method'a başka class'dan ulaşmak için classısmi.methodısmı yazmanız yeterli olacaktır.

```
public class Deneme {  
    static int sayi=20;  
    int yas=40;  
  
    public static void main(String[] args) {  
        sayi++;  
        yas++;  
        method1();  
        method2();  
    }  
  
    public static void method1(){  
        sayi++;  
        yas++;  
    }  
  
    public void method2(){  
        sayi++;  
        yas++;  
        method1();  
        method2();  
    }  
}
```



# Static Methods

Soru : Yandaki kod calistiginda  
konsolda ne yazdirir ?

```
public class Deneme {  
    static int sayi=20;  
    int yas=40;  
  
    public Deneme(){  
        sayi++;  
        yas++;  
    }  
  
    public static int method1(){  
  
        return sayi;  
    }  
  
    public int method2(){  
  
        return yas;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme();  
        Deneme obj2=new Deneme();  
        Deneme obj3=new Deneme();  
        System.out.println(obj3.yas + "," +obj3.sayi);  
    }  
}
```



Soru : Yandaki kod  
calistiginda konsolda  
ne yazdirir ?

# Static Methods

```
public class Deneme {  
    static int sayi=20;  
    int yas=40;  
  
    public Deneme(){  
        sayi++;  
        yas=10;  
    }  
  
    public static int method1(){  
  
        return 2*sayi;  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme();  
        Deneme obj2=new Deneme();  
        int sonuc=obj2.method1();  
        System.out.println(obj2.yas +","+obj2.sayi+","+sonuc);  
    }  
}
```



# Static Methods

Soru : Yandaki kod calistiginda konsolda ne yazdirir ?

```
public class Deneme {  
    static int sayi=20;  
    int yas=40;  
    static String okul="ITU";  
  
    public Deneme(int a,int b, String c){  
        this.sayi=a;  
        this.yas=b;  
        this.okul=c;  
    }  
  
    public static void method1(Deneme dnm){  
  
        System.out.println("sayi: "+dnm.sayi+  
                           ", yas: "+dnm.yas+  
                           ", okul: "+dnm.okul);  
    }  
  
    public static void main(String[] args) {  
  
        Deneme obj1=new Deneme( a: 20, b: 30, c: "ODTU");  
        Deneme obj2=new Deneme( a: 15, b: 25, c: "Marmara");  
        method1(obj1);  
        method1(obj2);  
    }  
}
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-29

### Pass By Value

### Mutable - Immutable Classes



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# Static Methods

Soru : Yandaki kod calistiginda konsolda ne yazdirir ?

What is the result of the following program?

```
1: public class Squares {  
2:     public static long square(int x) {  
3:         long y = x * (long) x;  
4:         x = -1;  
5:         return y;  
6:     }  
7:     public static void main(String[] args) {  
8:         int value = 9;  
9:         long result = square(value);  
10:        System.out.println(value);  
11:    } }
```

- A. -1
- B. 9
- C. 81
- D. Compiler error on line 9.
- E. Compiler error on a different line.



# Instance Variables vs Static Variables

## Instance Variable

- 1) instance variables .....'in icinde ama .....'in disinda olusturulur
- 2) Instance variables bir ..... 'e baglidir. Dolayisiyla, bir ..... olusturuldugunda olusur ve ..... silindiginde silinirler.
- 3) Instance variables .....ismi ile cagrılabilirler.
- 4) instance variable icin ilk deger atamasi yapmak .....dir. Ilk atama yapilmazsa default deger alir.
- 5) Her yeni obje olusturuldugunda, instance variables ilk atanan degere esit olur.**True / False**
- 6) Bir class'i kullanarak 2 instance variable'a sahip 6 obje olusturursak, 12 instance variables olusturmus oluruz. **True / False**

## Static Variable

- 1) Static variables .....'in icinde ama .....'in disinda olusturulur
- 2) Static variables bir ..... 'a baglidir. Dolayisiyla, bir ..... olusturuldugunda olusur ve ..... silindiginde silinirler.
- 3) Static variables .....ismi ile cagrılabilirler.
- 4) Static variable icin ilk deger atamasi yapmak ..... dir. Eger ilk atama yapilmazsa default deger alir.
- 5) Static variable'a her yeni deger atamasi oldugunda, degeri tum objeler icin degisir.**True / False**
- 6) Bir class'i kullanarak 2 static variable'a sahip 6 obje olusturursak, 2 static variables olusturmus oluruz. **True / False**

# Static Blocks

- Static bloklar class ilk calistirildigi anda calisir ve class'in baslangic degerlerini olusturur(**initialaze**).
- Static bloklar tum class uyelerinden, main method'dan bile once calisir
- Eger birden fazla static blok varsa Java'nin genel cercevesine uygun olarak once ustteki static blok calisir.

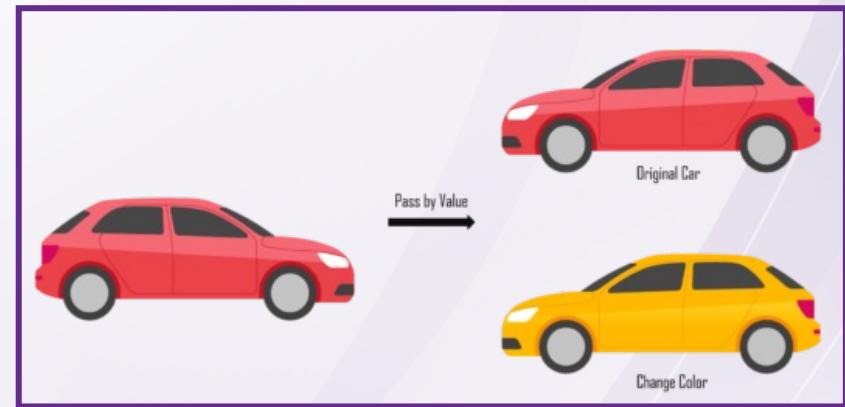
```
public class Deneme {  
  
    static int sayi=10;  
  
    Deneme(){  
        ++sayi;  
        System.out.println("Constructor calisti sayi : " + sayi);  
    }  
    static {  
        ++sayi;  
        System.out.println("ust static blok calisti, sayi : "+ sayi );  
    }  
  
    public static void main(String[] args) {  
        System.out.println("main method calisti, sayi : "+ sayi );  
        Deneme deneme=new Deneme();  
        System.out.println("main method obje olusturuldu, sayi : "+ sayi );  
    }  
  
    static {  
        ++sayi;  
        System.out.println("alt static blok calisti, sayi : "+ sayi );  
    }  
}
```



# Pass By Value / Pass By Reference

Pass By Value veya Pass By Reference yazılım dilini tasarlayanların verdiği / vereceği bir karardır.

Arabam sari olsa, nasıl dururdu ? Sorusunu cevaplamak için arabayı sariya boyamak yerine, bir kopya üzerinde değişiklik yapıp kontrol eder, eğer beğenirsek arabamizi sari boyatmaya karar veririz.



Java'da bir variable'i method'a gönderdiginde, direkt orjinal variable'i göndermeyi, orada yapılan değişikliğin kalıcı olmasının istedigimizde ise main method içerisinde method'dan gelen değeri, orjinal variable'a atamayı tercih etmистir.



# Pass By Value / Pass By Reference

**Uygulama :** Main method'da verilen fiyat degerine %10, %20 ve %30 indirim yaparak, indirimli fiyatı bize döndüren 3 method olusturun.

Main method'dan bu 3 method'u arka arkaya cagirip, return edilen fiyatları yazdirin.

```
public class PBV {  
    public static void main(String[] args) {  
  
        double fiyat=100;  
  
        System.out.println(indirimYap10(fiyat)); // 90.0  
        System.out.println(indirimYap20(fiyat)); // 80.0  
        System.out.println(indirimYap30(fiyat)); // 70.0  
  
        System.out.println(fiyat); // 100.0  
    }  
  
    public static double indirimYap10(double fiyat) {  
  
        return fiyat*0.9;  
    }  
    public static double indirimYap20(double fiyat) {  
  
        return fiyat*0.8;  
    }  
    public static double indirimYap30(double fiyat) {  
  
        return fiyat*0.7;  
    }  
}
```

# Pass By Value / Pass By Reference

**Uygulama :** Main method'da bir list olusturup elemanlar atayalim.

2 method olusturup once listeyi, sonra listedeki elemanlari degistirelim.

- ilk method'da bizim listemiz disinda bir liste olusturup deger atayalim, sonra yeni listeyi bizim asil listemize atayalim ve asil listemizi main method'a dondurelim.
- ikinci method'da bizim listemizin elementlerini degistirip, asil listemizi main method'a dondurelim.

```
public class PBV {  
    public static void main(String[] args) {  
  
        List<Integer> liste=new ArrayList<>();  
        liste.add(10);  
        liste.add(20); // [10, 20]  
  
        System.out.println(listeyiYeniDegerAta(liste)); // [30, 40]  
        System.out.println(liste); // [10, 20]  
  
        System.out.println(listeElementleriniDegistir(liste)); // [50, 60]  
        System.out.println(liste); // [50, 60]  
    }  
  
    public static List<Integer> listeyiYeniDegerAta(List<Integer> liste) {  
        List<Integer> yeniListe=new ArrayList<>();  
        yeniListe.add(30);  
        yeniListe.add(40);  
        liste=yeniListe;  
        return liste;  
    }  
  
    public static List<Integer> listeElementleriniDegistir(List<Integer> liste) {  
  
        liste.set(0,50);  
        liste.set(1,60);  
        return liste;  
    }  
}
```

Her iki method call'dan sonra listemizi main method'da yazdirip, degisimi kontrol edin.

# Pass By Value / Pass By Reference

Uygulama : Bir onceki uygulamayı bir array için de yapalım.

Sonuc : Java PassByValue kullanır.

List veya array gibi birden fazla element içeren objelerde,

- objeyi degistiremezsiniz
- elementleri degistirdigimizde objenin referansı degismediği için elementlerin değişikliği kalıcı olur.

```
public class PBV {  
    public static void main(String[] args) {  
  
        int[] arr={10, 20, 30};  
  
        System.out.println(Arrays.toString(arrayeYeniDegerAta(arr))); // 40, 50, 60  
        System.out.println(Arrays.toString(arr)); // [10, 20, 30]  
  
        System.out.println(Arrays.toString(arrayElementleriniDegistir(arr))); // [80, 90, 30]  
        System.out.println(Arrays.toString(arr)); // [80, 90, 30]  
    }  
    public static int[] arrayeYeniDegerAta(int[] arr) {  
        int[] yeniArr= {40,50,60};  
        arr=yeniArr;  
        return arr;  
    }  
    public static int[] arrayElementleriniDegistir(int[] arr) {  
  
        arr[0]=80;  
        arr[1]=90;  
        return arr;  
    }  
}
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-30

### Mutable - Immutable Classes



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# Mutable / Immutable Classes

Java OOP concept kullanır. Obje merkezli bu kullanımda objelerin mutable veya immutable olması onların özelliklerini, dolayısıyla da objelerin kullanım yerlerini değiştirecektir.



**Mutable Objects:** Mutable objeler'de, obje oluşturulurken verilen değerler sonradan değiştirilebilir.

**Immutable Objects:** Immutable objeler'de ise, obje oluşturulurken verilen değerler sonradan değiştirilemez.

Yani, immutable objeler değiştirilemez ve yeni değer atanamaz.

# Mutable / Immutable Classes

En çok kullanılan, immutable class'lar String ve Wrapper Class'lardır. Bu class'lardan oluşturulan objeler de immutable (**degistirilemez**) olacaklardır.

```
public static void main(String[] args) {  
  
    String isim= "Ali Can";  
  
    isim ="Veli Cem";  
  
}
```

Immutable objelere yeni bir değer atamak istediğimizde Java aynı isimde yeni bir obje oluşturur ve pointer'i yeni objeyi işaretleyecek şekilde değiştirir.

**Immutable Objects**'in amacı güvenli es zamanlı çalışma (**thread safe**) özelliğinden faydalananmaktır.



# Mutable vs Immutable Classes

Immutable class olan String'de method kullanarak yaptigimiz degisiklikler kalici olmazken, mutable class olan ArrayList'de method kullanarak yaptigimiz degisiklikler kalici olur.

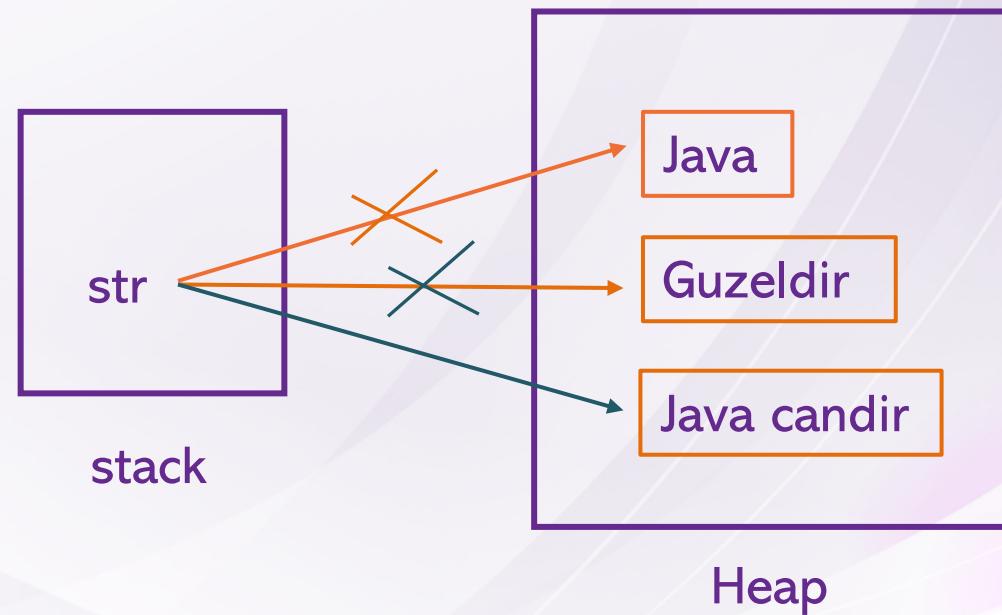
```
public static void main(String[] args) {  
  
    String isim= "Ali Can";  
  
    isim.toUpperCase();  
    isim.substring(3,5);  
  
    System.out.println(isim); // Ali Can  
  
    List<Integer> sayilar= new ArrayList<>();  
  
    sayilar.add(10);  
    sayilar.add(20);  
    sayilar.add(30);  
  
    System.out.println(sayilar);  
}
```

# Immutable Classes

String'de degisikligin kalici olmasi icin atama yapabiliriz.

Immutable class'larda atama olsa bile Java immutable objeyi degistirmez. Java ayni isimde yeni bir obje olusturur ve pointer'i yeni objeyi isaretleyecek sekilde degistirir.

```
public static void main(String[] args) {  
  
    String str= "Java";  
  
    str="Guzeldir";  
  
    str = "Java candir.";  
}
```

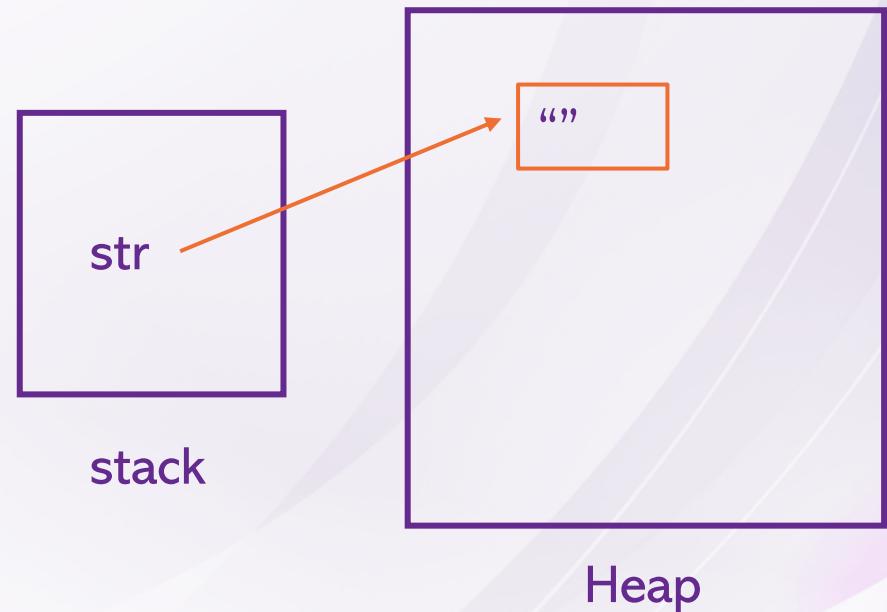




# Immutable Classes

Asagidaki kod calisirsa Java kac obje olusturur ?

```
String str="";  
  
for (int i = 1; i <=100 ; i++) {  
    str+=i;  
}
```



# String Pool

Java'da iki turlu String olusturulabilir.

## 1- Bugune kadar yaptigimiz sekilde bir String olusturmak.

Bu sekilde olusturulan veya direk kullanilan String'lerde Java String havuzunu kontrol eder, bire-bir ayni String'i bulursa yeni obje olusturmaz, bir objeyi birden fazla referans point edebilir.

```
String str1= "Java";  
  
String str2= new String( original: "Guzeldir");
```

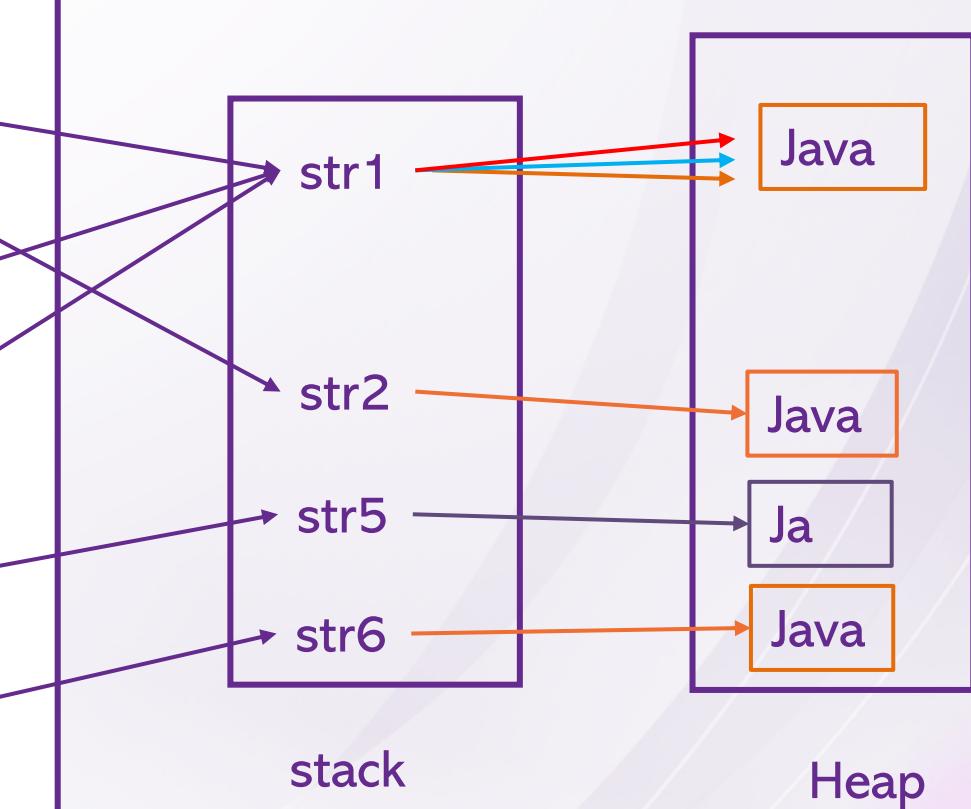
## 2- new keyword ile bir String olusturmak.

Bu sekilde olusturulan String'de, once esitligin sag tarafina bakilir new keyword'u oldugu icin yeni obje ve yeni referans olusturulur..



# Immutable Classes

```
String str1= "Java";  
  
String str2= new String( original: "Java");  
  
String str3="Java";  
  
String str4= "Ja" + "va";  
  
String str5= "Ja";  
String str6=str5.concat( str: "va");  
  
System.out.println(str1==str2); // false  
System.out.println(str1==str3); // true  
System.out.println(str1==str4); // true  
System.out.println(str1==str6); // false
```





# Mutable / Immutable Classes

Asagidaki kod blogu icin sonuc nedir? (Dogru olan tum siklari isaretleyin)

What is the result of the following code? (Choose all that apply)

```
13: String a = "";
14: a += 2;
15: a += 'c';
16: a += false;
17: if ( a == "2cfalse") System.out.println("==");
18: if ( a.equals("2cfalse")) System.out.println("equals");
```

- A.** Compile error on line 14.
- B.** Compile error on line 15.
- C.** Compile error on line 16.
- D.** Compile error on another line.
- E.** ==
- F.** equals
- G.** An exception is thrown.



# Mutable / Immutable Classes

Asagidaki kod blogu icin sonuc nedir?

What is the result of the following statements?

```
6: List<String> list = new ArrayList<String>();  
7: list.add("one");  
8: list.add("two");  
9: list.add(7);  
10: for(String s : list) System.out.print(s);
```

- A.** onetwo
- B.** onetwo7
- C.** onetwo followed by an exception
- D.** Compiler error on line 9.
- E.** Compiler error on line 10.



# Mutable / Immutable Classes

Asagidaki kod blogu icin sonuc nedir?

What is the result of the following statements?

```
3: ArrayList<Integer> values = new ArrayList<>();  
4: values.add(4);  
5: values.add(5);  
6: values.set(1, 6);  
7: values.remove(0);  
8: for (Integer v : values) System.out.print(v);
```

- A.** 4
- B.** 5
- C.** 6
- D.** 46
- E.** 45
- F.** An exception is thrown.
- G.** The code does not compile.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-31

### Date – Time

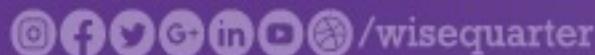
### Varargs



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





# Onceki Dersten Aklimizda Kalanlar

- 1- Pass By Value – Pass By Reference : Bu programlama dilini olusturanların vereceği bir karardır ve Java Pass By Value'yu tercih etmistir.
- 2- Pass By Value : Bir method'dan baska bir method'a variable veya obje gonderildiginde, variable veya objenin kendisinin degil degerinin gonderilmesidir.
- 3- Bu sayede diger method'da yapılan degisiklik asil variable veya objenin degerini degistirmez.
- 4- Dikkat edilmesi gereken bir nokta : Birden fazla element iceren array veya list gibi objelerin elementleri gidilen method'da degistirilirse, element degisikligi asil objeye de islenir. Fakat eger abjemizi gonderdigimiz method'da objeye new keyword kullanilarak yeni bir obje degeri atanirsa, yei objede yapılan degisiklikler orjinal objemizi ETKILEMEZ.
- 5- Immutable Classes : Immutable class'lar ve o class'lardan olusturulan objeler DEGISTIRILEMEZ. (bildigimiz en meshur immutable class'lar String ve Wrapper)
- 6- Mutable class'lar ise degistirilebilir. List ve array gibi..
- 7- Bu iki farkli class yapısında dikkat etmemiz gereken en onemli nokta : immutable class'larda method kullandigimizda orjinal variable'in degeri degismezken, mutable class'larda method kullandigimizda orjinal obje degisir.
- 8- String'de atama yaptigimizda String degistirilemeyecegi icin Java kopya bir obje olusturup, yeni degeri ona atar ve variable'in pointer'ini yeni olusturdugu objeye yonlendirir. Eski objeler Garbage Collector tarafından toplanmaya bekler.

# Date & Time

Java'da tarih ve saat için 3 Class'dan yararlanabiliriz.

```
LocalDate tarih = LocalDate.now();
System.out.println(tarih);
// 2022-08-30
```

```
LocalTime saat = LocalTime.now();
System.out.println(saat);
// 17:01:21.670687
```

```
LocalDateTime tarihZaman=LocalDateTime.now();
System.out.println(tarihZaman);
// 2022-08-30T17:01:21.670732
```

Tarih ve saat yazım şekli için Java bircok standart'a uygun sonuc uretebilir, biz de istedigimiz formata donusturebiliriz.



# LocalDate

```
LocalDate trh=LocalDate.now();
```

```
System.out.println(trh); // 2022-08-31
```

```
System.out.println(trh.getDayOfWeek()); // WEDNESDAY
```

```
System.out.println(trh.getDayOfMonth()); // 31
```

```
System.out.println(trh.getDayOfYear()); // 243
```

```
System.out.println(trh.getMonth()); // AUGUST
```

```
System.out.println(trh.getMonthValue()); // 8
```

```
System.out.println(trh.plusYears( yearsToAdd: 3)
                    .plusMonths( monthsToAdd: 2)
                    .plusDays( daysToAdd: 10)); // 2025-11-10
```

```
System.out.println(trh.minusWeeks( weeksToSubtract: 3)
                    .minusDays( daysToSubtract: 3)); // 2022-08-07
```

LocalDate.now( )

method'u o anki tarihi  
döndürür

get....( )

ile tarihteki istedigimiz  
bilgiyi kullanabiliriz.

→ plus...( ) ve minus...( )

ile ileri veya geri tarihe  
gidebiliriz.



# LocalDate

```
LocalDate dogTar1=LocalDate.of( year: 2011, month: 10, dayOfMonth: 12);  
LocalDate dogTar2=LocalDate.of( year: 2011, month: 10, dayOfMonth: 14);  
  
System.out.println(dogTar1.isLeapYear()); // false  
  
System.out.println(trh.lengthOfYear()); // 365  
  
System.out.println(trh.withDayOfMonth(20)); // 2022-08-20
```

LocalDate.of( ) method'u  
istedigimiz degerlerle tarih  
olusturabiliriz

isLeapYear( )

method'u tarih'in leap  
year olup olmadigini  
döndürür

lengthOfYear( ) yılın uzunlugunu döndürür

with...( ) tarihteki gun, ay, yil  
gibi bir bolumu  
istedigimiz deger ile  
bize döndürür

# LocalTime

LocalDate de LocalTime ile benzer method'lara sahiptir.

LocalDate'i istedigimiz degerlerle olusturabilir, ileri veya geri gidebilir, satin istedigimiz bir bolumunu alabilir veya istedigimiz farkli bir deger ile degistirebiliriz..

```
/*
bir for loop olusturalim
1'den 10000'e kadar olan sayilari toplatalim
ve bu islemi ne kadar surede yaptigini bulalim
*/
LocalTime forLoopBasi=LocalTime.now();
System.out.println(forLoopBasi);

int toplam=0;
for (int i = 1; i <=100000 ; i++) {
    toplam+=i;
}
LocalTime forLoopSonu=LocalTime.now();
System.out.println(forLoopSonu);

int nanoSaniye= forLoopSonu.getNano()- forLoopBasi.getNano();
System.out.println(nanoSaniye);
```

Iki satirin calismasi arasindaki zaman farkini bulmak icin islemin basladigi ve bittiği satirlarda LocalTime objesi olusturup bunların farkını alabilirim.

# LocalDate

Istedigimiz ulke ve/veya sehirdeki zamani kullanmak LocalDate.now( ) method'unda **Zonel.of( istenenSehirZonel)** parametresi kullanabiliriz.

```
public static void main(String[] args) {  
  
    LocalTime saat = LocalTime.now();  
  
    System.out.println(saat); // 19:03:27.658174  
  
    LocalTime localSaat1=LocalTime.now(ZoneId.of( zonel: "Europe/London"));  
    System.out.println(localSaat1); // 18:06:44.835579  
  
    LocalTime localSaat2=LocalTime.now(ZoneId.of( zonel: "Turkey"));  
    System.out.println(localSaat2); // 20:07:19.380256  
  
    LocalTime localSaat3=LocalTime.now(ZoneId.of( zonel: "America/New_York"));  
    System.out.println(localSaat3); // 13:09:40.408075  
}
```

# Period

Period Class'i ile iki tarih arasindaki zaman dilimini elde edebiliriz.

Iki zaman arasindaki period olarak bulduktan sonra get( ) methodlari ile aradaki farki yil, gun gibi istedigimiz zaman birimleri acisindan da elde edebiliriz.

```
public static void main(String[] args) {  
  
    LocalDate bugun=LocalDate.now();  
    LocalDate dogumTarihi=LocalDate.of( year: 1972, month: 01, dayOfMonth: 01);  
  
    Period yas=Period.between(dogumTarihi,bugun);  
  
    System.out.println(yas); // P50Y8M  
  
    System.out.println(yas.getYears()); // 50  
  
}
```

# DateTimeFormatter

Date Time Formatter ile tarih ve saatı istediğimiz formatta yazdırabiliriz.

```
LocalDateTime ldt=LocalDateTime.now();
System.out.println(ldt); // 2022-09-01T19:23:27.851814
```

```
DateTimeFormatter dtf=DateTimeFormatter.BASIC_ISO_DATE;
```

```
System.out.println(dtf.format(ldt)); // 20220901
```

```
DateTimeFormatter dtf2=DateTimeFormatter.ISO_WEEK_DATE;
```

```
System.out.println(dtf2.format(ldt)); // 2022-W35-4
```

```
DateTimeFormatter dtf3=DateTimeFormatter.ofPattern("dd/MM/YYYY");
```

```
System.out.println(dtf3.format(ldt)); // 01/09/2022
```

```
DateTimeFormatter dtf4=DateTimeFormatter.ofPattern("d/M/YY");
```

```
System.out.println(dtf4.format(ldt)); // 1/9/22
```

```
DateTimeFormatter dtf5=DateTimeFormatter.ofPattern("d/MMM/YY");
```

```
System.out.println(dtf5.format(ldt)); // 1/Sep/22
```

Format'i dünya capında belirlenen tarih ve saatı formatları ile yazdırabileceğimiz gibi

Kendimizin belirleyeceğim bir formatta da yazdırabiliriz.

# DateTimeFormatter

Date Time Formatter ile tarih ve saatı istediğimiz formatta yazdırabiliriz.

## GÜN

d : basta 0 varsa onu yazmadan gün numarası

dd: tek haneli günleri 01 gibi basına sıfır yazarak gün numarası

DDD : yılın kaçinci günü olduğunu yazar

E, EE, EEE : gün isminin ilk 3 harfi

EEEE : gün isminin tamamını

AY (Ay için M, dakika için m kullanılır)

M : basta 0 varsa onu yazmadan ay numarası

MM: tek haneli ayları 01 gibi basına sıfır yazarak ay numarası

MMM : Ay isminin ilk 3 harfi

MMMM : Ay isminin tamamı

YY : yılın son iki rakamını

YYYY : Yılın tamamını

Saat : (24 saat üzerinden istiyorsak H, 12 saat duzeninde istiyorsak h)

HH : saatin tamamı, tek rakamlı saat olursa 02 gibi

H : tek rakamlı saat olursa sadece saatı

a yazarsak AM veya PM değerini yazar

Format'i String olarak belirlerken  
kullanacağımız harfler ile  
tarih ve/veya saat  
istedigimiz bolumunu,  
istedigimiz formatta  
yazdırabiliriz.



# Varargs

Java'da method call yapildiginda, Java compile time da iki kontrol yapar

- 1- Method Ismi
- 2- Method call'da kullanılan argument'ler ile method'larda bulunan parametrelerin uyumu

Bu iki kontrol neticesinde istenen isim, parametre sayisi ve parametrelerin data turlerinde farklılık varsa, Java CTE verir.

```
topla( ...a: 3,4); // int[] a=[3,4] = 7
topla( ...a: 3,4,5); // int[] a=[3,4,5] = 12
topla( ...a: 3,4,5,6,7); // int[] a=[3,4,5,6,7] = 25
topla( ...a: 3,5,7,9,7,5,4,1,2); // = 43
}

public static void topla(int... a) {
    int toplam=0;
    for (int each: a
        ) {
        toplam+=each;
    }

    System.out.println("Verilen sayilarin toplami : " + toplam);
}
```

Varargs, methodlardaki aynı turden olan parametrelerin sayisi konusunda bize esneklik kazandırır.

Varargs, list gibi esnek sayıda parametre alır, ancak Array altyapısını kullanır. Dolayısıyla varargs'daki datalarla işlem yapmak için Array method'ları kullanılır.

Varargs, parametre sayisi 0 olduğunda da CTE vermez, boş bir array oluşturur.



# Varargs

```
/*
verilen sayilardan ilki haric digerlerini toplayip
bulunan toplam degerini ilk sayı ile carpip
yazdiran bir method olusturun
input 3,4,5,6
output 3 * (4+5+6) = 45
*/
islemYap( a: 3, ...b: 4,5,6,7,8,9,8,7,6,6,6,6,6,2,4);
}

public static void islemYap(int a, int... b) {
    int toplam=0;

    for (int each:b
        ) {
        toplam+=each;
    }
    System.out.println("istenen islemin sonucu : " + a*toplam);
}
```

Varargs'i belirtmek için data turunden sonra ... kullanılır.

Bir vararg'in eleman sayısı sınırlanılamaz, bundan dolayı varargs parametre en sona yazılmalıdır.

Aynı sebeple bir method'da birden fazla varargs parametre olarak kullanılamaz.

Bir method'da varargs'dan sonra parametre yazılırsa Java CTE verir.



# Varargs

Soru : Asagidaki Class calistirilirsa konsolda ne yazdirir ?

```
public class C09 {  
  
    public static void main(String[] args) {  
        new C09().C09( a: 1, ...b: "Java");  
  
        C09 obj=new C09();  
        obj.C09( a: 2, ...b: "Java", "Guzeldir");  
    }  
  
    public void C09(int a, String... b){  
  
        System.out.print(b[b.length-a]+ " ");  
    }  
}
```

# Varargs

Soru : Asagidaki method'lardan hangileri calisir?

Which of the following compile? (Choose all that apply)

- A.** public void moreA(int... nums) {}
- B.** public void moreB(String values, int... nums) {}
- C.** public void moreC(int... nums, String values) {}
- D.** public void moreD(String... values, int... nums) {}
- E.** public void moreE(String[] values, ...int nums) {}
- F.** public void moreF(String... values, int[] nums) {}
- G.** public void moreG(String[] values, int[] nums) {}



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-32

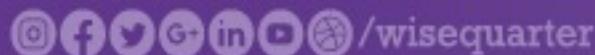
String Builder  
Access Modifier



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# String Builder

Java'da metin datalar ile kullanabilecek 3 Class vardır.

Bunlar'dan hangisinin kullanilacagini, ihtiyaclarla gore, development asamasinda karar verilir.

Class	Mutability	Multi Thread	hiz
1- String	Immutable		
2- StringBuilder	Mutable	Multi Thread calismayi desteklemez	Daha hizlidir
3- StringBuffer	Mutable	Multi Thread calismayi destekler	Daha Yavastir

# String Builder

String Immutable oldugundan, her degisiklikte yeni objeler olusturacaktir. Ozellikle kullanici sayisi ve metin veriler fazla oldugunda hafiza kullanimi acisindan String yerine String Builder kullanilabilir.

String Builder olustururken 3 farkli sekilde olusturulabilir.

```
StringBuilder sb1=new StringBuilder();
// kapasitesi 16 olan bos bir SB olusturur
System.out.println(sb1.capacity()); // 16

StringBuilder sb2= new StringBuilder("Ali Can");
// yazilan String'e uygun kapasitede SB olusturur
// ve icine Ali Can yazar
System.out.println(sb2.capacity()); // 23

StringBuilder sb3= new StringBuilder( capacity: 7);
// kapasitesi 7 olan bir SB olusturur
System.out.println(sb3.capacity()); // 7
```

String Builder'i olustururma yontemimiz kullanilacak hafizayı etkileyecegi icin buyuk uygulamalarda buna da dikkat edilmelidir.



# String Builder

Kullanılacak data için kapasite belirterek StringBuilder oluşturabiliriz.

```
StringBuilder sb = new StringBuilder( capacity: 7);
```

0	1	2	3	4	5	6
---	---	---	---	---	---	---

```
sb.append("Ali");
```

A	I	i				
0	1	2	3	4	5	6

```
sb.append(" Kemal");
```

A	I	i		K	e	m	a	I								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

**NOT :** String Builder'in kapasitesini asan bir ekleme yapıldığında, kapasitesini otomatik olarak **2 \* eski kapasite + 2** yapar.



# String Builder

```
StringBuilder sb1= new StringBuilder( capacity: 7);

System.out.println(sb1.capacity()); // 7
System.out.println(sb1.length()); // 0

sb1.append("Ali Can");
System.out.println(sb1.capacity()); // 7
System.out.println(sb1.length()); // 7

sb1.append("Bilmeyen var mı ? ");
System.out.println(sb1.capacity()); // 25
System.out.println(sb1.length()); // 25

sb1.append("Inanmayan beri gelsin.");
System.out.println(sb1.capacity()); // 52
System.out.println(sb1.length()); // 47

sb1.trimToSize();
System.out.println(sb1.capacity()); // 47
System.out.println(sb1.length()); // 47
```

String Builder'da **capacity( )** hafızadaki ayrılan bolumu gösterirken, **length( )** reel olarak uzunlugu verir.

Capacity ile length'i eşitlemek istersek, yani **kapasiteyi kullanılan uzunluga indirmek** istersek **sb.trimToSize( )** method'unu kullanabiliriz.



# String Builder

```
StringBuilder sb1=new StringBuilder("Veli Cem");

sb1.delete(4,7); // ilk index dahil, 2.index haric olarak aradakile

System.out.println(sb1); // Veli

sb1.deleteCharAt( index: 4);

System.out.println(sb1); // Veli

sb1.insert( offset: 4, str: " Han");

System.out.println(sb1); // Veli Han

sb1.replace( start: 5, end: 8, str: "Tan");

System.out.println(sb1); // Veli Tan

sb1.reverse();

System.out.println(sb1); // naT ileV
```

String Builder'da da String'deki method'lara benzer bircok method vardır.

Method isimleri ve islevleri farkliliklar gosterebilir. Ornegin String'deki replace method'u ile StringBuilder'daki replace method'u farkli kullanima sahiptir.

StringBuilder'daki reverse method'u string'de bircok islem yaparak gerceklestirecegimiz terse cevirme islemini direk yapar.



# String Builder

`==`, `equals`, `compareTo`

```
StringBuilder sb1= new StringBuilder("Ali Can");

StringBuilder sb2= new StringBuilder("Ali Can");

String str="Ali Can";

System.out.println(sb1==sb2); // false

System.out.println(sb1.equals(sb2)); // false

System.out.println(sb1.equals(sb1)); // true

// System.out.println(sb1==str); farkli data turleri
// oldugu icin Java CTE verir

System.out.println(sb1.equals(str)); // false

System.out.println(sb1.compareTo(sb2)); // 0
```

String builder'da iki String Builder'i karsilastirmak icin `==` kullanilirsa false doner.

`equals` ile iki String Builder'i karsilastirdigimizda da false doner. Istisnasi String builder'i kendisi ile karsilastirdiginizda equals method'u true donecektir.

`equals` ile bir String Builder ile bir String'i karsilastirdigimizda Java CTE vermez ama farkli data turleri oldugundan her zaman false dondurur.

Iki String Builder'in ayni oldugunu anlamayan en iyi yolu `compareTo()` dur. Esit ise 0 doner, esit degil ise ilk farkli harflerin arasindaki farki verir.



# String Builder

String builder'da olmayan String method'larini kullanmak istersek sb.toString( ) ile String Builder'i String'e cevirebiliriz.

```
StringBuilder sb1= new StringBuilder("Java Guzeldir");

sb1.substring( start: 5);
// String class'indan calisir dolayisiyla immutable olur

System.out.println(sb1.substring( start: 5)); // Guzeldir

// sb1= sb1.substring(5); farkli data turu oldugundan atamayı kabul etmez

System.out.println(sb1); // Java Guzeldir

// SB'da contains yok
// sb1 Java iceriyor mu ?

System.out.println(sb1.toString().contains("Java")); // true
```

Bu cevirm islemini yaptigimizda data turunu degistirdigimizi ve String immutable oldugu icin method'larin kalici degisiklik yapmayacagini unutmamamiz gereklidir.



# String Builder

Soru : Asagidaki kod calistirildiginda sonuc ne olur ?

What is the result of the following code?

- ```
7: StringBuilder sb = new StringBuilder();  
8: sb.append("aaa").insert(1, "bb").insert(4, "ccc");  
9: System.out.println(sb);
```
- A.** abbaaccc
  - B.** abbaccca
  - C.** bbaaaccc
  - D.** bbaaccca
  - E.** An exception is thrown.
  - F.** The code does not compile.



# String Builder

Soru : Asagidaki kod calistirildiginda sonuc ne olur ?

What is the result of the following code?

```
2: String s1 = "java";
3: StringBuilder s2 = new StringBuilder("java");
4: if (s1 == s2)
5:     System.out.print("1");
6: if (s1.equals(s2))
7:     System.out.print("2");
```

- A.** 1
- B.** 2
- C.** 12
- D.** No output is printed.
- E.** An exception is thrown.
- F.** The code does not compile.



# String Builder

Soru : Asagidaki kod calistirildiginda sonuc ne olur ?

Which are the results of the following code? (Choose all that apply)

```
String numbers = "012345678";
System.out.println(numbers.substring(1, 3));
System.out.println(numbers.substring(7, 7));
System.out.println(numbers.substring(7));
```

- A. 12
- B. 123
- C. 7
- D. 78
- E. A blank line.
- F. An exception is thrown.
- G. The code does not compile.



# String Builder

Soru : Asagidaki kod calistirildiginda sonuc ne olur ?

What is the result of the following code?

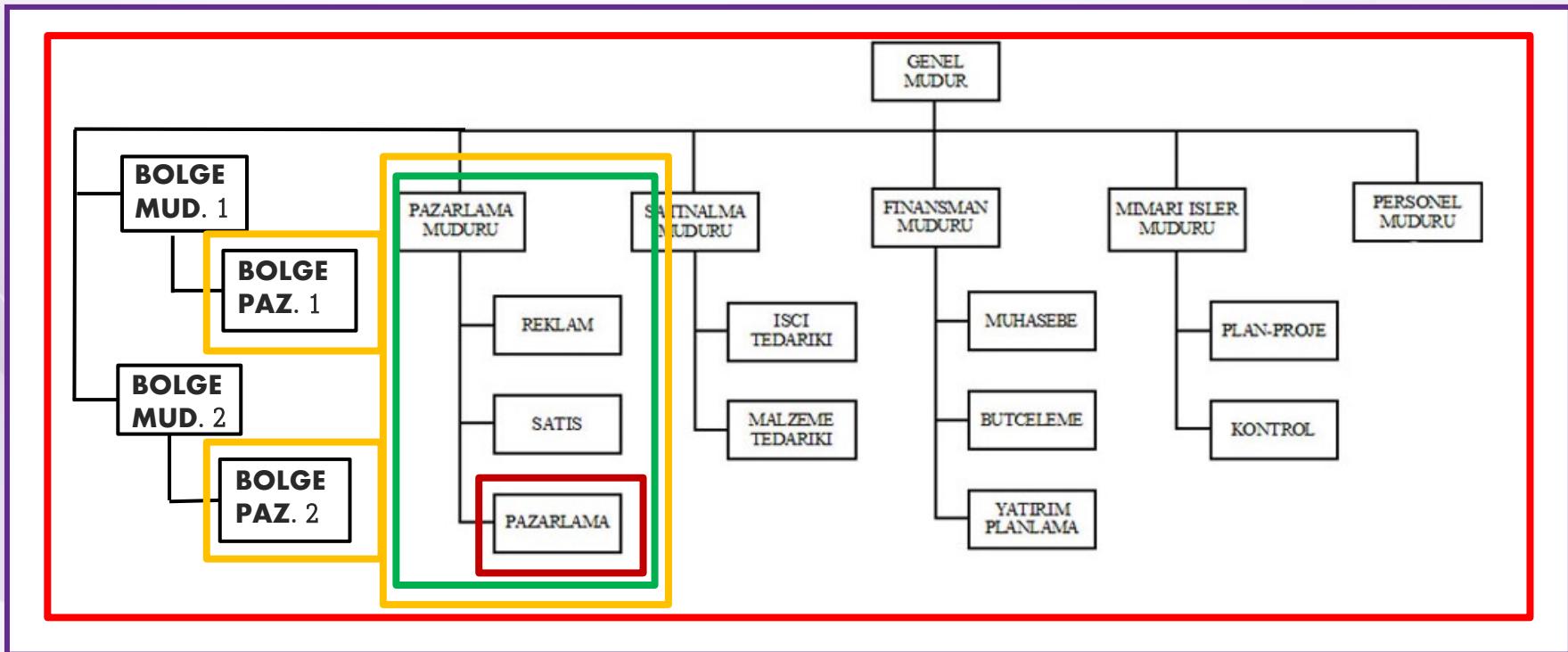
```
4: int total = 0;  
5: StringBuilder letters = new StringBuilder("abcdefg");  
6: total += letters.substring(1, 2).length();  
7: total += letters.substring(6, 6).length();  
8: total += letters.substring(6, 5).length();  
9: System.out.println(total);
```

- A. 1
- B. 2
- C. 3
- D. 7
- E. An exception is thrown.
- F. The code does not compile.

# Access Modifier

Buyuk bir projede calistiginizda class'lara ve datalara erisim yetkisi saglamak onemli olacaktir.

Java OOP konsepte gercek hayattaki ihtiyaclarimizi kod dunyasina uyardamistir.



# Access Modifier

Access Modifiers (erisim belirleyici) bir class uyesinin scope'unu yani nerelerden erisilebilecegini belirler.

Bir class uyesini olustururken, bu class uyesine nerelerden erisilebilecegini ve kullanabilecegini belirleyen access modifier **kullanmak zorundayiz**.

Simdiye kadar buna dikkat etmemistik, cunku Java access modifier yazilmasa da **default access modifier** tanimlar ve kodun calismasini saglar.

Java'da 4 access modifier vardir.

- 1- Private
- 2- Default
- 3- Protected
- 4- Public

**NOT-** Class'lar public veya default olmak zorundadir, private veya protected olamazlar.



# Access Modifier

1- Private : Private olarak belirlenen class uyelerini sadece icinde oldugu class'dan kullanabilirsiniz.

The screenshot shows a Java project structure and a code editor. The project tree on the left has a red 'X' drawn over it, indicating that private members are not accessible from outside the package. The code editor on the right shows a class definition with private static members and public static methods. A green checkmark is placed over the public methods to indicate they are accessible.

```
1 package package1;
2
3 public class Class1 {
4
5     private static String privateVariable;
6     private static void privateMethod(){
7
8
9     public static void main(String[] args) {
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27 }
```



# Access Modifier

2- Default : Default olarak belirlenen class uyelerini sadece icinde oldugunuz package'dan kullanabilirsiniz.

The screenshot shows a Java project structure on the left and a code editor on the right. In the project structure, several packages are listed under 'src': package1, package2, package3, and package4. Under package1, there are Class1, Class2, ClassChild1, and ClassChild2. Under package2, there are Class1, Class2, ClassChild1, and ClassChild2. Under package3, there are Class1, Class2, ClassChild1, and ClassChild2. A large red 'X' is drawn over the entire package1 folder, indicating that members defined there are not accessible from other packages. The code editor displays the following Java code:

```
1 package package1;
2
3 public class Class1 {
4
5     static String defaultVariable;
6     static void defaultMethod(){
7
8
9     public static void main(String[] args) {
10
11
12
13
14     public static void method1(){
15
16
17
18
19     public void method2(){
20
21
22
23 }
```

A green checkmark is placed over the 'defaultVariable' and 'defaultMethod' declarations, and another large green checkmark is placed over the 'method1' declaration, both indicating that these members are accessible within their package.

**NOT :** Default olarak belirlemek istedigimiz class uyelerini declare ederken **default** yazilmaz.

Biz access modifier'i yazmazsak, Java otomatik olarak default access modifier olarak tanimlayacaktir.



# Access Modifier

3- Protected : Protected olarak belirlenen class üyelerini içinde olduğunuz package'dan ve baska class'lardaki child class'lardan kullanabilirsiniz.

The screenshot shows a Java project structure and a code editor. The project tree on the left has packages package1, package2, package3, and package4. package1 contains Class1, Class2, ClassChild1, and ClassChild2. package2 and package3 both contain Class1, Class2, ClassChild1, and ClassChild2. package4 is empty. The code editor on the right shows a class definition:

```
1 package package1;
2
3 public class Class1 {
4
5     protected static String protectedVariable;
6     protected static void protectedMethod(){
7
8
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23 }
```

Annotations highlight visibility rules:

- Class1 members:** protectedVariable and protectedMethod are highlighted with a teal border. A green checkmark is placed over the package1 folder in the project tree, indicating they are visible within the same package.
- ClassChild1 and ClassChild2 members:** These classes are shown in package2 and package3 respectively. Their members are crossed out with red X's, indicating they are not visible from outside their respective packages.
- Class1 members in package4:** The members of Class1 in package4 are also crossed out with red X's, indicating they are not visible from outside their package.
- Class1 members in package1:** The members of Class1 in package1 are highlighted with an orange border and a green checkmark, indicating they are visible within the same package.
- Class1 members in package2 and package3:** The members of Class1 in package2 and package3 are highlighted with an orange border and a green checkmark, indicating they are visible within their respective packages.
- Class1 members in package4:** The members of Class1 in package4 are highlighted with an orange border and a green checkmark, indicating they are visible within their package.



# Access Modifier

4- Public : Public olarak belirlenen class uyelerini içinde olduğunuz projedeki tüm class'lardan kullanabilirsiniz. Hic bir sınırlama (**restriction**) icermez.

The screenshot shows a Java project structure and a code editor. The project tree on the left contains a package named 'package1' which is highlighted with an orange box and has a green checkmark pointing to it. The code editor on the right shows the following Java code:

```
1 package package1;
2
3 public class Class1 {
4
5     public static String publicVariable;
6     public static void publicMethod(){
7
8
9     public static void main(String[] args) {
10
11
12
13
14     public static void method1(){
15
16
17
18
19     public void method2(){
20
21
22 }
```

Two sections of the code are highlighted with orange boxes and have green checkmarks pointing to them: the declaration of 'publicVariable' and the declaration of 'method1'. This illustrates that both variables and methods are accessible from any class within the same package.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-33

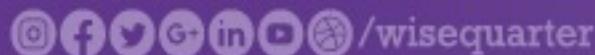
### Encapsulation Inheritance



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





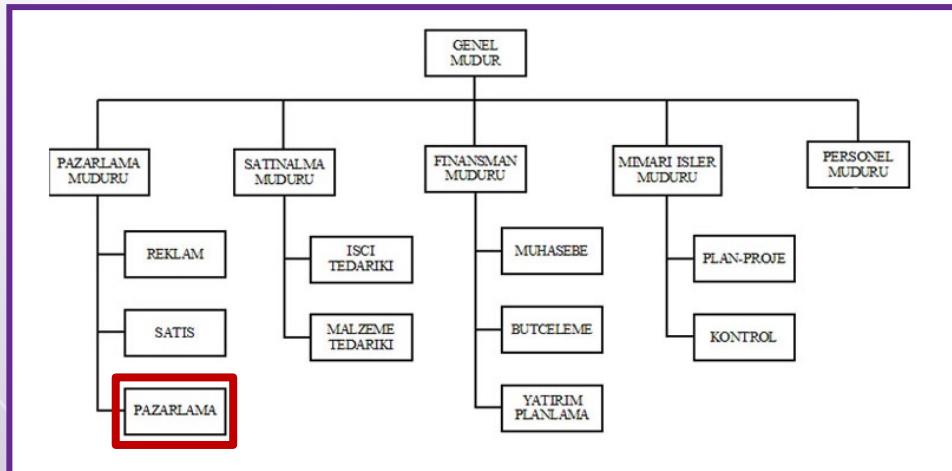
# Onceki Dersten Aklimizda Kalanlar

- 1- Access Modifier : Bir class uyesine (class level variables, methods, constructors..) hangi class'lardan erisebilecegimizi belirler.
- 2- **Onemli NOT** : Bir class uyesine erisim yetkisi / yontemi icin sadece access modifier yeterli olmaz, class uyesinin **static** olup olmadigina da bakmak gerekir.
- 3- 4 access modifier vardir. Dardan genise dogru yazarsak
  - **private** : sadece ve sadece icinde bulundugu class'dan erisilebilir
  - **default access modifier** : icinde bulunulan class ve icinde bulunulan package'daki diger class'lardan erisilebilir.  
default access modifier yazi ile yazilmaz. Bir class uyesinin onunde gorunur bir access modifier yoksa, access modifier'i default'tur deriz.
  - **protected** : icinde bulunulan class, icinde bulunulan package'daki diger class'lar, baska package'lar altindaki child class'lar
  - **public** : tum class'lardan erisilebilir.

# Encapsulation

Pazarlama birimine bir rapor görevi verildi.

Görev ve yetkilendirme :



- 1- satis bolumundeki personel rapor'un olusmasi icin gerekli bilgileri girebilmeli ancak sonucları gorememeli.
- 2- Rapor olusturulduktan sonra, izin verilen kullanilar raporu gorebilmeli ancak veriler üzerinde degisiklik yapamamali.

Access modifiers bir class uyesine erisim yetkisini duzenler ancak, read / write yetkilerini yani gorebilme ve degistirebilme yetkilerinin ayirimini yapamaz.

Bu iki ozelligi ayirmaniz gerekiyorsa encapsulation KULLANMALISINIZ.

# Encapsulation

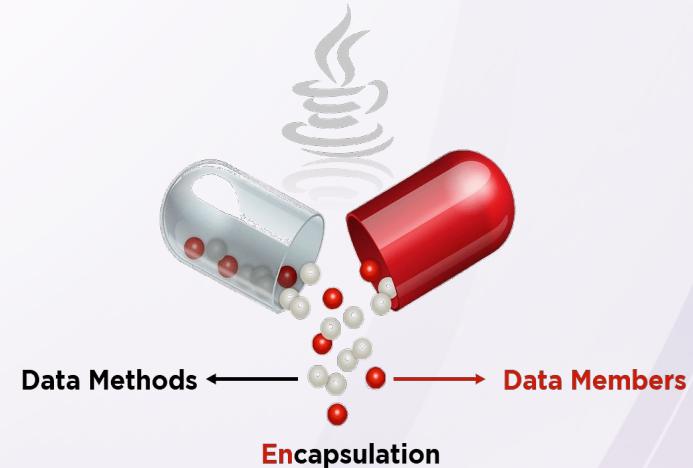
Encapsulation, Java'nin OOP concept'inin temel bileşenlerindendir.

Encapsulation, bir obje için önemli olan variable ve method'ları bir class'a koyup, bu variable ve method'lara erişim yetkisini okuma(getter) ve yazma(setter) olarak özelleştirebilme amacıyla gider.

Java'da bir class uyesine erişim yetkisi access modifier ile belirlenir, ancak access modifier'larla get ve set yetkisinin ayıratılmamız mümkün değildir.

Encapsulation'da variable'lara atama yapma veya okuma getter ve setter method'ları ile yapıldığından, bu method'lar içerisinde datayı inceleyebilir, uygun olmayan değerlerin atanmasını engelleyebiliriz.

Encapsulation ile bir variable'i hem static yaparak başka class'lardan rahatça ulaşılabilirliğini sağlayıp, hem de kimsenin yeni değer atamasına izin vermeyerek variable'in değerini koruyabiliyoruz.





# Encapsulation Nasil Uygulanir?

Encapsulation kontrollu erisim saglamak icin kullanilir.

Bunun icin once tum yetkileri iptal eder, sonra istedigimiz oranda erisim yetkisi duzenleriz.

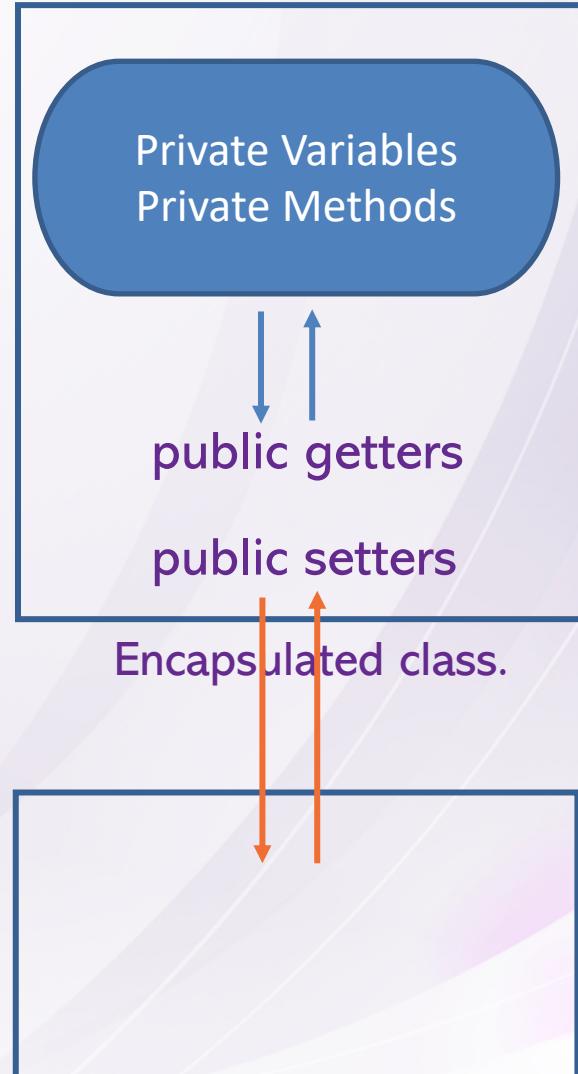
**1.Adim : Encapsule edilecek class uyelerini private yapin**

Bu durumda diger class'lardan private class uyelerine erisim, mumkun olmayacaktir.

**2.Adim : Encapsule etmek istedigimiz class uyeleri icin yetki vermek istediginiz oranda getter ve/veya setter method'lari olusturun.**

getter ve/veya setter method'lari ayni class'da olduklari icin private variable ve method'lara rahatlikla erisebilirler.

Diger class'lar da **public** olan getter ve/veya setter method'larina rahatlikla erisebilirler.



# Getters & Setter (Java Beans)

getter method'lari bilgiyi okumak (**read**) icin kullanilir.

Eger bilgi geri dondurulurken yapilmasi gereken baska bir islem varsa, encapsule edilen class'daki private bir method ile bu islem yapilabilir.

setter method'lari bilgiye atama yapmak yani yasmak(**write**) icin kullanilir

Atama yapmadan once bir kontrol yapmak gerekirse encapsule edilen class'daki private bir method ile bu islem yapilabilir.

Bir variable icin sadece setter method'u varsa o variable'a data girisi yapilabilir ama kimse datalari goremez.

```
public class CC {  
  
    private static String isim;  
  
    public static String getIsim() {  
        isimYazdir();  
        return isim;  
    }  
  
    public static void setISIM(String isim) {  
  
        databaseEkle();  
        isimKontrolEt(isim);  
        CC.isim=isim;  
    }  
}
```

Encapsulated class.



# Getter & Setter Naming Convention

Genel olarak, **getter method'ları** `getVariableIsMi`, **setter method'ları** `setVariableIsMi` şeklinde adlandırılır.

**boolean variable'lar için getter method'ları** `isVariableIsMi` şeklinde yazılır.

```
public class CC {  
  
    private static String isim;  
    private boolean atHoliday;  
  
    public static String getIsim() {  
  
        return isim;  
    }  
  
    public static void setIsim(String isim) {  
        CC.isim=isim;  
    }  
  
    public boolean isAtHoliday() {  
        return atHoliday;  
    }  
  
    public void setAtHoliday(boolean atHoliday) {  
        this.atHoliday = atHoliday;  
    }  
}
```

# Encapsulation Genel Tekrar

## 1) Encapsulation nedir?

Encapsulation, datalara erisim yetkisini read ve write olarak ayirmak icin kullanilan bir yontemdir.

## 2) Encapsulation icin datalara direk erisim nasil engellenir?

Data'lar icin private access modifier kullanarak diger class'lardan erisimin onune geceriz.

## 3) Private datalara diger class'lardan ulasabiliriz?

Getter ve setter method'larini kullanarak ulasabiliriz.

## 4) getter( ) method'u ne yapar ?

Encapsule edilen datalari okumamizi saglar.

## 5) setter ( ) method'u ne yapar ?

Encapsule edilen datalara deger atayabilmemizi saglar

## 6) immutable class nedir?

Encapsule edilen bir class'da sadece getter method'u olusturursak datalari okuyabiliriz ama degistiremeyiz. Bu tur class'lara immutable class denir.

## 7) setter( ) method'lari icin naming convention nedir?

Tum data turleri icin isimler "set" ile baslar.

## 8) getter( ) method'lari icin naming convention nedir?

Boolean data turu icin "is" ile, diger data turleri icin "get" ile baslar.



# Encapsulation

Asagidaki isimlendirmelerden hangileri uygundur ?

Which are methods using JavaBeans naming conventions for accessors and mutators?  
(Choose all that apply)

- A. public boolean getCanSwim() { return canSwim;}
- B. public boolean canSwim() { return numberWings;}
- C. public int getNumWings() { return numberWings;}
- D. public int numWings() { return numberWings;}
- E. public void setCanSwim(boolean b) { canSwim = b;}



# Encapsulation

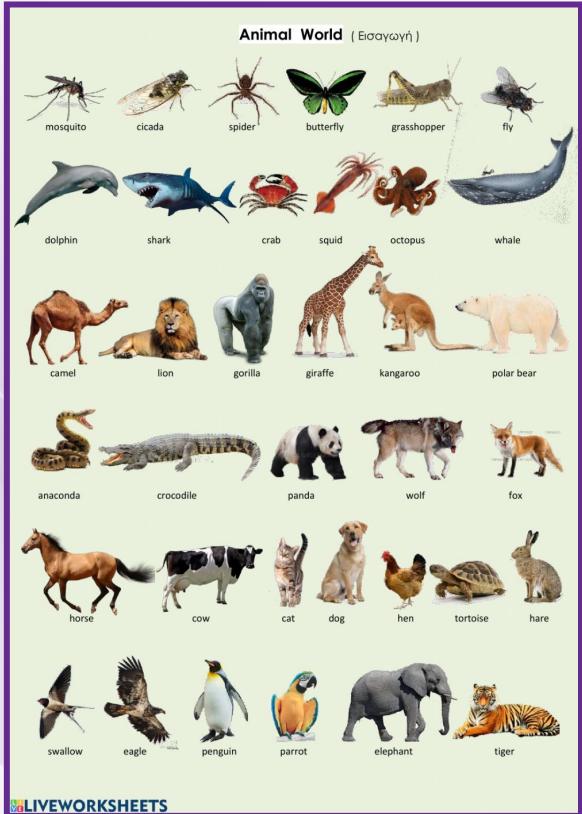
Asagidakilerden hangileri dogrudur ?

Which of the following are true? (Choose all that apply)

- A. Encapsulation uses package private instance variables.
- B. Encapsulation uses private instance variables.
- C. Encapsulation allows setters.
- D. Immutability uses package private instance variables.
- E. Immutability uses private instance variables.
- F. Immutability allows setters.



# Inheritance (Kalitim-Miras)



Hayvanlar Alemi

hareket( ) : hareket ederler  
solunum( ) : nefes alırlar  
beslenme( ) : beslenirler  
cogalma( ) : cogalırlar  
omur( ) : yasar ve olurler



Kuslar

kanat( ) : kanatlidirlar  
solunum( ) : akcigerle nefes alırlar  
gaga( ) : gagalari vardir  
cogalma( ) : yumurtayla cogalırlar

hareket( ) : yuruler  
beslenme( ) : et ve ot



Kumes Kuslari



Avcı Kuslar

hareket( ) : ucarlar  
beslenme( ) : et yerler  
pence( ) : pencelidir  
gaga( ) : sivri gagali



# Inheritance (Kalitim-Miras)





# Inheritance (Kalitim-Miras)



Yildiz Hastanesi



Insan Kaynaklari



Muhasebe

Isim :  
Soyisim :  
maas( ) : maas alirlar  
izin( ) : izinleri vardir  
emekli( ) : emekli olabilir

SgkNo :  
RaporluMu :  
mesai( ) : saat ucreti\*2  
stdMaas( ) : 30\*8\*12



Doktor



Hemsire



Teknik personel

nobet( ) : haftada 1 gun  
mesai( ) : saat ucreti\*3  
stdMaas( ) : 30 \* 8 \* 25

Bas Hemsire

nobet( ) : haftada 1 gun  
stdMaas( ) : 30 \* 8 \* 20

Servis Hemsiresi

nobet( ) : haftada 2 gun  
stdMaas( ) : 30 \* 8 \* 18

Poliklinik Hemsiresi

nobet( ) : yok  
stdMaas( ) : 30 \* 8 \* 12

mesai( ) : saat ucreti\*2.5  
nobet( ) : ayda 1 gun  
stdMaas( ) : 30 \* 8 \* 12



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-34

### Inheritance



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

@ /wisequarter

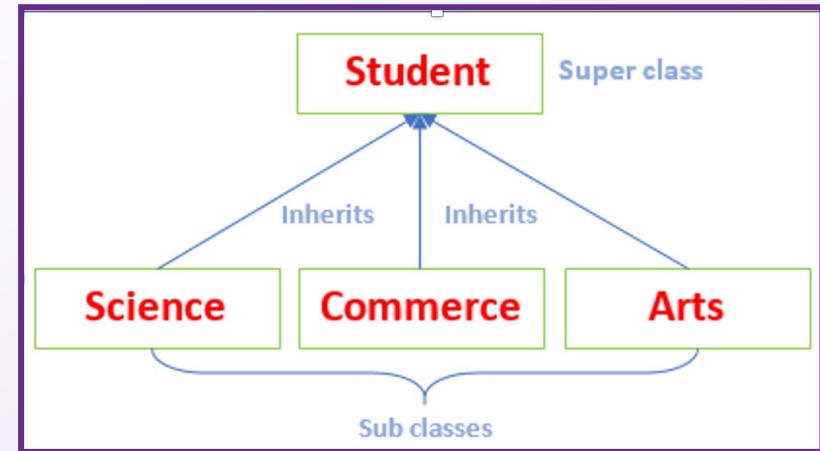
# Inheritance (Kalitim-Miras)

Inheritance Java OOP Concept'i olusturan mekanizmalardandir.

Var olan bir Class'daki tum ozellikleri (**variable/method**) baska bir class'dan kullanabilmemizi saglar.

Gercek hayattan farkli olarak, parent class child edinmez, **child class extends keyword** kullanarak parent edinir.

Bir uygulamada parent-child iliskisi 2 durumda olusturulabilir.



**1-** Bir uygulamayı sifirdan tasarliyorsak, ortak ozellikleri barindiracak class'lari parent class olarak tasarlayabiliriz.

**2-** Proje gelistirme asamasinda da sonradan ortaya cikan ihtiyaclar cercevesinde, ozelliklerini kullanmak istedigimiz bir class'i parent edinebiliriz.



# Inheritance / Access Modifier Kullanimi

Inheritance'in bize sagladigi en buyuk avantaj REUSABILITY'dir.

Inheritance'da **genel ozellikler** parent class'larda, **spesifik ozellikler** ise child class'larda olur.

Bir class olusturuldugunda parent class olarak tasarlanmasa bile sonradan parent edinilme ihtimali dusunulerek class uyelerine erisim access modifier ile düzenlenebilir.

- private ve/veya static class uyeleri inherit edilemezler.
- public class uyelerini tum proje icerisinden, default access modifier'ina sahip class uyelerini bulundugu package'dan kullanabiliriz
- Inheritance icin en uygun access modifier protected'dir.

Class olusturulurken ileride parent edinilmesi ihtimali dusunulerek, child class'lardan kullanilmasina izin vermek istedigimiz class uyeleri protected olarak işaretlenebilir.

## ADVANTAGES OF INHERITANCE

- o Sometimes we need to repeat the code or we need repeat the whole class properties. So It helps in various ways.
  - 1.) It saves memory space.
  - 2.) It saves time.
  - 3.) It will remove frustration.
  - 4.) It increases reliability of the code
  - 5.) It saves the developing and testing efforts



# Inheritance

```
public class IK {  
  
    protected String isim = "Deger Atanmamis";  
    protected String soyisim = "Deger Atanmamis";  
  
    protected void maas(){  
        System.out.println("Personel maas alir");  
    }  
  
    protected void izin(){  
        System.out.println("Personelin izin hakki vardir");  
    }  
  
    protected void ozelSigorta(){  
        System.out.println("personelle ozel sigorta destegi yapilir");  
    }  
}
```

Parent  
Class  
(Super)

```
public class Muhasebe extends IK{  
  
    protected String sgkNo = "Deger atanmamis";  
    protected boolean raporluMu;  
    protected int minSaatUcreti = 12;  
  
    protected int fazlaMesai(int fazlaMesaiSuresi){  
  
        return fazlaMesaiSuresi * minSaatUcreti * 2;  
    }  
    protected int standartMaas(){  
  
        return 30 * 8 * minSaatUcreti;  
    }  
}
```

Child Class  
(Sub Class)

```
public class Doktor extends Muhasebe{  
    public static void main(String[] args) {  
  
        Doktor dr = new Doktor();  
        dr.isim = "Recep";  
        dr.soyisim = "Yilmaz";  
        dr.raporluMu = false;  
        dr.sgkNo = "3423455234";  
        dr.minSaatUcreti = 25;  
  
        System.out.println(dr.standartMaas()); // 6000 30*25*8  
    }  
}
```

Child Classes  
(Sub Classes)

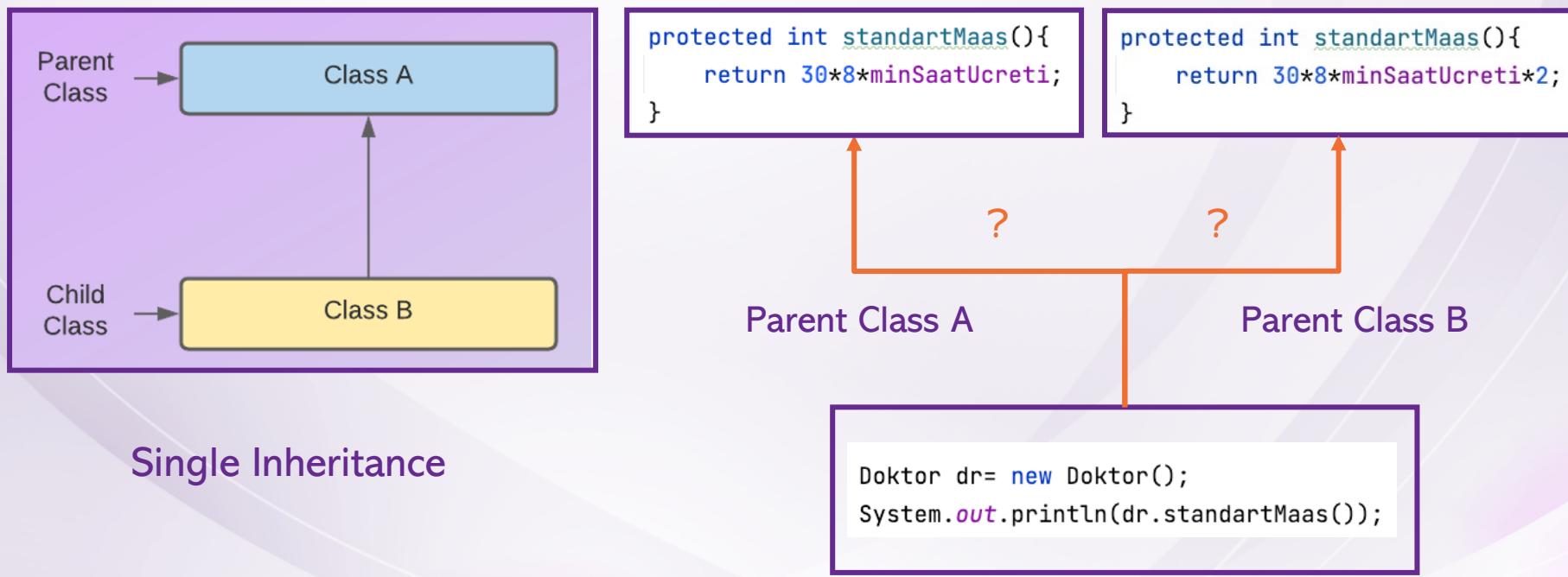
```
public class Hemshore extends Muhasebe{  
  
    public static void main(String[] args) {  
  
        Hemshore hmsr1 = new Hemshore();  
  
        hmsr1.isim = "Ayse";  
        hmsr1.soyisim = "Yilmaz";  
    }  
}
```

Parent Class  
(Super)



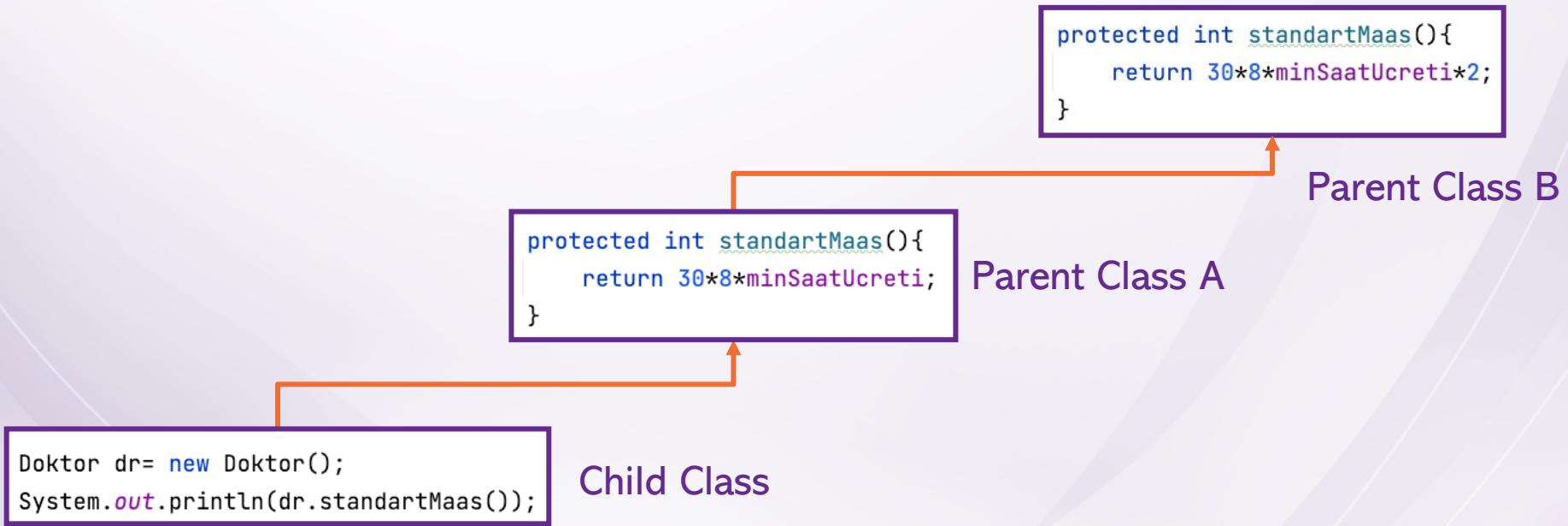
# Inheritance Türleri

Java single inheritance'i kabul eder, multiple inheritance'i kabul etmez. (Bir çocugun anne-babası bir tanedir)



# Inheritance Türleri

Eğer sadece bir parent class ihtiyacımızı karşılamıyorsa, tasarım **multilevel inheritance** olarak yapılabilir.



Bir class'in parent class'i, grand parent class'i ve grand grand parent class'i olabilir.

Birden fazla parent'da aynı özellik varsa ilk bulduğu kullanır.



# Inheritance Türleri

Bir child class'in biden fazla parent class'i olamaz ancak, birden fazla child class aynı class'i parent edinebilirler..

```
public class A {  
    protected static int sayi=10;  
}
```

Parent Class

```
class B extends A{  
  
    public static void main(String[] args) {  
        System.out.println(sayi);  
    }  
}
```

Child Class A

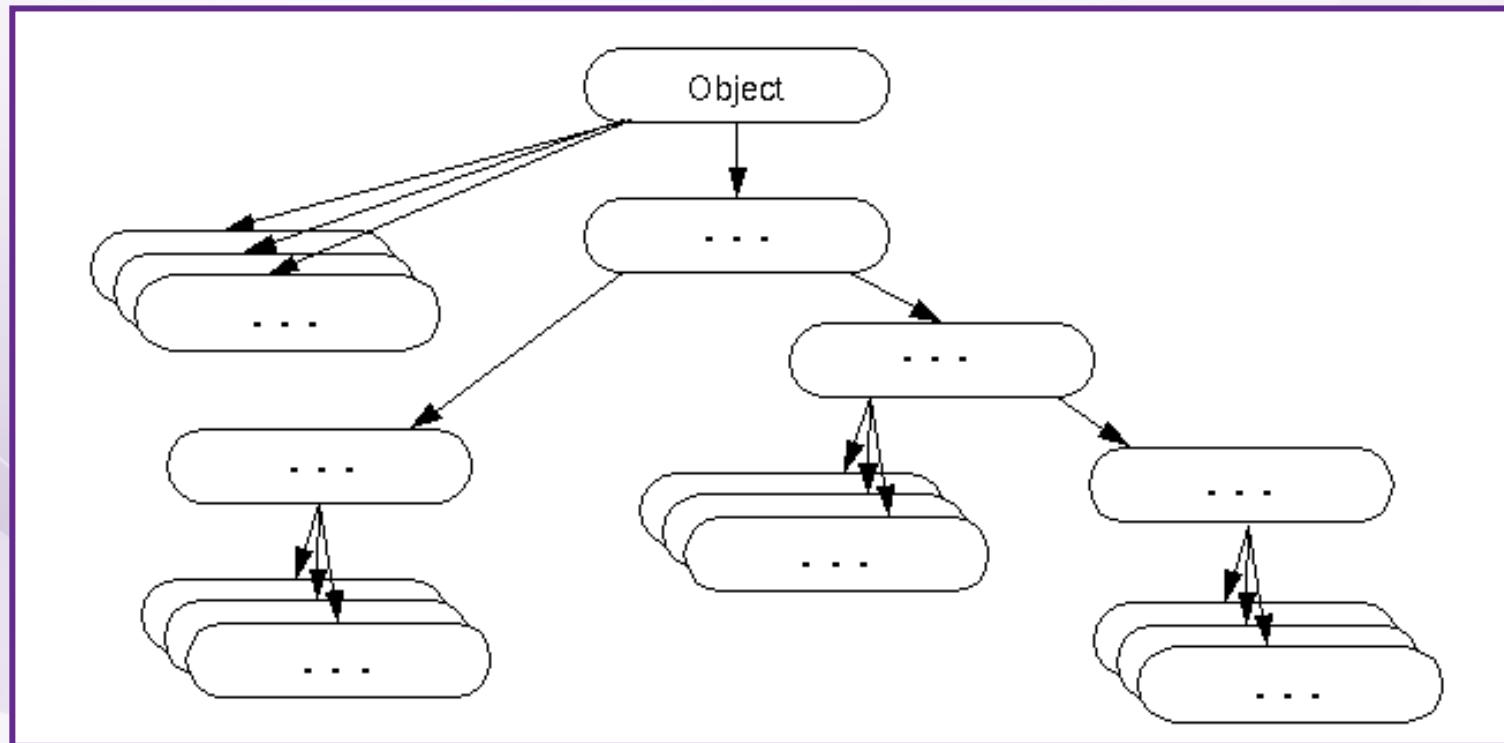
```
class C extends A{  
  
    public static void main(String[] args) {  
        sayi=25;  
    }  
}
```

Child Class B



# Inheritance

Java'da tüm class'lar Object class'ının child class'idirlar. Java oluşturulan tüm class'lari Object class'ından inherit eder.

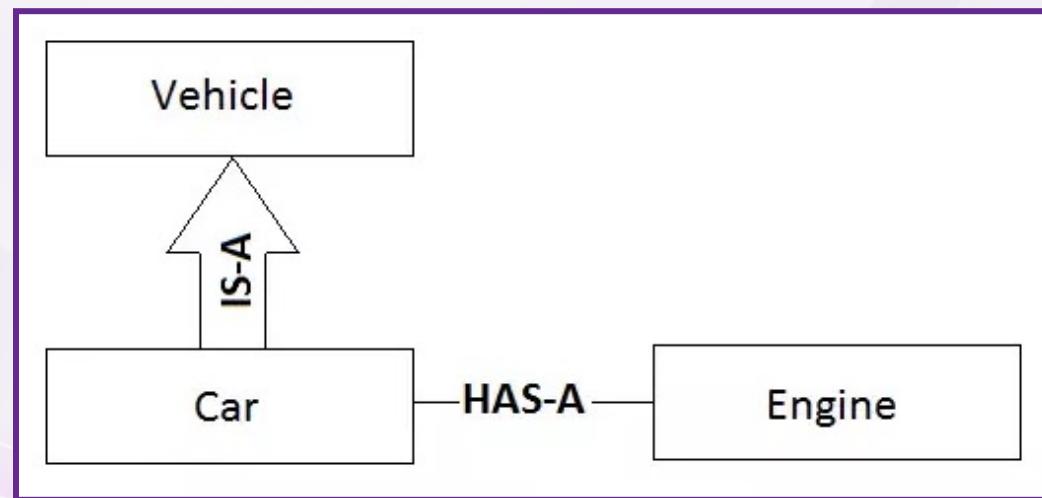


# IS-A & HAS-A Relationship

IS A ve HAS A ilişkisi, parent ve child arasında kullanılan bir tanımlamadır.

Corolla IS A Toyota, Toyota IS A car, car IS A vehicle... child'dan parent'a doğru bir ilişki belirtir.

Apartman HAS A daire, daire HAS A mutfak, mutfağın HAS A dolap... parent'dan child'an doğru bir ilişki belirtir.





Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-35

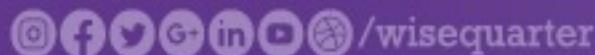
Inheritance  
Constructor Call



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





# Onceki Dersten Aklimizda Kalanlar

- 1- Inheritance : OOP konsept kapsaminda bir class'in baska bir class'daki tum class uyelerine sahip olabilecegi bir konseptir.
- 2- Java'daki inheritance insanlardan farkli calisir. Insanlardaki gibi parent'lar child edinmez, child class'lar ozelliklerine sahip olmak istedikleri bir class'i parent edinirler. Bunun icin extends keyword kullanilir
- 3- Java'da bir class ANCAK bir class'i parent edinebilir ( Java multiple inheritance kabul etmez), ama birden fazla class'daki ozellikleri almak isterse hiyerarsik olarak multi level inheritance kullanabiliriz.
- 4- child class ile parent class arasında IS-A relationship, parent ile child arasında HAS-A relationship vardir.
- 5- Genel olarak 2 durumda inheritance kullanilir
  - Eger yeni bir proje olusturuyorsak ve bazi class'lari bir cati altında toplamamiz gerekiyorsa
  - Devam eden bir projede yeni olusturulan bir class'in eskiden olusturulan bir class'daki tum ozellikleri almasini istiyorsak
- 6- Bir class baska bir class'I parent edindiginde, oncelikle parent class'daki tum ozelliklere sahip olur, ama isterse o ozellikleri kendisine uyardayabilir ve yeni ozellikler ekleyebilir.



# Inheritance'da Constructor Call

```
public class AToyota {  
  
    AToyota(){  
        System.out.println("Toyota cons");  
    }  
    String marka="Toyota";  
    String uretimYeri="Belirtilmemis";  
}
```

```
public class BCorolla extends AToyota{  
  
    BCorolla(){  
        System.out.println("Corolla cons");  
    }  
    String model="Corolla";  
    String uretimYeri="Turkiye";  
}
```

```
public class CDizelCorolla extends BCorolla{  
  
    CDizelCorolla(){  
        System.out.println("dizel corolla cons");  
    }  
  
    public static void main(String[] args) {  
  
        CDizelCorolla arb1=new CDizelCorolla();  
  
        System.out.println(arb1.marka); // Toyota  
        System.out.println(arb1.uretimYeri); // Turkiye  
        System.out.println(arb1.model); // Corolla  
    }  
}
```

```
Toyota cons  
Corolla cons  
dizel corolla cons  
Toyota  
Turkiye  
Corolla
```

Bir child class'da obje olusturdugumuzda bu obje sadece kendi class'indaki ozelliklere degil tum parent class'lardaki ozelliklere de erisebilir

Bunun altinda Java'daki super( ) constructor call mekanizmasi yatomaktadir.



# Inheritance'da Constructor Call

```
public class AToyota {  
  
    AToyota(){  
        System.out.println("Toyota cons");  
    }  
  
    String marka="Toyota";  
    String uretimYeri="Belirtilmemis";  
}
```

```
public class BCorolla extends AToyota{  
  
    BCorolla(){  
        super();  
        System.out.println("Corolla cons");  
    }  
  
    String model="Corolla";  
    String uretimYeri="Turkiye";  
}
```

```
public class CDizelCorolla extends BCorolla{  
  
    CDizelCorolla(){  
        super();  
        System.out.println("dizel corolla cons");  
    }  
  
    public static void main(String[] args) {  
  
        CDizelCorolla arb1=new CDizelCorolla();  
  
        System.out.println(arb1.marka); // Toyota  
        System.out.println(arb1.uretimYeri); // Turkiye  
        System.out.println(arb1.model); // Corolla  
    }  
}
```

```
Toyota cons  
Corolla cons  
dizel corolla cons  
Toyota  
Turkiye  
Corolla
```

Bir class olusturdugumuzda Java'nin o class'a default constructor yerlestirdigi gibi, extends keyword kullanilan class'larda olusturulan her bir constructor'in ilk satirina da parametresiz **super ()** constructor call yerlestirir



# Inheritance'da Constructor Call

```
public class DVolvo {  
    DVolvo(){  
        System.out.println("Volvo parametresiz cons");  
    }  
  
    DVolvo(String abs){  
  
        System.out.println("Volvo parametreli cons");  
    }  
    String marka="volvo";  
}
```

**Kural 1 :** Java'nin class'lara otomatik olarak koydugu default constructor'in parametresiz oldugu gibi extends keyword kullanilan class'larda olusturulan her constructor'in ilk satirina konulan super() de parametresizdir.

```
public class EXC90 extends DVolvo{  
  
    String model="xc90";  
  
    EXC90(String renk){  
        System.out.println(renk + " bir XC90");  
    }  
  
    public static void main(String[] args) {  
        EXC90 arb1=new EXC90( renk: "Kirmizi");  
    }  
}
```



```
Volvo parametresiz cons  
Kirmizi bir XC90
```



# Inheritance'da Constructor Call

```
public class DVolvo {  
    DVolvo(){  
        System.out.println("Volvo parametresiz cons");  
    }  
  
    DVolvo(String abs){  
  
        System.out.println("Volvo parametreli cons");  
    }  
    String marka="volvo";  
}
```



```
public class FXC90 extends DVolvo{  
    String model="XC90";  
    FXC90(){  
    }  
    FXC90(String renk){  
        System.out.println(renk+ " renginde bir "+model);  
    }  
    FXC90(String renk, String yakit){  
        // super(String renk, String yakit); parent class'da yok CTE olusur  
        this( renk: "mavi");  
        System.out.println(renk+ " renginde bir "+ yakin+"li " +marka);  
    }  
    public static void main(String[] args) {  
        FXC90 arb1=new FXC90( renk: "Kirmizi", yakit: "Benzin");  
    }  
}
```

**Kural 2 :** Java'nin child class'lardaki constructor'lara otomatik olarak koymak istedigimiz parametresiz super( ) yerine farkli constructor call kullanabiliriz.

**Kural 3 :** Cagirdigimiz constructor parent class'da yoksa CTE olusur.

**Kural 4 :** Biz super( ) disinda herhangi bir constructor call yaparsak bir constructor'in icinde sadece 1 tane constructor call olabileceginden Java'nin olusturdugu super( ) silinir

**Kural 5 :** Her constructor'da super( ) olmak zorunda degildir, this( ) de olabilir.

this( ) icinde bulundugumuz class'daki constructor'lari calistirir, super( ) ise parent class'daki constructor'lari calistirir



# Inheritance'da Constructor Call

Asagidaki 3 class islevsel olarak birbiri ile aynidir.

```
public class HXC90 extends DVolvo{  
}
```

```
public class HXC90 extends DVolvo{  
  
    HXC90(){  
  
    }  
}
```

```
public class HXC90 extends DVolvo{  
  
    HXC90(){  
        super();  
    }  
}
```



# Inheritance'da Constructor Call

Soru : Asagidaki Sinif class'i calistirilirsa consol'da ne yazdirir ?

```
public class Okul {  
    public Okul() {  
        System.out.println("Parent class cons.");  
    }  
}
```

```
class Sinif extends Okul {  
    public Sinif(int age) {  
        super();  
        System.out.println("child class parametreli cons.");  
    }  
    public Sinif() {  
        this(11);  
        System.out.println("child class parametresiz cons.");  
    }  
  
    public static void main(String[] args) {  
        Sinif sinif1=new Sinif();  
    }  
}
```



# Inheritance'da Constructor Call

Soru : Asagidaki Sinif class'indaki CTE'ler nasıl düzelttilir ve calistirilirsa consol'da ne yazdirir ?

```
class Okul {  
    Okul(String s){  
        System.out.println("Okul : " + s );  
    }  
}
```

```
public class Sinif extends Okul{  
    public Sinif(String k){  
        System.out.println("Sinif: " + k);  
    }  
  
    public static void main(String[] args) {  
        Sinif obj=new Sinif();  
    }  
}
```



# Inheritance'da Constructor Call

Soru : Asagidaki Sinif class'indaki CTE'ler nasil düzelttilir ve calistirilirsa consol'da ne yazdirir ?

```
class Okul {  
    Okul(String s){  
        System.out.println("Okul : " + s );  
    }  
}
```

```
public class Sinif extends Okul{  
    public Sinif(String k){  
        super(k);  
        System.out.println("Sinif: " + k);  
    }  
  
    public static void main(String[] args) {  
        Sinif obj=new Sinif( k: "8.sinif");  
    }  
}
```

Okul : 8.sinif  
Sinif: 8.sinif



# Inheritance'da Class Uyelerini Kullanma

Java'da yazdigimiz kodlarda, herhangi bir satirda bir variable'in hangi degeri alacagini belirlemek icin Java asagidaki siralama ile variable'i arar ve ilk buldugu degeri kullanir.

```
class Okul {  
    String okulIsmi="Yildiz koleji";  
    String telefon="3123423454";  
}  
  
public class Sinif extends Okul{  
    String sinif="11-F";  
    String telefon="3125676789";  
  
    public static void main(String[] args) {  
        Sinif obj=new Sinif();  
        System.out.println(obj.sinif); // 11-F  
        System.out.println(obj.telefon); // 3125676789  
        System.out.println(obj.okulIsmi); // Yildiz koleji  
    }  
}
```

1- icerisinde oldugumuz scope

2- icerisinde oldugumuz class level'daki instance variable'lar

3- parent class'daki instance variable'lar



# Inheritance'da Class Uyelerini Kullanma

```
class Okul {  
    String okulIsmi="Yildiz koleji";  
    String telefon="3123423454";  
}  
  
public class Sinif extends Okul{  
    String sinif="11-F";  
    String telefon="3125676789";  
    Sinif(String telefon){  
        System.out.println(telefon); // 3127658798  
        System.out.println(this.telefon); // 3125676789  
        System.out.println(super.telefon); // 3123423454  
    }  
  
    public static void main(String[] args) {  
  
        Sinif obj=new Sinif(telefon: "3127658798");  
    }  
}
```

Variable'i constructor içerisinde kullanacaksak ve bu sıralamaya uymak istemezsek, kullanmak istediğimiz variable'i kendimiz belirleyebiliriz.

- 1- sadece variable ismini yazarsak içerisinde olduğumuz scope'daki variable'in değerini
- 2- this.variableismi'ni yazarsak içerisinde olduğumuz class'daki instance variable'in değerini
- 3- super.variableismi'ni yazarsak parent class'daki instance variable'in değerini bize döndürür.

**NOT:** super( ) ve this ( ) constructor call içerisinde olduğu constructor'in ilk satırında olabilir ve sadece biri kullanılabilir. Ancak this. ve super. , istenilen satırda ve istenilen sayıda kullanılabilir.



# Inheritance'da Class Uyelerini Kullanma

Soru : Aşağıdaki Sınıf class'i çalıştırılırsa consol'da ne yazdırır ?

```
class Okul {  
    public void adresYazdir(){  
        System.out.println("Okul adres");  
    }  
}
```

```
public class Sinif extends Okul{  
    Sinif(){  
        System.out.println("Sinif Constructor");  
        super.adresYazdir();  
    }  
    public void adresYazdir(){  
        System.out.println("Sinif adres");  
    }  
  
    public static void main(String[] args) {  
        Sinif obj=new Sinif();  
        obj.adresYazdir();  
    }  
}
```

Sinif Constructor  
Okul adres  
Sinif adres



# Inheritance'da Class Uyelerini Kullanma

Soru : Yandaki Sinif class'i calistirilirsa consol'da ne yazdirir ?

```
class Okul {  
    Okul(){  
        System.out.println("Okul Constructor");  
    }  
  
    protected int sayi1=10;  
    protected int sayi2=15;  
    protected String isim1="Ali";  
    protected String isim2="Veli";  
}
```

```
public class Sinif extends Okul{  
    int sayi1=30;  
    int sayi3=40;  
    String isim2="Ayse";  
    String isim3="Fatma";  
    Sinif(){  
        System.out.println("Sinif Constructor");  
  
        System.out.println(this.sayi1);  
        System.out.println(super.sayi1);  
        System.out.println(this.sayi2);  
        System.out.println(super.sayi2);  
        System.out.println(this.sayi3);  
        System.out.println(super.sayi3);  
        super.isim1="Hatice";  
        System.out.println(this.isim1);  
        System.out.println(super.isim1);  
        this.isim2="Kemal";  
        System.out.println(this.isim2);  
        System.out.println(super.isim2);  
        System.out.println(this.isim3);  
        System.out.println(super.isim3);  
    }  
    public static void main(String[] args) {  
        Sinif obj=new Sinif();  
    }  
}
```

Okul Constructor

Sinif Constructor

30

10

15

15

40

Hatice

Hatice

Kemal

Veli

Fatma



# Inheritance / Genel Tekrar

1- Inheritance'in avantajlari nelerdir ?

- A) Reusability B) Maintenance C) Less Code

2- Bir Class'a Parent Class olusturmak icin Syntax nedir?

public class ChildClassIsmi extends ParentClassIsmi

3- Hangi access modifier'lar inherit edilebilir ?

public ve protected olanlar her yerden, default olanlar ayni paketten inherit edilebilir.

4- super( ) ile this( )'in farki nedir?

super( ) parent class'dan, this( ) ise içinde bulunan class'dan constructor çağırma için kullanılır

5- super( ) ile super.'nin farki nedir?

super( ) parent class'dan constructor, super. ise variable veya method çağırma için kullanılır

6- this( ) ile this.'nin farki nedir?

this( ) constructor, this. ise class variable veya method'u çağırma için kullanılır



# Inheritance / Genel Tekrar

7- super. ile this.'nin farki nedir?

super. parent class'dan variable veya method cagirmak icin kullanilir, this. ise icinde bulunulan class'da class level variable veya method'lari cagirmak icin kullanilir.

this. veya super. kullanilmazsa Java standart kullanım seklinde anlar, kontrol etmesi gereken scope'lari sirasiyla kontrol eder ve ilk buldugu degeri kullanir.

8- super( ) ve this( ) bulunduklari constructor'da ilk sirada olmalıdır. True / False ~~True / False~~

9- super( ) ve this( ) bir constructor'da sadece 1 kere kullanilabilir. True / False ~~True / False~~

10- super( ) ve this( ) ayni constructor'da birlikte kullanilabilir. True / False ~~True / False~~



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-36

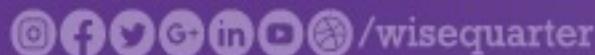
Inheritance Datatype Kullanımı  
Overriding



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



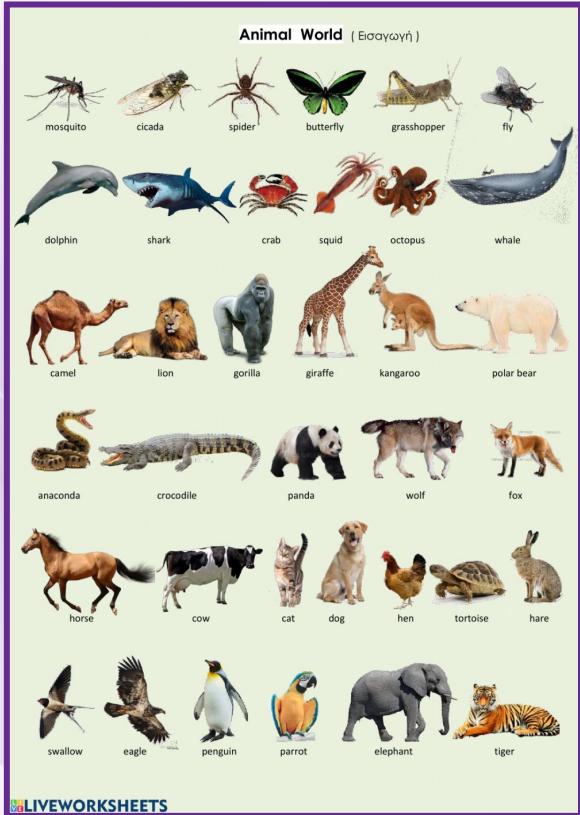


# Onceki Dersten Aklimizda Kalanlar

- 1- Inheritance kullanan bir child class'dan olusturulan tum objeler, olusturulduklari class'daki ozelliklerle birlikte, parent class(lar)'daki ozelliklere de sahip olur.
- 2- Bir obje olusturuldugunda class'daki instance variable'larin birer kopyasini alip, obje ile iliskilendiren Constructor'dir.
- 3- Child class'da olusturulan her obje icin parent class'daki constructor'larin da calismasi gereklidir ki o class'lardaki ozelliklere sahip olabilsin
- 4- Java'da extends keyword kullanican bir class'da var olan veya olusturulan her constructor'in ilk satirinda constructor call OLMALIDIR.
- 5- Kodu yazan kisi constructor'in ilk satirina programa uygun bir constructor call yazabilir.
- 6- Kodu yazan kisi constructor'in ilk satirina constructor call koymazsa Java otomatik olarak parent class'daki parametresiz constructor'i calistiran super( );'u koyar.
- 7- Bir child class'da local olarak bir variable ismi yazildiginda, Java bu variable'in degerini bulabilmek icin sirasiyla A- o scope'a B- Class level'a C- parent class(lar)a bakar
- 8- eger this.variable ismi yazilrsa, scope'a bakmadan direk class level'dan baslar
- 9- eger super.variable ismi yazilrsa scope ve class level pass gecilir
- 10- arama yaparken hep yukari dogru calisilir, bulunamazsa asagi bakilmaz, CTE olur



# Inheritance'da Data Type Kullanimi



Hayvanlar Alemi

hareket( ) : hareket ederler  
solunum( ) : nefes alırlar  
beslenme( ) : beslenirler  
cogalma( ) : cogalırlar  
omur( ) : yasar ve olurler



Kuslar

kanat( ) : kanatlidirlar  
solunum( ) : akcigerle nefes alırlar  
gaga( ) : gagalari vardir  
cogalma( ) : yumurtayla cogalırlar

hareket( ) : yuruler  
beslenme( ) : et ve ot



Kumes Kuslari



Avcı Kuslar

hareket( ) : ucarlar  
beslenme( ) : et yerler  
pence( ) : pencelidir  
gaga( ) : sivri gagali



# Inheritance'da Data Type Kullanımı





# Inheritance'da Data Type Kullanimi

```
public class AToyota {  
    String marka="Toyota";  
    int uretimYili;  
    String uretimYeri="Deger Atanmadi";  
}
```

```
public class BCorolla extends AToyota{  
    String renk="Belirtilmedi";  
    String uretimYeri="Turkiye";  
    String yakit="Tanimlanmadi";  
}
```

Bir child class'da obje olusturulurken data turu olarak parent class'lar secilebilir.

Boyle bir obje olusturmanin temel amaci constructor kullanan class'dan, parent class'in sahip oldugu ozelliklere sahip bir obje olusturmaktir.

Bu durumda obje child class'in objesi olmakla birlikte ozellikleri data turu olarak secilen parent class'a ait olur.

```
public class CDizelCorolla extends BCorolla{  
    String yakit="Dizel";  
    String motor="1.4 dizel motor";  
    String renk="Tanimlanmadi";
```

```
public static void main(String[] args) {  
  
    CDizelCorolla arb1=new CDizelCorolla();  
    System.out.println(arb1.motor); // C 1.4 dizel motor  
    System.out.println(arb1.renk); // C Tanimlanmadi  
    System.out.println(arb1.yakit); // C Dizel  
    System.out.println(arb1.uretimYeri); // B Turkiye  
    System.out.println(arb1.marka); // A Toyota
```

```
BCorolla corolla= new BCorolla();  
BCorolla arb2=new CDizelCorolla();  
System.out.println(arb2.yakit); // B Tanimlanmadi  
System.out.println(arb2.renk); // B Belirtilmedi  
// arb2.motor data turu olarak secilen  
// BCorolla class'inda motor variable'i olmadiginden CTE verir  
System.out.println(arb2.uretimYeri); // B Turkiye  
System.out.println(arb2.marka); // A Toyota
```

```
AToyota arb3= new CDizelCorolla();  
// arb3.motor A'da motor yok  
// arb3.renk A'da renk yok  
// arb3.yakit A'da yakit yok  
System.out.println(arb3.uretimYeri); // A Deger atanmadi  
System.out.println(arb3.marka); // A Toyota
```



# Inheritance'da Data Type Kullanimi

```
public class AToyota {  
    String marka="Toyota";  
    int uretimYili;  
    String uretimYeri="Deger Atanmadi";  
}
```

```
public class BCorolla extends AToyota{  
    String renk="Belirtilmedi";  
    String uretimYeri="Turkiye";  
    String yakit="Tanimlanmadi";  
}
```

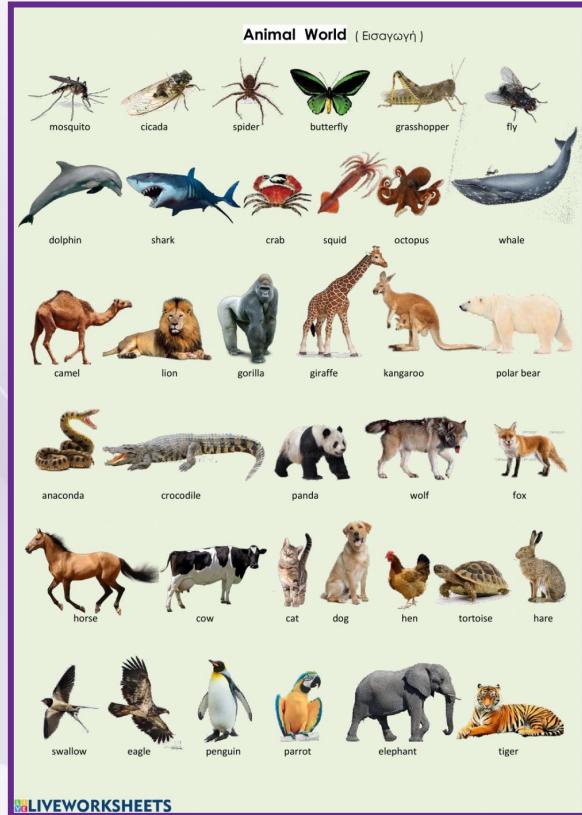
Bir obje olusturulurken data turu constructor'dan farkli ise o objeye ait variable degerlerinin ne oldugunu bulmak icin ilk bakiyamiz gereken scope, data turu olarak secilen parent class olmalidir, o class'da variable bulunamazsa, parent'larina baklilar

Eger aranan variable data turu olarak secilen class ve onun parent'larinda yoksa CTE olusur.

```
public class CDieselCorolla extends BCorolla{  
    String yakit="Dizel";  
    String motor="1.4 dizel motor";  
    String renk="Tanimlanmadi";  
  
    public static void main(String[] args) {  
  
        CDieselCorolla arb1=new CDieselCorolla();  
        System.out.println(arb1.motor); // C 1.4 dizel motor  
        System.out.println(arb1.renk); // C Tanimlanmadi  
        System.out.println(arb1.yakit); // C Dizel  
        System.out.println(arb1.uretimYeri); // B Turkiye  
        System.out.println(arb1.marka); // A Toyota  
  
        BCorolla corolla= new BCorolla();  
        BCorolla arb2=new CDieselCorolla();  
        System.out.println(arb2.yakit); // B Tanimlanmadi  
        System.out.println(arb2.renk); // B Belirtilmedi  
        // arb2.motor data turu olarak secilen  
        // BCorolla class'inda motor variable'i olmadiginden CTE veri.  
        System.out.println(arb2.uretimYeri); // B Turkiye  
        System.out.println(arb2.marka); // A Toyota  
  
        AToyota arb3= new CDieselCorolla();  
        // arb3.motor A'da motor yok  
        // arb3.renk A'da renk yok  
        // arb3.yakit A'da yakit yok  
        System.out.println(arb3.uretimYeri); // A Deger atanmadi  
        System.out.println(arb3.marka); // A Toyota
```



# Overriding



Hayvanlar Alemi

hareket( ) : hareket ederler  
solunum( ) : nefes alırlar  
beslenme( ) : beslenirler  
cogalma( ) : cogalırlar  
omur( ) : yasar ve olurler



Kuslar

kanat( ) : kanatlidirlar  
solunum( ) : akcigerle nefes alırlar  
gaga( ) : gagalari vardir  
cogalma( ) : yumurtayla cogalırlar

hareket( ) : yuruler  
beslenme( ) : et ve ot



Kumes Kuslari



Avcı Kuslar

hareket( ) : ucarlar  
beslenme( ) : et yerler  
pence( ) : pencelidir  
gaga( ) : sivri gagali

# Overriding

Overriding, child class'in parent class'da var olan bir method'u kendisine uyarlamasıdır.

Overriding için parent class'daki method child class'da da oluşturulup body'si değiştirilir.



karpuz



karpuz

## Hayvanlar class

solunum( ) : nefes alırlar

cogalma( ) : cogalırlar

**Overridden Method**  
**(Gecersiz kilan)**

## Kuslar class

solunum( ) : akcigerle nefes alırlar

cogalma( ) : yumurtayla cogalırlar

**Overriding Method**  
**(Gecersiz kilan)**

# Overriding

```
public class AHayvanlar {
    String tur="Hayvan";
    String isim="Hayvan Isim belirtilmemi";
    void hareket(){
        System.out.println("Hayvanlar hareket eder");
    }
    void solunum(){
        System.out.println("Hayvanlar nefes alir");
    }
    void beslenme(){
        System.out.println("Hayvanlar beslenirler");
    }
    void cogalma(){
        System.out.println("Hayvanlar cogalirlar");
    }
    void omur(){
        System.out.println("Hayvanlar yasar ve olur");
    }
}
```

extends

```
public class BKuslar extends AHayvanlar{
    String tur="Kus";
    String isim="Kus Isim belirtilmemi";
    String ayak="Kusların ayagi vardir";
    void kanat(){
        System.out.println("Kuslar kanatlidir");
    }
    void solunum(){
        System.out.println("Kuslar akcigerle nefes alirlar");
    }
    void gaga(){
        System.out.println("Kusların gagalari vardir");
    }
    void cogalma(){
        System.out.println("Kuslar yumurtayla cogalir");
    }
}
```

extends

```
public class CKumesKuslari extends BKuslar{
    String tur="Kumes Kusu";
    String isim="Kumes kusu Isim belirtilmemi";
    String yasamYeri="Kumes kuslari kumeste yasar";
    void hareket(){
        System.out.println("Kumes hayvanları yurur");
    }
    void beslenme(){
        System.out.println("Kumes hayvanları et veya ot yerler");
    }
    public static void main(String[] args) {
        CKumesKuslari kk1= new CKumesKuslari();
        System.out.println(kk1.tur), // C Kumes Kusu
        System.out.println(kk1.isim); // C Kumes kusu Isim belirtilmemi
        System.out.println(kk1.yasamYeri); // C Kumes kuslari kumeste yasar
        System.out.println(kk1.ayak); // B Kusların ayagi vardir
        kk1.hareket(); // C Kumes hayvanları yurur
        kk1.beslenme(); // C Kumes hayvanları et veya ot yerler
        kk1.kanat(); // B Kuslar kanatlidir
        kk1.solunum(); // B Kuslar akcigerle nefes alir
        kk1.gaga(); // B Kusların gagalari vardir
        kk1.cogalma(); // B Kuslar yumurtayla cogalir
        kk1.omur(); // A Hayvanlar yasar ve olur
    }
}
```

Child class'da olusturulan objenin data turu ve constructor'i ayni ise,  
objenin ozelliklerini bulmak icin, objenin ait oldugu class'tan baslayip parent class'lara devam ederiz, ilk buldugumuz deger gecerli olur.



# Overriding

```
public class AHayvanlar {  
  
    String tur="Hayvan";  
    String isim="Hayvan Isim belirtilmemi";  
  
    void hareket(){  
        System.out.println("Hayvanlar hareket eder");  
    }  
    void solunum(){  
        System.out.println("Hayvanlar nefes alir");  
    }  
    void beslenme(){  
  
        System.out.println("Hayvanlar beslenirler");  
    }  
    void cogalma(){  
        System.out.println("Hayvanlar cogalirlar");  
    }  
    void omur(){  
        System.out.println("Hayvanlar yasar ve olur");  
    }  
}
```

extends

```
public class BKuslar extends AHayvanlar{  
  
    String tur="Kus";  
    String isim="Kus Isim belirtilmemi";  
    String ayak="Kuslarin ayagi vardir";  
  
    void kanat(){  
        System.out.println("Kuslar kanatlidir");  
    }  
    void solunum(){  
        System.out.println("Kuslar akcigerle nefes alirlar");  
    }  
    void gaga(){  
        System.out.println("Kuslarin gagalari vardir");  
    }  
    void cogalma(){  
        System.out.println("Kuslar yumurtayla cogalir");  
    }  
}
```

extends

```
public class CKumesKuslari extends BKuslar{  
  
    String tur="Kumes Kusu";  
    String isim="Kumes kusu Isim belirtilmemi";  
    String yasamYeri="Kumes kuslari kumeste yasar";  
  
    void hareket(){  
        System.out.println("Kumes hayvanları yurur");  
    }  
    void beslenme(){  
        System.out.println("Kumes hayvanları et veya ot yerler");  
    }  
    public static void main(String[] args) {  
        BKuslar kk2= new CKumesKuslari();  
        System.out.println(kk2.tur); // B Kus  
        System.out.println(kk2.isim); // B Kus Isim belirtilmemi  
        System.out.println(kk2.yasamYeri); // CTE  
        System.out.println(kk2.ayak); // B Kuslarin ayagi vardir  
        kk2.hareket(); // C Kumes hayvanları yurur  
        kk2.beslenme(); // C Kumes hayvanları et veya ot yerler  
        kk2.kanat(); // B Kuslar kanatlidir  
        kk2.solunum(); // B Kuslar akcigerle nefes alir  
        kk2.gaga(); // B Kuslarin gagalari vardir  
        kk2.cogalma(); // B Kuslar yumurtayla cogalir  
        kk2.omur(); // A Hayvanlar yasar ve olur  
    }  
}
```

Objenin data turu ve constructor'i farklı ise,  
objenin **statik ozelliklerini(variable)** bulmak için,  
data turu seçilen class'tan başlayıp parent  
class'lara devam ederiz, ilk bulduğumuz değer  
gecerli olur.

**Dinamik ozellikler (method)** için ise özellik bulunduktan sonra override edilmiş mi diye  
kontrol ederiz, birden fazla override varsa en güncel hali geçerli olur.



# Overriding

```
public class AHayvanlar {  
  
    String tur="Hayvan";  
    String isim="Hayvan Isim belirtilmemi";  
  
    void hareket(){  
        System.out.println("Hayvanlar hareket eder");  
    }  
    void solunum(){  
        System.out.println("Hayvanlar nefes alir");  
    }  
    void beslenme(){  
  
        System.out.println("Hayvanlar beslenirler");  
    }  
    void cogalma(){  
        System.out.println("Hayvanlar cogalirlar");  
    }  
    void omur(){  
        System.out.println("Hayvanlar yasar ve olur");  
    }  
}
```

```
extends  
  
public class BKuslar extends AHayvanlar{  
  
    String tur="Kus";  
    String isim="Kus Isim belirtilmemi";  
    String ayak="Kuslarin ayagi vardir";  
  
    void kanat(){  
        System.out.println("Kuslar kanatlidir");  
    }  
    void solunum(){  
        System.out.println("Kuslar akcigerle nefes alirlar");  
    }  
    void gaga(){  
        System.out.println("Kuslarin gagalari vardir");  
    }  
    void cogalma(){  
        System.out.println("Kuslar yumurtayla cogalir");  
    }  
}
```

```
extends  
  
public class CKumesKuslari extends BKuslar{  
  
    String tur="Kumes Kusu";  
    String isim="Kumes kusu Isim belirtilmemi";  
    String yasamYeri="Kumes kuslari kumeste yasar";  
  
    void hareket(){  
        System.out.println("Kumes hayvanlari yurur");  
    }  
    void beslenme(){  
        System.out.println("Kumes hayvanlari et veya ot yerler");  
    }  
    public static void main(String[] args) {  
        AHayvanlar kk3=new CKumesKuslari();  
        System.out.println(kk3.tur); // A Hayvan  
        System.out.println(kk3.isim); // A Hayvan Isim belirtilmemi  
        System.out.println(kk3.yasamYeri); // CTE  
        System.out.println(kk3.ayak); // CTE  
  
        kk3.hareket(); // C Kumes hayvanlari yurur  
        kk3.beslenme(); // C Kumes hayvanlari et veya ot yerler  
        kk3.kanat(); // A'da bu method yok CTE  
        kk3.solunum(); // B Kuslar akcigerle nefes alir  
        kk3.gaga(); // A'da gaga yok CTE  
        kk3.cogalma(); // B Kuslar yumurtayla cogalir  
        kk3.omur(); // A Hayvanlar yasar ve olur  
    }  
}
```

Variable veya method aradigimiz class veya parent'larinda bulunamazsa Java CTE verir.

CTE olmadigi durumda, bir child class'da olusturulan objelerin data turleri degistikce static ozellikleri (variable degerleri) de degisir,

ancak dinamik ozellikler(method sonuclari) en guncel halini aldigii icin data turu degisse de ayni sonuclari verirler.



# Overriding

```
public class EParent {  
  
    void method1(){  
        System.out.println("parent method1");  
    }  
  
    void method2(){  
        System.out.println("parent method2");  
    }  
}
```

extends

```
public class FChild extends EParent{  
  
    @Override  
    void method1() {  
        System.out.println("child method1");  
    }  
  
    void method2() {  
        System.out.println("child method1");  
    }  
}
```

intelliJ overriding method urettiginde, @Override notasyonu kullanir. Her ne kadar bu notasyonu kullanmak mecburi olmasa da, kullanmakta fayda vardir. Biz de yazdigimiz overriding method'lara bu notasyonu ekleyebiliriz.

@Override notasyonu kullanilirsa Java bu iki method'u gozetim altinda tutar ve overridden method silinirse CTE verir. Boylece kod calismasi sirasinda ortaya cikacak muhtemel sorunlarin onune gecilir.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-37

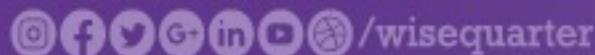
### Overriding Polymorphism



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)





# Onceki Dersten Aklimizda Kalanlar

- 1- Java'daki OOP 4 temel konsept uzerine kuruludur. A- Encapsulation, B- Inheritance, C- Polymorphism, D- Abstraction(Abstract class ve Interface)
- 2- Java'da bir obje olusturup, ilk deger atamasi yapmak icin Constructor calismalidir.
- 3- Bir obje olusturulurken hangi class'in constructor'I calisirsa obje o class'in ozelliklerini tasiyacaktir.
- 4- Java'da bir class'dan constructor calistirilarak obje olusturuldugunda, data type olarak o class ve parent class'lar secilebilir.
  - Variable'lar icin data turu hangi class secildiyse aramaya oradan baslar, parent class'lara dogru ilerleriz ve ilk buldugumuz degeri aliriz. Eger variable bu aramada bulunamazsa, child'a dogru donus olmaz, CTE olusur
  - Method'larda ise yine aramaya data turu olan class'dan baslar, parent'lara dogru ilerler AMMA ilk buldugu degeri hemen almaz, data turu secilen class'dan constructor secilen class'a kadar en guncel halini arar. Ve o degeri alir. Data turu secilen class veya parent'larda o method'u bulamazsa child'a donmez CTE olusur.
- 6- Parent class'da olusturulan bir method'un child class'da yeniden olusturulup, body'si degistirilerek child class'a uyarlanmasina overriding(gecersiz kilma) denir.



## 1- static, final ve private method'lar override edilemez

- Parent class'daki **final** veya **static** olarak işaretlenen bir method ile aynı isimde bir method'u child class'da oluşturursanız Java CTE verir.
- Parent class'da **private** olarak işaretlenen bir method ile aynı isimde bir method child class'da oluşturulsrsa Java CTE vermez.

Parent class'daki method **private** olduğundan ona ulaşılamayacağından aynı isimde bir method olarak görünmez.

Child class'daki method overriding method değil, bağımsız bir method olarak kullanılır.

@Override notasyonu yazmak istenirse Java CTE verir.

```
public class GParent {  
    private void method1(){  
        System.out.println("parent method1")  
    }  
  
    void method2(){  
        System.out.println("parent method2");  
    }  
  
    final void method3(){  
        System.out.println("parent method2")  
    }  
  
    static void method4(){  
        System.out.println("parent method2")  
    }  
}
```

# Overriding Kuralları

```
public class HChild extends GParent{  
  
    void method1() {  
        System.out.println("child method1");  
    }  
  
    void method2() {  
        System.out.println("child method2");  
    }  
  
    void method3() {  
        System.out.println("child method2");  
    }  
  
    void method4() {  
        System.out.println("child method2");  
    }  
  
    public static void main(String[] args) {  
  
        GParent obj = new HChild();  
        obj.method2();  
        obj.method1();  
    }  
}
```



# Overriding Kurallari

2- Child class'daki overriding method'un **access modifier'i** parent class'dakinden daha kısıtlayıcı olamaz



cocuk babayı  
kısıtlayamaz

Yandaki class'lar incelendiginde

Parent class'daki method1 **default access modifier'i**na sahip oldugundan, child class'daki overriding method'un access modifier'i **default, protected veya public** olabilir

Parent class'daki method2 **protected** oldugundan, child class'daki overriding method'un access modifier'i **protected veya public** olabilir

```
public class KParent {  
    void method1(){  
        System.out.println("parent method1");  
    }  
  
    protected void method2(){  
        System.out.println("parent method2");  
    }  
}
```

↑ extends

```
public class LChild extends KParent{  
  
    @Override  
    public void method1() {  
    }  
  
    @Override  
    protected void method2() {  
    }  
}
```



# Overriding Kurallari

3- Child class'daki overriding method'un return type'i parent class'daki ile aynı veya covariant data turunde olabilir.

Covariant data turu : İki data turunun birbiri arasında parent child ilişkisi olması (alt tür) demektir. IS-A relationship ile kontrol edilebilir.

Primitive data turlerinde birbirinin alt turu olma ihtimali olmadiginden, parent ve child class'in return type'lari aynı olmalıdır.

Yandaki örnek incelendiginde parent class'daki method1'in return type'i void, method 2'nin ise primitive data olan int'dir. Dolayisiyla child class'da bu method'lari override eden method'lar da aynı return type'a sahip olmalıdır.

method3'un return type'i ise Object'tir. Child class'da method3'un return type'i Object'in alt turu olan hersey olabilir. (String, list, array, Integer ....)

```
public class MParent {  
    void method1(){  
        System.out.println("parent method1");  
    }  
  
    int method2(){  
        System.out.println("parent method2");  
        return 3;  
    }  
  
    Object method3(){  
        System.out.println("parent method3");  
        return "Java";  
    }  
}
```

extends ↑

```
public class NChild extends MParent{  
  
    @Override  
    void method1() {  
    }  
  
    @Override  
    int method2() {  
        return 5;  
    }  
  
    @Override  
    String method3() {  
        return "Hava";  
    }  
}
```



# Overriding Kurallari

4- Override edilen bir method ile override eden method ikisi birden calismaz.

Eger ikisinin de calismasini istersek, overriding method icine super. Ile overridden method'u calistirabiliriz.

```
public class KParent {  
    void method1(){  
        System.out.println("parent method1");  
    }  
  
    protected void method2(){  
        System.out.println("parent method2");  
    }  
}
```

extends

```
public class LChild extends KParent{  
  
    @Override  
    public void method1() {  
        super.method1();  
        System.out.println("child method1");  
    }  
}
```



# Polymorphism

Polymorphism, bir objenin farkli davranabilme ozelligidir.

Java bir objenin farkli ozellikler kazanabilmesini, method'lara girilecek farkli parametrelere gore, method sonuclarinin degismesi sayesinde yapar.

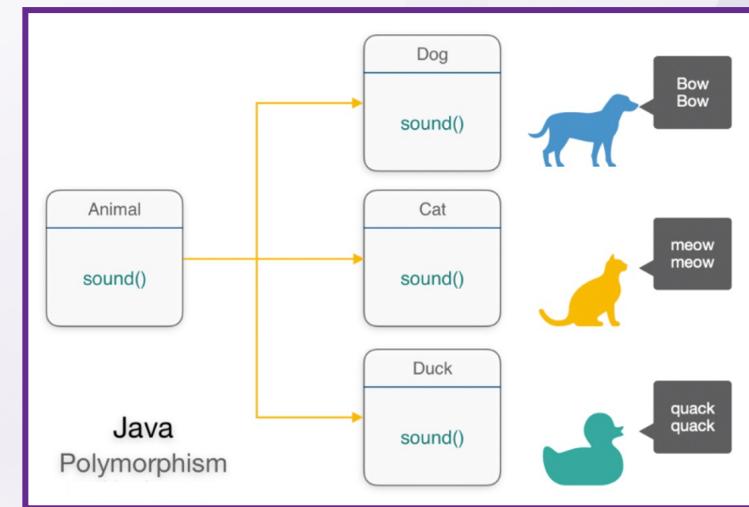
Ayni isimde ama farkli islevli (sonucu farkli) method'lar olusturmak icin 2 olasilik vardir.

## 1 – Overloading (Compile time/static polymorphism)

Ayni class'da ismi ayni, method signature'i farkli methodlar kullanma

## 2 – Overriding (Run time / dinamik polymorphism)

Parent ve child class'da signature'i ayni method body'si farkli methodlar kullanma



# Polymorphism

## Overloading

Aynı class'da aynı isimde fakat farklı signature'lara sahip method'lar oluşturmaktadır.

Signature : method ismi, parametre sayısı ve farklı data türündeki parametrelerin sıralanmasını kapsar

Overloading olması için ismin aynı olması gereklidir, dolayısıyla farklı method'lar oluşturabilmek için parametre sayısı, farklı data türü kullanımı ve/veya farklı data türündeki parametrelerin sıralaması değiştirilebilir.



Overloading'de method body'sindeki kodlar, access modifier, return type veya parametre isimlerinin bir önemi yoktur.

# Polymorphism

## Overloading

Java ayni class'da ayni isim ve signature'a sahip birden fazla method olusturulmasina izin vermez.

Ancak signature degistirilerek yandaki gibi ayni isimde istendigi kadar method olusturulabilir.

Java bu methodlardan hangisinin kullanilacagina, method call sirasinda kullandigimiz argument'lere gore karar verir.

- Argument'ler ile parametrelerin %100 ortustugu method varsa onu kullanir.
- %100 ortusen method bulamazsa casting ile kullanabilecegi method'lara bakar.
- Casting ile kullanabilecegi method sayisi birden fazla ise en az casting yaparak kullanabilecegini tercih eder.

```
public class C01_Overloading {  
    public static void main(String[] args) {  
        C01_Overloading obj = new C01_Overloading();  
        obj.islem( a: 3, b: "Java"); // 3 Java  
        obj.islem( a: "Ali", b: "Can"); // Ali  
        obj.islem( a: "Hava" , b: 5); // 5  
        obj.islem( a: 5, b: 6); // 11.0  
        obj.islem( a: 3.2, b: 5.6); // 8.8  
    }  
    public void islem(double a, double b) {  
        System.out.println(a + b);  
    }  
    public void islem(String a, int b) {  
        System.out.println(b);  
    }  
    public void islem(String a, String b) {  
        System.out.println(a);  
    }  
    public void islem(int a, String b){  
        System.out.println(a + " " + b);  
    }  
}
```



# Polymorphism

## Overriding

Overriding inheritance ile birlikte kullanabilecek bir yöntemdir.

Child class'lara method'lari kendine özelleştirme imkani tanır.

Overriding için parent ve child class'lar birlikte düşünülmeli, objenin data turu ve constructor'i farklı olmalıdır

Overriding için parent ve child class'lardaki method'lар aynı isim ve signature'a sahip olmalıdır.

Overriding method kendisine uyarlamayı method body'sinde yapmalıdır.

```
public class C01_Overloading {  
  
    public void islem(double a, double b) {  
        System.out.println(a + b);  
    }  
    public void islem(String a, int b) {  
        System.out.println(b);  
    }  
    public void islem(String a, String b) {  
        System.out.println(a);  
    }  
    public void islem(int a, String b){  
        System.out.println(a + " " + b);  
    }  
}
```

```
public class C02_OVERRIDING extends C01_Overloading{  
  
    public static void main(String[] args) {  
  
        C02_OVERRIDING obj=new C02_OVERRIDING();  
        obj.islem( a: 4, b: 5); // 20.0  
        obj.islem( a: "Veli", b: "Cem"); // Veli  
        obj.islem( c: "Hasan", d: 4); // Hasan  
  
        C01_Overloading obj2=new C02_OVERRIDING();  
        obj2.islem( a: 5, b: 6.4); // 32.0  
        obj2.islem( a: 5, b: "Ali"); // 5 Ali  
  
        C01_Overloading obj3=new C01_Overloading();  
        obj3.islem( a: 5, b: 6); // 11.0  
        obj3.islem( a: 5, b: "Ali"); // 5 Ali  
    }  
  
    public void islem(double a, double b){  
        System.out.println(a * b);  
    }  
    public void islem(String c, int d){  
        System.out.println(c);  
    }  
}
```



# Polymorphism

Soru 1 - Asagidaki class'da CTE nasıl giderilir? CTE giderilirse konsolda ne yazdirir ?

```
class ParentClass {  
    public void method1(String temp) {  
        System.out.println("Parent class " + temp);  
    }  
  
    public class Child01 extends ParentClass {  
        public int method1(String temp) {  
            System.out.println("Child class " + temp);  
            return 10;  
        }  
  
        public static void main(String[] args) {  
            Child01 obj = new Child01();  
            obj.method1( temp: "Deneme");  
        }  
    }  
}
```



# Polymorphism

Soru 1 - Asagidaki class'da CTE nasıl giderilir? CTE giderilirse konsolda ne yazdirir ?

```
class ParentClass {  
    public void method1(String temp) {  
        System.out.println("Parent class " + temp);  
    }  
}  
  
public class Child01 extends ParentClass {  
    public void method1(String temp) {  
        System.out.println("Child class " + temp);  
    }  
  
    public static void main(String[] args) {  
        Child01 obj = new Child01();  
        obj.method1( temp: "Deneme");  
    }  
}
```

Child class Deneme



# Polymorphism

Soru 2 - Asagidaki class calistirilirsa konsolda ne yazdirir ?

```
class Parent{
    public void method1() {
        System.out.println("Parent class ");
    }
}

public class Child01 extends Parent{

    public void method1() {
        System.out.println("Child class ");
    }

    public static void main(String[] args) {
        Child01 obj=new Child01();
        obj.method1();
    }
}
```

Child class



# Polymorphism

Soru 3 - Aşağıdaki class çalıştırılırsa konsolda ne yazdırır ?

```
class Parent{
    public void method1() {
        System.out.println("Parent class ");
    }
}

public class Child01 extends Parent{

    public void method1() {
        System.out.println("Child class ");
        super.method1();
    }

    public static void main(String[] args) {

        Parent obj = new Child01();
        obj.method1();
    }
}
```

Child class  
Parent class



# Polymorphism

Soru 4 - Asagidaki class'da CTE nasıl giderilir? CTE giderilirse konsolda ne yazdirir ?

```
class Parent{
    protected final void method1() {
        System.out.println("Parent class ");
    }
}

public class Child01 extends Parent{

    final void method1() {
        System.out.println("Child class ");
    }

    public static void main(String[] args) {

        Parent obj = new Parent();
        obj.method1();
    }
}
```



# Polymorphism

Soru 5 - Aşağıdaki class çalıştırılırsa konsolda ne yazdırır ?

```
class Parent{
    public void method1() {
        System.out.println("Parent Class");
    }
}

class Child extends Parent{
    public void method1() {
        System.out.println("Child Class");
    }
}

public class Runner {

    public static void main(String[] args) {
        Parent p=new Child();
        p.method1();
    }
}
```

Child class



# Polymorphism

Soru 6 - Asagidaki class calistirilirsa  
konsolda ne yazdirir ?

```
public class Runner {  
    public static void main(String[] args) {  
        C obj1=new C(); obj1.create();  
        U obj2=new D(); obj2.update();  
        R obj3=new R(); obj3.read();  
        new D().delete();  
    }  
}  
class C {  
    public void create() {  
        System.out.println("c");  
    } }  
class U {  
    void update() {  
        System.out.println("u");  
    } }  
class R extends C {  
    public void create() {  
        System.out.println("C");  
    }  
    protected void read() {  
        System.out.println("R");  
    } }  
class D extends U {  
    void update() {  
        System.out.println("U");  
    }  
    void delete() {  
        System.out.println("D");  
    } }
```

C  
U  
R  
D



# Polymorphism

Soru 7 - Aşağıdaki class çalıştırılırsa konsolda ne yazdırır ?

```
class Super{  
  
    public Integer getInt() {  
        return 10;  
    }  
}  
  
public class Runner extends Super{  
public Integer getInt() {  
    return 20;  
}  
  
public static void main(String[] args) {  
    Super obj=new Runner();  
    Runner chld=new Runner();  
    System.out.println(obj.getInt() +  
        ", " + chld.getInt());  
}
```

20, 20



Soru 8 - Yandaki class  
calistirilirsa konsolda ne  
yazdirir ?

```
class Super{  
    public void m1() {  
        System.out.println("m1, Super class");  
    }  
}  
  
class Child extends Super {  
    public void m1() {  
        System.out.println("m1, Child class");  
    }  
  
    public void m2() {  
        System.out.println("m2, Child class");  
    }  
}  
  
public class Runner {  
    public static void main(String[] args) {  
        Super spr = new Super();  
        Child chld = new Child();  
        chld.m2();  
        spr.m1();  
        chld.m1();  
        spr = chld;  
        spr.m1();  
    }  
}
```

m2, Child class  
m1, Super class  
m1, Child class  
m1, Child class



# Onceki Dersten Aklimizda Kalanlar

1- Inheritance'da bir class'dan olusturulan objelerin data turu olarak o class veya parent class(lar)i secilebilir. Burada amac child class'dan olusturulan objenin data turu olarak secilen class'daki ozellikleri kullanabilmesidir.

2- Bir obje olusturulurken constructor ve data turu ayni class ise objenin ozelliklerini. (variable/method) bulmak icin o class'a bakariz, bulamazsa parent class(lar)a dogru yukari cikariz

3- Obje olusturulurken data turu ve constructor farkli ise aramaya data turu olan class'dan baslanir

- variable'lar icin o class'dan baslanir ve ilk bulunan kullanilir.
- method'lar icin o class'dan baslanir aranan method bulunursa hemen kullanilmaz, daha gunceli var mi ? diye constructor'in oldugu class'a kadar inilir. Eger data turu olan class'da bulunamazsa data turunun parent'larina bakilir, orada da bulunamazsa CTE verir. (Constructor kullanilan class'da o method'un olmasi durumu degistirmez)

4- Overriding : Parent class'da var olan bir method'un child class tarafindan update edilmesi yani parent class'daki method'un gecersiz kilinmasidir.



## Soru 9 – Yandaki class calistirilirsa konsolda ne yazdirir ?

```
class GP{
    int islem(int a,int b) {
        if(a>b)
            return a;
        else
            return b;
    }
}

class P extends GP{
    int islem(int a , int b) {
        return a*b;
    }
}

class C extends P{
    int islem(int a , int b) {
        return b - a;
    }
}

public class Runner {
    public static void main(String[] args) {
        P obj1 = new C();
        System.out.println(obj1.islem( a: 100 , b: 200));
        GP obj2 = new P();
        System.out.println(obj2.islem( a: 200 , b: 300));
    }
}
```

100

60000



Soru 10 - Yandaki class  
calistirilirsa konsolda ne  
yazdirir ?

# Polymorphism

```
class GP{  
    int product(int i) {  
        int result=i*i;  
        System.out.println( "GP : "+i + ", " + result );  
        return result;  
    }  
    class P extends GP{  
        int product(int i) {  
            int result=i+i;  
            System.out.println("P : " +i + ", " + result);  
            return result;  
        }  
        class C extends P {  
            int product(int i) {  
                int result = i * 2;  
                System.out.println("C : " + i + ", " + result);  
                return result;  
            }  
            public class Runner {  
                public static void main(String[] args) {  
                    GP obj1 = new GP();  
                    P obj2 = new P();  
                    GP obj3 = new C();  
                    C obj4 = new C();  
                    obj1.product( i: 1);  
                    obj2.product( i: 2);  
                    obj3.product( i: 3);  
                    obj4.product( i: 4);  
                }  
            }  
        }  
    }  
}
```

|           |
|-----------|
| GP : 1, 1 |
| P : 2, 4  |
| C : 3, 6  |
| C : 4, 8  |



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-38

### Abstract Classes



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# Abstract Classes

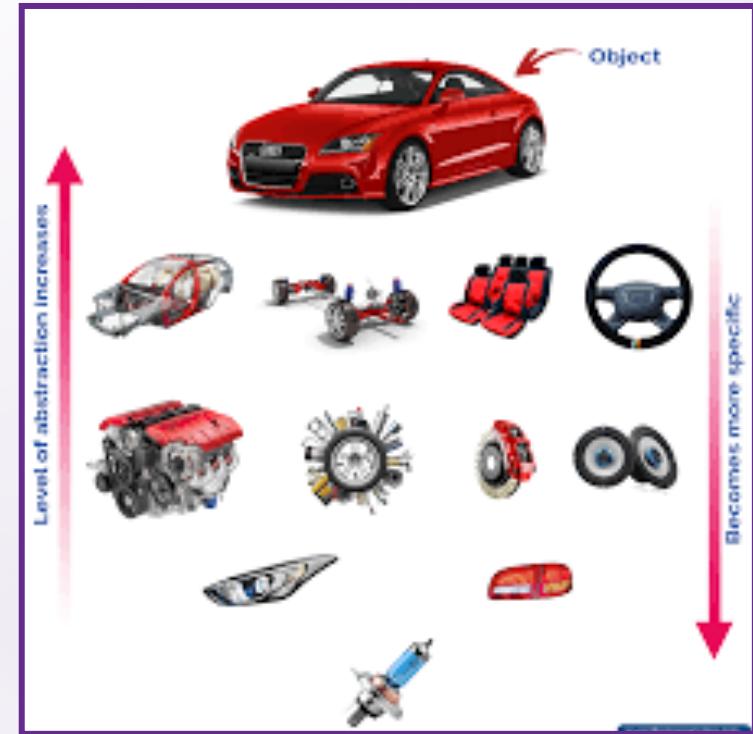
Abstraction (soyutluk) bir mecburiyet degil, islerimizi duzenlememize yarayan bir kolayliktir.

Eger bir class child class'larinda MUTLAKA OLMASI GEREKEN method'lari belirlemek istiyorsa abstraction kullanilir.

Java'da abstraction kullanabilecegimiz iki yapi vardir.

1- Abstract Classes (Partial abstraction)

2- Interfaces(Full abstraction)



Inheritance ile parent class'daki dinamik ozellikleri child class'a uyanabilir, buradaki temel hedef bu ozellikleri inherit etmeyi ZORUNLU HALE getirmektir.

# Abstract Classes

Ekmek cesitleri :

- Tam buğday ekmeği
- Ekşi mayalı ekmek
- Kepek ekmeği
- Çavdar ekmeği
- Tandır ekmeği
- Anatolia Baget.
- Arpa-Tam Tahıl Ekmeği.
- Fransız Baget.
- Siyez Ekmek.
- Chia Tohumlu Ekmek.
- Kavılıca.
- Keimkraft Ekmek.



Bu kadar çok çeşit olunca bir unlu mamulun ekmek olarak tanımlanabilmesi için gerekli özelliklerin belirlenmesi gerekmektedir.

## EKMEK NEDİR ?

Türkiye'de ekmek üretimi, Türk Gıda Kodeksi **Ekmek ve Ekmek Çeşitleri Tebliği**'ne göre yapılmaktadır.



### Bu Tebliğ:

- Buğday unundan veya
- Buğday ununa **düger tahıl unları karıştırılarak** yapılmış ekmeği,
- Ekmek çeşitlerini ve diğer ekmek çeşitleri ile ekşi hamur ekmeklerini kapsamaktadır.



# Abstract Classes



Public Class Ekmek

un( )  
maya( )  
tuz( )  
bicim( )  
gramaj( )



Public Class CevizliEkmek

un( )  
maya( )  
tuz( )  
bicim( )  
gramaj( )  
ekMalzeme( )



Public Class TandirEkmegi

un( )  
maya( )  
tuz( )  
bicim( )  
gramaj( )

# Abstract Classes

Inheritance kullanarak child class'lari parent class'daki method'lari override etmeye mecbur birakamazsiniz.

```
public class AEkmek {  
    public void un( ){  
        System.out.println("Ekmekte un kullanilir");  
    }  
    public void maya( ){  
        System.out.println("Ekmekte maya kullanilir");  
    }  
    public void tuz( ){  
        System.out.println("Ekmekte tuz kullanilabilir");  
    }  
    public void bicim( ){  
        System.out.println("Ekmeklerin standart bir bicimi olur");  
    }  
    public void gramaj( ){  
        System.out.println("Ekmeklerin turune gore gramaj degisir");  
    }  
}
```

```
public class BCevizliEkmek extends AEkmek{  
  
    public static void main(String[] args) {  
  
        BCevizliEkmek ekmek1=new BCevizliEkmek();  
        ekmek1.un();  
        ekmek1.bicim();  
        ekmek1.gramaj();  
        ekmek1.bicim();  
        ekmek1.tuz();  
    }  
}
```

Child class isterse method'u override edip kendisine uyarlar, isterse de direkt parent class'daki haliyle kullanir.



# Abstract Classes

Java'da child class'lari parent class'daki method'lari override etmeye mecbur kilmak icin abstract class veya interface kullanilir.

Abstract class ve interface'ler obje olusturmak icin degil, SADECE child class'larini, parent'da belirlenen method'lari kendilerine uyardamaya mecbur etmek icin kullanilir.

```
public abstract class CAbstractEkmek {  
  
    public abstract void un();  
  
    public abstract void maya();  
  
    public void tuz(){  
  
        System.out.println("Ekmekte tuz kullanilabilir");  
    }  
  
    public abstract void bicim();  
    public abstract void gramaj();  
}
```

```
public class DChildOfAbstractEkmek extends CAbstractEkmek{  
    public void un() {  
        System.out.println("Cevizli ekmek kepekli undan yapilir");  
    }  
    public void maya() {  
        System.out.println("Cevizli ekmekte binde 2 oraninda maya kullanilir");  
    }  
    public void bicim() {  
        System.out.println("Cevizli ekmek somun ekmek seklindedir");  
    }  
    public void gramaj() {  
        System.out.println("Cevizli ekmek icin gramaj 650 gr.");  
    }  
    public static void main(String[] args) {  
        DChildOfAbstractEkmek ekmek1= new DChildOfAbstractEkmek();  
        ekmek1.bicim();  
        ekmek1.gramaj();  
        ekmek1.un();  
        ekmek1.maya();  
    }  
}
```

**Kural 1 :** Abstract bir class'i parent edinen concrete class'lar, abstract class'da abstract olarak tanimlanmis tum method'lari override etmek zorundadir.



# Abstract Classes

Kural 2 : Bir class'i abstract class yapmak icin class keyword'den once abstract keyword kullanilir.

Kural 3 : Bir method'u abstract method yapmak icin return type'dan once abstract keyword kullanilir.

Abstract method'lar mutlaka child class'larda override edileceginden body'si olmaz

```
public abstract class CAbstractEkmek {  
  
    public abstract void un( );  
  
    public abstract void maya( );  
  
    public void tuz( ){  
  
        System.out.println("Ekmekte tuz kullanilabilir");  
    }  
  
    public abstract void bicim( );  
    public abstract void gramaj( );  
}
```

```
public class DChildOfAbstractEkmek extends CAbstractEkmek{  
  
    public void un() {  
        System.out.println("Cevizli ekmek kepekli undan yapilir");  
    }  
    public void maya() {  
        System.out.println("Cevizli ekmekte binde 2 oraninda maya kullanilir");  
    }  
    public void bicim() {  
        System.out.println("Cevizli ekmek somun ekmek seklindedir");  
    }  
    public void gramaj() {  
        System.out.println("Cevizli ekmek icin gramaj 650 gr.");  
    }  
  
    public static void main(String[] args) {  
        DChildOfAbstractEkmek ekmek1= new DChildOfAbstractEkmek();  
        ekmek1.bicim();  
        ekmek1.gramaj();  
        ekmek1.un();  
        ekmek1.maya();  
    }  
}
```

# Abstract Classes

**Kural 4 :** Abstract class'lar abstract veya concrete(abtract olmayan) method'lar bulundurabilirler. Ancak concrete class'larda abstract method ollusturulamaz.

**Kural 5 :** Bir abstract class'daki abstract method'lar override edilmek zorundadir ancak, abstract class'daki concrete method'lar icin mecburiyet yoktur(opsiyonel).

```
public abstract class CAbstractEkmek {
    public abstract void un();
    public abstract void maya();
    public void tuz() {
        System.out.println("Ekmekte tuz kullanilabilir");
    }
    public abstract void bicim();
    public abstract void gramaj();
}
```

```
public class DChildOfAbstractEkmek extends CAbstractEkmek{
    public void un() {
        System.out.println("Cevizli ekmek kepekli undan yapilir");
    }
    public void maya() {
        System.out.println("Cevizli ekmekte binde 2 oraninda maya kullanilir");
    }
    public void bicim() {
        System.out.println("Cevizli akmek somun ekmek seklindedir");
    }
    public void gramaj() {
        System.out.println("Cevizli ekmek icin gramaj 650 gr.");
    }
    public static void main(String[] args) {
        DChildOfAbstractEkmek ekmek1= new DChildOfAbstractEkmek();
        ekmek1.bicim();
        ekmek1.gramaj();
        ekmek1.un();
        ekmek1.maya();
    }
}
```



# Abstract Classes

## Kural 6 :

Abstract bir child class, parent'i olan abstract class'lardaki method'lari implement etmek ZORUNDA DEGILDIR. Parent-child iliskisi ile tum method'lara ulasabilir.

```
public abstract class DParent {  
  
    public abstract void method1();  
  
    public abstract void method2();  
  
    public void method3(){  
        System.out.println("Parent class'daki method 3 calisti");  
    }  
}
```

## Kural 7 :

Abstract class'lardan sonra gelen ilk concrete class, parent abstract class(lar)'da concrete yapilmayan tum abstract method'lari implement etmelidir.

```
public abstract class GAbstractChild extends DParent{  
  
    public void method1(){  
  
    }  
    public abstract void method4();  
  
    public void method5(){  
  
    }  
}
```

```
public class HConcreteGrandChild extends GAbstractChild{  
  
    @Override  
    public void method2() {  
  
    }  
  
    @Override  
    public void method4() {  
  
    }  
}
```

# Abstract Classes

**Kural 8 :** Abstract class'lar class olduklari icin constructor'lari vardir ama **abstract** class'lardan obje olusturulamaz.

**NOT:** Abstract class'lar obje olusturmak icin degil, concrete child class'larinin sahip olmasi gereken ozellikleri tanimlamak icin olusturulur.

```
public abstract class GAraba {  
    public abstract void motor();  
  
    public abstract String marka();  
  
    public abstract void yakit();  
  
    public static void main(String[] args) {  
        GAraba arb= new GAraba();  
    }  
}
```

**Kural 9 :** final ve private method'lar override edilemeyecekleri icin, abstract method'lar final veya private olarak tanimlanamaz.



# Abstract Classes

Soru : Asagidaki kod sorunsuz calisir mi ?

```
abstract class Araba{
    public abstract String model();
}

abstract class Toyota extends Araba{
    public abstract void motor();
}

abstract class Corolla extends Toyota{
    public String model(){
        return "Corolla";
    }
}

public class DizelCorolla extends Corolla{

    public void motor() {
        System.out.println("Corolla Dizel araclar 1.6 dizel motor kullanir");
    }
}
```



# Abstract Classes

Soru : Asagidaki kodun calisabilmesi icin ne yapilmalidir ?

```
abstract class Araba{
    public abstract String model();
    abstract int tekerSayisi();
}

abstract class Toyota extends Araba{
    public abstract void motor();
    int tekerSayisi(){
        return 4;
    }
    abstract String vitesTuru();
}

abstract class Corolla extends Toyota{
    public String model(){
        return "Corolla";
    }
    abstract String kasa();
}

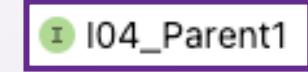
public class DizelCorolla extends Corolla{
    public void motor() {
        System.out.println("Corolla Dizel araclar 1.6 dizel motor kullanir");
    }
}
```

# Interfaces

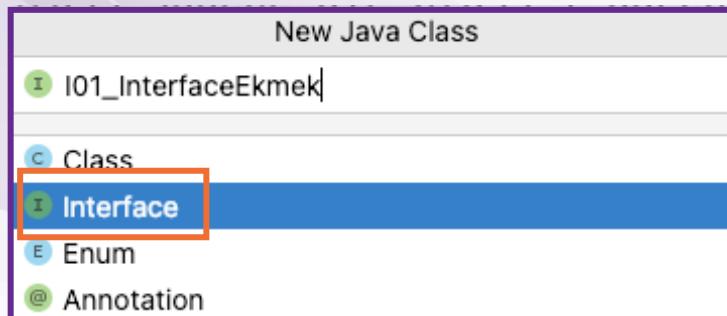
Interface icinde sadece abstract method'larin bulundugu bir yapidir.

Abstract class'lar abstract ve concrete method'lar bulundurabilir (partial abstraction), interface'ler ise sadece abstract method'lar bulundurur (full abstraction)

Interface yapı olarak da class ve abstract class'lardan farklıdır.



Interface'ler child class'ların uyarlamaya mecbur olduğu abstract method'ların bulunduğu bir sablon gibidir. Obje oluşturma amacıyla değil child class'ları biçimlendirme amacı tasır. Constructor'ları yoktur, interface'lerden obje oluşturulamaz.



Interface oluşturmak için IntelliJ'de  
New → Class → Interface seçilmelidir.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-39

### Interfaces



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# Interfaces

Interface özel yapısından dolayı içinde bulunan variable ve method'ların tamamı aynı yapıdadır.

1- Tüm variable'lar public, static ve final'dir. Bu keyword'ler yazilsa da yazilmasa da farketmez.

```
public interface I03_Interface {  
  
    public static final int sayi1=10;  
    public int sayi2=20; // public, static ve final'dir.  
    static final int sayi3=30; // public, static ve final'dir.  
}
```

Tüm variable'lar final olduğundan oluşturulurken değer atanması zorunludur, cunku final variable'larda sonradan değer degistirilemez.

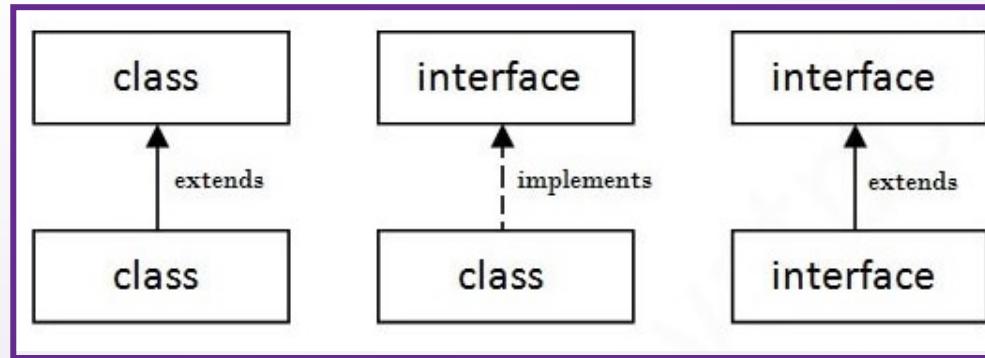
2- Tüm method'lar public ve abstract'dir. Bu keyword'ler yazilsa da yazilmasa da farketmez.

```
public interface I03_Interface {  
  
    public abstract void method1();  
    public void method2(); // public ve abstract  
    abstract String method3(); // public ve abstract  
    int method4(); // public ve abstract  
}
```



# Interfaces ve Inheritance

- 1- Bir class'dan bir class'i inherit etmek istendiginde **extends** keyword kullanilir. Class'dan Interface'i inherit etmek icin ise **implements** keyword kullanilir.



- 2- Interface'in en buyuk avantaji coklu inheritance'i desteklemesidir.

Bir class baska bir class'a extends ile child olup ayni zamanda implements keyword ile aralarinda virgul kullanarak birden fazla interface'i de implement edebilir.

```
public class NChild0fI06 extends KKepekliEkmek implements I06_Interface, I05_Parent2{
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-40

### Interfaces



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Onceki Dersten Aklimizda Kalanlar

- 1- Java'da bir proje olustururken, kendisinden obje olusturulmayacak, sadece child class'lari icin cati gorevi gorecek yapilara ihtiyacimiz olur.
- 2- Java'nin bu ihtiyac icin urettigi cozum abstraction'dir.
- 3- Java'da 2 cesit abstract yapı bulunur. A- Abstract Class B- Interface
- 4- aralarindaki fark Abstract Class'lar kismi abstraction saglar yani bunyelerinde abstract ve/veya concrete method'lar bulundurabilirler. Interface ise full abstraction saglar yani icinde concret method bulundurmamak üzere tasarlanmistir.
- 5- Full abstraction ozelliginden dolayi Interface'lerde bulunan tum method'lar public,abstract olurlar. Bu keyword'leri yazmasak da Java method'lari bu sekilde kabul eder.
- 6- Interface'lerde bulunan tum variable'lar da public, static ve final'dir.
- 7- Interface'lerin en buyuk avantaji inheritance'da ortaya cikar. Bir class sadece bir class'i inherit edebilirken, bir class istedigi kadar Interface'i kendisine implement edinebilir.
- 8- class – class veya interface – interface arasında inheritance icin extends keyword kullanilirken, class ile interface arasında implements keyword kullanilir.
- 9- bir class extends ile baska bir class'i parent edinip, ayni zamanda istedigi kadar interface'i aralarina , yazarak implement edebilir.

# Interfaces ve Inheritance

Interface bir c̄s̄it yapilacaklar listesi(to do list)dir.

Interface'de tamamen abstract method'lar oldugundan ve abstract method'larin body'si olmadigindan bir method'un nasil yapilacagini belirlemez.

Interface mutlaka implement edilecek method'lari belirler, concrete child class'lar o method'u kendilerine implement ederken nasil yapacaklarini method body'sinde tanimlarlar.

Boyut

hamur( )  
deepSos( )  
tabak( )

hamur( )  
deepSos( )  
tabak( )

Tür

malzeme( )  
sos( )  
peynir( )

malzeme( )  
sos( )  
peynir( )

malzeme( )  
sos( )  
peynir( )

Yan Malzemeler

icecek( )  
patates( )  
ekSos( )

icecek( )  
patates( )  
ekSos( )

Buyuk

Kucuk

4 Peynirli

Karisik

Kavurmali

Bol Malzeme

Standart





# Interfaces ve Inheritance

3- Birden fazla Interface implement edildiginde, bu interface'lerde ayni isimde variable ve method'lar bukunabilir. Java hangisini kullanacagini net bilmelidir.

```
public interface I04_Parent1 {  
  
    int sayi1=10;  
    int sayi2=20;  
  
    void method1();  
  
    int method2();  
}
```

```
public interface I05_Parent2 {  
  
    int sayi2=30;  
    int sayi1=50;  
  
    String method1();  
  
    int method2();  
}
```

```
public class MChildOf2Interfaces implements I04_Parent1,I05_Parent2{  
  
    public static void main(String[] args) {  
        System.out.println(I04_Parent1.sayi1); //10  
        System.out.println(I04_Parent1.sayi2);  
    }  
    public void method1() {}  
    public int method2() {  
        return 0;  
    }  
}
```

- Implement ettigimiz Interface'lerde ayni isimde variable varsa hangisinin kullanilacagini belirtmek icin static yontem yani InterfaceAdi.variableAdi yazilir.



# Interfaces ve Inheritance

- Implement ettigimiz Interface'lerde ayni isimde method varsa return type'ina bakilmalidir.

```
public interface I04_Parent1 {  
  
    int sayi1=10;  
    int sayi2=20;  
  
    void method1();  
  
    int method2();  
}
```

```
public class MChildOf2Interfaces implements I04_Parent1,I05_Parent2{  
  
    public static void main(String[] args) {  
        System.out.println(I04_Parent1.sayi1); //10  
        System.out.println(I04_Parent1.sayi2);  
    }  
    public void method1() {}  
    public int method2() {  
        return 0;  
    }  
}
```

```
public interface I05_Parent2 {  
  
    int sayi2=30;  
    int sayi1=50;  
  
    String method1();  
  
    int method2();  
}
```

- \* Return type'lari ayni ise, ikisinin de body'si olmadigindan hangisinin override edildiginin bir onemi yoktur,
- \* Return type'lari farkli ise, override eden method'un return type'i implement ettiği iki method'un return type'i ile overriding kurallari çerçevesinde uyumlu olmalıdır, aksi takdirde java CTE verir, kod calismaz.



# Abstract Classes vs Interfaces

## Abstract classes

- 1) Class'dir
- 2) abstract ve concrete method'lar konabilir
- 3) Kismi olarak abstraction saglar.
- 4) Birden fazla abstract class'a direk child olunamaz. Coklu inheritance'i desteklemez.
- 5) Hiz acisindan avantajlidir
- 6) Icindeki tüm nesnelerin "public" olması zorunlu degildir
- 7) soyut olmayan metodlar static olarak tanimlanabilir.
- 8) Abstract class constructor'a sahiptir

## Interface

- 1) Class degildir.
- 2) sadece abstract method'lar konabilir.
- 3) Tam abstraction (soyutluk) saglar
- 4) Bir class'dan istediginiz kadar interface'i inherit edebilirsiniz. Coklu inheritance'a uygundur.
- 5) Hiz acisindan abstract class'a gore yavastir.
- 6) Icindeki tüm nesnelerin "public" olması gereklidir
- 7) metodlar static olamaz
- 8) Interface constructor'a sahip degildir



# Interfaces'de Body'si Olan Method Olur mu?

Wise Quarter  
first class IT courses

Interface'de sadece public ve abstract method'lar bulunur.

Java8 ile bu kurala bir istisna getirilmistir.

Bu istisna'nin getirilme sebebi developer'ların ihtiyac halinde sonradan interface'e ekledikleri bir method'u, o interface'i implement eden tüm eski class'lara implement etmek istememeleridir.

Bu istisna sayesinde basına default veya static keyword eklenen method'ların bodysi olur ve bu method'lari child class'lardan implement etme zorunluluğu yoktur.

```
public interface I06_Interface {  
  
    void method1(); // bu method public ve abstractir  
  
    public default void method2(){  
        System.out.println("interface default method");  
    }  
  
    public static void method3(){  
        System.out.println("interface static method");  
    }  
}
```

Burada kullanılan default keyword access modifier degildir.

Default access modifier gorunur sekilde yazılırsa CTE olur. Ayrıca yandaki interface'de görüleceği gibi default olarak tanımlanan method'un basında access modifier olarak public vardır.

Bu kullanım da default keyword'un farklı kullanımlarındandır.



# Interfaces'de Body'si Olan Method Olur mu?

Wise Quarter  
first class IT courses

Interface'de istisna olarak kullanılan static ve default keyword'lerine sahip method'ların tek farklı child class'dan bu method'lara erişim yontemidir.

```
public interface I06_Interface {  
  
    void method1(); // bu method public ve abstractir  
  
    public default void method2(){  
        System.out.println("interface default method")  
    }  
  
    public static void method3(){  
        System.out.println("interface static method");  
    }  
}
```

```
public class NChildOfI06 implements I06_Interface{  
  
    public void method1() {  
  
    }  
  
    public static void main(String[] args) {  
  
        I06_Interface.method3();  
  
        NChildOfI06 obj = new NChildOfI06();  
        obj.method2();  
    }  
}
```

Static keyword ile oluşturulan istisnai method'lara static yolla ulaşılabilir.

Default keyword ile oluşturulan istisnai method'a ise obje oluşturularak ulaşılabilir.

# Tekrar Soruları

- 1) Abstract class kullanılarak obje olusturamayız. ~~True/False~~
- 2) Interface kullanılarak obje olusturabiliriz. ~~True/False~~
- 3) `public abstract int sumOfTwo(int n1, int n2) { }` syntax olarak doğru mudur ? ~~True/False~~
- 4) `public class Animal { public abstract void sound() }` syntax olarak doğru mudur ?  
~~True/False~~
- 5) `public abstract class Animal {  
 public abstract void eat();  
 public void sound( ){ } }` syntax olarak doğru mudur ? ~~True/False~~
- 6) Abstract class final olarak tanımlanamaz. ~~True/False~~
- 7) `public abstract class Animal {  
 private abstract void sound(); }` syntax olarak doğru mudur ? ~~True/False~~
- 8) Java birden fazla class'a extends ile inherit etmenize izin vermez. ~~True/False~~
- 9) Java birden fazla interface'e implements ile inherit etmenize izin verir. ~~True/False~~
- 10) Bir interface abstract veya concrete method'lara sahip olabilir. ~~True/False~~



# Interfaces

## Soru :

Which of the following statements can be inserted in the blank line so that the code will compile successfully? (Choose all that apply)

```
public interface CanHop {}  
public class Frog implements CanHop {  
    public static void main(String[] args) {  
        _____ frog = new TurtleFrog();  
    }  
}  
public class BrazilianHornedFrog extends Frog {}  
public class TurtleFrog extends Frog {}
```

- A. Frog
- B. TurtleFrog
- C. BrazilianHornedFrog
- D. CanHop
- E. Object
- F. Long

A, B, D, E. The blank can be filled with any class or interface that is a supertype of TurtleFrog. Option A is a superclass of TurtleFrog, and option B is the same class, so both are correct. BrazilianHornedFrog is not a superclass of TurtleFrog, so option C is incorrect. TurtleFrog inherits the CanHope interface, so option D is correct. All classes inherit Object, so option E is correct. Finally, Long is an unrelated class that is not a superclass of TurtleFrog, and is therefore incorrect.



# Interfaces

## Soru :

Choose the correct statement about the following code:

```
1: interface HasExoskeleton {  
2:     abstract int getNumberOfSections();  
3: }  
4: abstract class Insect implements HasExoskeleton {  
5:     abstract int getNumberOfLegs();  
6: }  
7: public class Beetle extends Insect {  
8:     int getNumberOfLegs() { return 6; }  
9: }
```

- A. It compiles and runs without issue.
- B. The code will not compile because of line 2.
- C. The code will not compile because of line 4.
- D. The code will not compile because of line 7.
- E. It compiles but throws an exception at runtime.

D. The code fails to compile because `Beetle`, the first concrete subclass, doesn't implement `getNumberOfSections()`, which is inherited as an abstract method; therefore, option D is correct. Option B is incorrect because there is nothing wrong with this interface method definition. Option C is incorrect because an abstract class is not required to implement any abstract methods, including those inherited from an interface. Option E is incorrect because the code fails at compilation-time.

# Interfaces

## Soru :

Choose the correct statement about the following code:

```
1: public interface Herbivore {  
2:     int amount = 10;  
3:     public static void eatGrass();  
4:     public int chew() {  
5:         return 13;  
6:     }  
7: }
```

- A. It compiles and runs without issue.
- B. The code will not compile because of line 2.
- C. The code will not compile because of line 3.
- D. The code will not compile because of line 4.
- E. The code will not compile because of lines 2 and 3.
- F. The code will not compile because of lines 3 and 4.

F. The interface variable amount is correctly declared, with public and static being assumed and automatically inserted by the compiler, so option B is incorrect. The method declaration for eatGrass() on line 3 is incorrect because the method has been marked as static but no method body has been provided. The method declaration for chew() on line 4 is also incorrect, since an interface method that provides a body must be marked as default or static explicitly. Therefore, option F is the correct answer since this code contains two compile-time errors.



# Interfaces

## Soru :

Choose the correct statement about the following code:

```
1: public interface CanFly {  
2:     void fly();  
3: }  
4: interface HasWings {  
5:     public abstract Object getWindSpan();  
6: }  
7: abstract class Falcon implements CanFly, HasWings {  
8: }
```

- A. It compiles without issue.
- B. The code will not compile because of line 2.
- C. The code will not compile because of line 4.
- D. The code will not compile because of line 5.
- E. The code will not compile because of lines 2 and 5.
- F. The code will not compile because the class Falcon doesn't implement the interface methods.

A. Although the definition of methods on lines 2 and 5 vary, both will be converted to `public abstract` by the compiler. Line 4 is fine, because an interface can have `public` or `default` access. Finally, the class `Falcon` doesn't need to implement the interface methods because it is marked as `abstract`. Therefore, the code will compile without issue.

# Interfaces

## Soru :

Which statements are true for both abstract classes and interfaces? (Choose all that apply)

- A. All methods within them are assumed to be abstract.
- B. Both can contain public static final variables.
- C. Both can be extended using the extend keyword.
- D. Both can contain default methods.
- E. Both can contain static methods.
- F. Neither can be instantiated directly.

B, C, E, F. Option A is wrong, because an abstract class may contain concrete methods. Since Java 8, interfaces may also contain concrete methods in form of static or default methods. Although all variables in interfaces are assumed to be public static final, abstract classes may contain them as well, so option B is correct. Both abstract classes and interfaces can be extended with the extends keyword, so option C is correct. Only interfaces can contain default methods, so option D is incorrect. Both abstract classes and interfaces can contain static methods, so option E is correct. Both structures require a concrete subclass to be instantiated, so option F is correct.



# Interfaces

Soru :

Which statements are true about the following code? (Choose all that apply)

```
1: interface HasVocalCords {  
2:     public abstract void makeSound();  
3: }  
4: public interface CanBark extends HasVocalCords {  
5:     public void bark();  
6: }
```

- A. The CanBark interface doesn't compile.
- B. A class that implements HasVocalCords must override the makeSound() method.
- C. A class that implements CanBark inherits both the makeSound() and bark() methods.
- D. A class that implements CanBark only inherits the bark() method.
- E. An interface cannot extend another interface.

C. The code compiles without issue, so option A is wrong. Option B is incorrect, since an abstract class could implement HasVocalCords without the need to override the makeSound() method. Option C is correct; any class that implements CanBark automatically inherits its methods, as well as any inherited methods defined in the parent interface. Because option C is correct, it follows that option D is incorrect. Finally, an interface can extend multiple interfaces, so option E is incorrect.

# Exceptions



Kod normal calisiyor.

Exception olusmaz



Ongorulen bir hata olustu.

Hata handle edilir,  
kod calismaya devam eder



Ongorulmeyen bir hata olustu.

Exception olusur,  
kod calismasi durur

Java kodlari calistirirken sorunlarla karsilasabilir.

- Sorunla karsilastiginda oncelikle bu sorun kodda ongorulmus ve bir cozum uretilmis ise istedigimiz hata kodlarini calistirir ve sonra siradaki kodlari calistirmaya devam eder.
- Sorunla karsilastiginda cozum uretilmemis ise kodu calistirmayi durdurur (**stops execution**) ve exception verir(**throws exception**)

# Exceptions

Yandaki kod kullanidan alınan iki tamsayıyı birbirine bolup, sonucu yazdırılmak için hazırlanmıştır.

Ancak kullanıcı bolecek sayıyı 0 girdiginde **sayı / 0** tanimsız olduğundan Java bu işlemi yapamaz. Kodun çalışmasını o satır itibarıyle durdurur ve exception bilgilerini konsolda yazdırır.

Konsolda yazılanlardan

- 1- Exception turunu
- 2- Exception'in sebebini
- 3- Exception'in olustugu satiri
- 4- Kodumuzun normal olmayan bir şekilde sonlandırıldığı bilgilerine ulaşabiliriz.

```
5 ► public class C01_Exceptions {  
6  
7 ►   public static void main(String[] args) {  
8  
9  
10    Scanner scan= new Scanner(System.in);  
11    System.out.println("Lutfen bolunecek tamsayiyi girin");  
12    int a= scan.nextInt();  
13    System.out.println("Lutfen bolecek tamsayiyi girin");  
14    int b= scan.nextInt();  
15  
16    System.out.println("Bolme isleminin sonucu : "+ a/b);  
17    System.out.println("Iyi gunler");  
18 }
```

```
Lutfen bolunecek tamsayiyi girin  
20  
Lutfen bolecek tamsayiyi girin  
0  
Exception in thread "main" java.lang.ArithmetricException Create breakpoint : / by zero  
at day37_exceptions.C01_Exceptions.main(C01_Exceptions.java:15)
```

```
Process finished with exit code 1
```



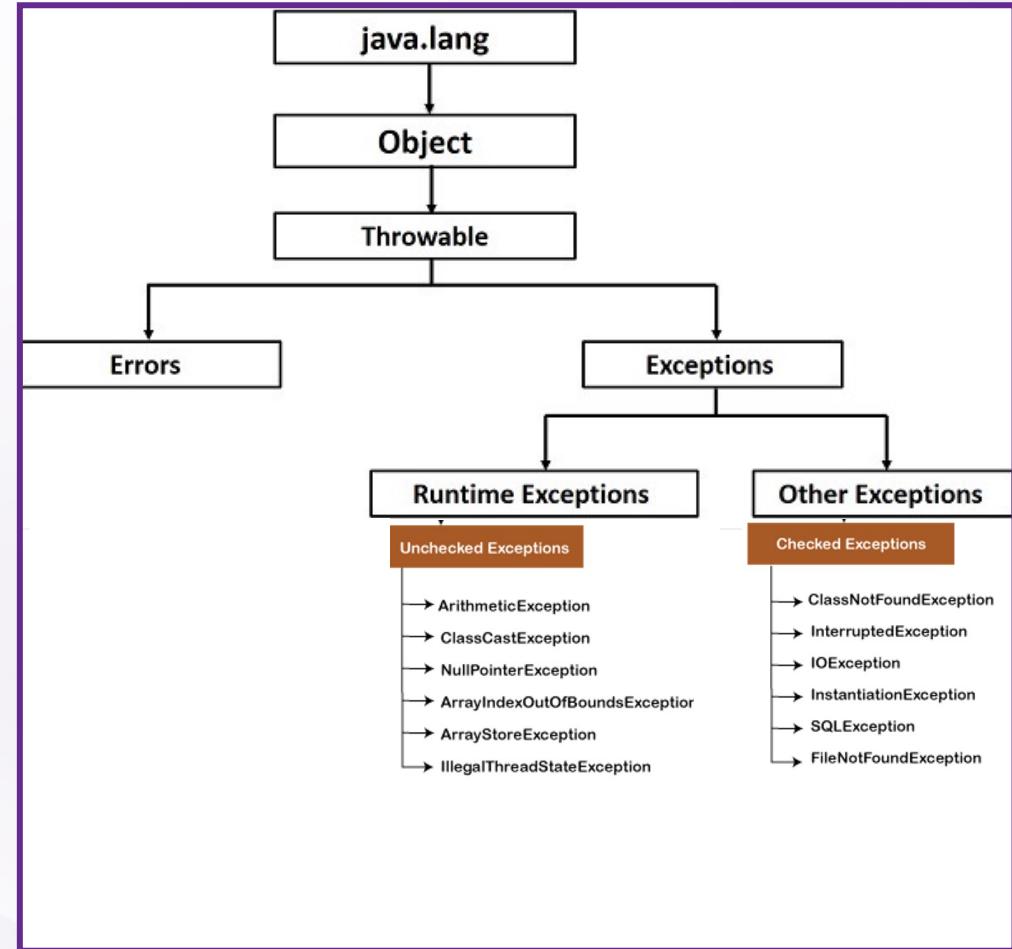
# Exceptions

Sayı / O problemini if-else ile cozebiliriz ancak if-else ile cozemeyecegimiz çok farklı exception'lar vardır.

Java bu sorunları handle edebilmemiz için Object class'ı altında Class'lar oluşturarak bizim için method'lar hazırlamıştır.

## Temel olarak Exception'lar

- **Checked Exceptions** : kodumuzu yazarken ortaya çıkan exception'lar
- **Unchecked Exceptions** : kodumuzu çalıştırınca ortaya çıkan exception'lar olarak ikiye ayrılır.



# Exceptions

## try/ catch (/ finaly) bloklari

Yazdigimiz kodda exception olusma ihtimali ongordugumuzde o kod blogunu try- catch ile sarmalariz

try/ catch blogunda 3 bolum vardir

1- try blogu : yapmak istedigimiz, sorun cikarma potansiyeline sahip code parcacigi

2- catch keyword ve ( ) bolumu:

- catch keyword ve ( ) sabittir
- parantez icine yazilacak exception turu kodumuza gore degisecektir
- e yakalanan exception'i koyacagimiz obje'dir

3- catch { } blogu : exception yakalandiginda calisacak kodlar

try catch blogu ile hem sorunu kullaniciya bildirip hem de kodun calismaya devam etmesini saglayabiliriz.

Buna handle exception denir, boylece kodumuz calismasina devam etmis olur

```
public class C02_TryCatch {  
    public static void main(String[] args) {  
        Scanner scan= new Scanner(System.in);  
        System.out.println("Lutfen bolunecek tamsayiyi girin");  
        int a= scan.nextInt();  
        System.out.println("Lutfen bolecek tamsayiyi girin");  
        int b= scan.nextInt();  
  
        try {  
            System.out.println("Bolme isleminin sonu : "+ a/b);  
        } catch (ArithmaticException e) {  
            e.printStackTrace();  
        }  
        System.out.println("Iyi gunler");  
    }  
}
```

# Exceptions

## catch blogunda kullanilan e

Try-catch blogu, try blogundaki kodlari catch blogunda yazilan exception olusuncaya kadar calistirir.

Kod yazilan exception turu ile karsilastigi satirda normal calismasini durdurur ve direkt catch bloguna gider.

catch blogundaki e yazdigimiz exception turunde bir exception yakalandiginda store edecegimiz variable ismidir

ismi e olmak zorunda degildir ancak genel kullanim e seklindedir

e objesi ile kullaniciya throwable class'inda olusturulan hazir exception messajlari yazdirilabilir veya biz e'yi kullanmadan istedigimiz exception messajini yazdirabiliriz

```
public static void main(String[] args) {
    Scanner scan;
    boolean tekrar=true;
    int girilensayi=0;

    while(tekrar){
        scan = new Scanner(System.in);
        System.out.println("Lutfen bir tamsayi giriniz");

        try {
            girilensayi= scan.nextInt();
            break;
        } catch (InputMismatchException e) {
            // System.out.println(e.getMessage()); // null
            // System.out.println(e.toString());
            // java.util.InputMismatchException
            // e.printStackTrace(); // uzun hata mesaji

            System.out.println("Adam gibi tamsayi degeri girsene");
        }
    }
    System.out.println("girilen sayinin karesi : " + girilensayi*girilensayi);
}
```



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-41

### Exceptions



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

@ /wisequarter



# Onceki Dersten Aklimizda Kalanlar

- 1- Exceptions (Istisna): Kodlarimiz calisirken gerçeklesen istisnai durumlardir.
- 2- Java kodlari calistirirken, beklenmedik(handle edemedigi) bir durumla karsilasirsa calismayı durdurur (**stops execution**) ve hatanin kaynagini, hatanin cinsini ve hatanin gerceklestigi satiri iceren bir mesaj verir. (**throws exception**)
- 3- Kodlarimizi yazarken exception olusabilecek ihtimalleri ongorebiliyorsak, java'ya ongordugumuz exception durumlarinda ne yapmasini istedigimizi **try-catch/(finally)** bloklari ile söyleyebiliriz.
- 4- eger try-catch ile sarmalanmis (**surround**) bir kod parçası calisirken, exception olusursa, olusan exception'i catch blogu ile handle edip etmediginizi kontrol eder, ongordugunuz exception(lar) gerçeklesen exception'i kapsiyorsa, bu durumda java catch blogundaki kodlari calistirir, exception firlatmadan kodlari calistirmaya devam eder.
- 5- Temel olarak exception'lar ikiye ayrılır
  - Run Time Exceptions (UnChecked) : kodlar calismaya basladiktan sonra ortaya çıkan exception'lardir.
  - Compile Time Exceptions(Checked) : Kodlarimizi yazar yazmaz java'nin farkina vardigi exception'lardir. Genellikle IO(dosya okuma ve yazma) ile ilgili exception'lardir.

# Exceptions

## Checked Exceptions ve throws keyword :

Java'da kodlarimizi yazarken daha onceden olusturulmus class'lardan yardim alabiliriz.

```
public class C04_IO {  
    public static void main(String[] args) {  
  
        String path="src/day37_exceptions/TextFile";  
        FileInputStream fis = new FileInputStream(path);  
  
    }  
}
```

Kullanicagimiz class, yapacagi islemler acisindan bazi exception ihtimallerini barindiriyorsa, olusabilecek bu exception'lar icin bastan tedbir almanizi isteyebilir.

Ornegin Java ile bilgisayaramizda var olan bir dosyaya erismek, ondaki bilgileri okumak veya yeni bilgiler eklemek istedigimizde Java'dan File class'ini veya farkli class'lari kullanabiliriz.

Bu class'larin herbiri yaptigi islere gore dosyaya ulasamama, dosyayı okuyamama veya dosyaya yazdiramama gibi exception ihtimallerini barindiracagindan bu class'lari kullanir kullanmaz Java sizden ne yapmasi gerektigini soracaktir.

Checked Exception'lar Java'nin kodlari calistirmasina izin vermeyeceginden bu class'lari kullanir kullanmaz Java'ya exception durumunda ne yapacagini soylemeniz gerekir.

# Exceptions

Checked Exceptions ve throws keyword :

Compile Time Exceptions icin iki secenegimiz vardir.

```
public class C04_IO {  
    public static void main(String[] args) {  
  
        String path="src/day37_exceptions/TextFile";  
        FileInputStream fis = new FileInputStream(path);  
  
    }  
}
```

1- Exception'i handle etmek ve exception olussa bile kodumuzun calismaya devam etmesini istiyorsak, try-catch blogu kullanabiliriz.

2- Exception olustugunda kod calismaya devam etmesin, exception verip kodun calismasini durdursun istiyorsak method signature'ina throws keyword'u ile ongorulen exception ismini yasmak yeterli olur.

throws keyword'u kullandigimizda exception ihtimalini ongordugumuzu ama handle etmek istemedigimizi hem Java'ya hem de bizden sonra kodlari kullanacak kisilere deklare etmis oluruz.

```
public class C04_IO {  
    public static void main(String[] args) throws FileNotFoundException {  
  
        String path="src/day37_exceptions/TextFile";  
        FileInputStream fis = new FileInputStream(path);  
  
    }  
}
```

# Multiple Exceptions

Yazdigimiz kod icin birden fazla exception olusma ihtimali varsa, bu exceprtion'lari handle etmeden once aralarinda parent-child iliskisi var mi diye bakmamiz gerekir.

Ornegin yandaki kod incelendiginde, String ve Array'den element yazdirirken, index'in sinir disinda olma riskini barindirir.

String'de girilen index sinir disinda ise `StringIndexOutOfBoundsException` olusur, Array'de ise `ArrayIndexOutOfBoundsException` olusacaktir.

```
public static void main(String[] args) {  
  
    String str = "Java";  
  
    int[] arr = {3, 4, 6, 8, 9};  
  
    // 0'dan 10'a kadar rastgele bir sayı üretip  
    // str ve arr'deki o index'e sahip elementleri yazdırıyalım  
  
    Random rnd = new Random();  
    int rastgeleSayi = rnd.nextInt(bound: 10);  
  
    System.out.println(rastgeleSayi);  
    System.out.println(str.substring(rastgeleSayi, rastgeleSayi + 1));  
    System.out.println("Index String'in sınırları dışında");  
}
```

Bu iki exception birbiri ile parent-child iliskisine sahip olmadiklarindan bu kodlari handle etmek icin daha fazla secenegimiz olacaktir.

# Multiple Exceptions

- 1- Risk olusturan kodlari ayri ayri try-catch bloklari ile sarmalayabiliriz
- 2- Iki islemi tek bir try-catch bloguna koyup yakalanacak exception olarak ongordugumuz iki exception'i da icine alacak **RunTimeException** veya **Exception** gibi bir exception class'i kullanabiliriz.

Bu durumda yakalanan exception genel oldugundan exception durumunda yazdiracagimiz bilgi iki exception'i da kapsayan genel bir ifade olabilir.

- 3- Tek try, multiple catch blogu kullanabiliriz.

Bu durumda, try blogunda yakalanan ilk exception'dan sonraki satirlarin calismayacagini goz onunde bulundurmamiz gerekir.

```
public static void main(String[] args) {  
  
    String str= "Java";  
  
    int[] arr = {3,4,6,8,9};  
  
    // 0'dan 10'a kadar rastgele bir sayı üretip  
    // str ve arr'deki o index'e sahip elementleri yazdırıyalım  
  
    Random rnd= new Random();  
    int rastgeleSayi= rnd.nextInt( bound: 10);  
  
    System.out.println(rastgeleSayi);  
  
    try {  
        System.out.println(str.substring(rastgeleSayi,rastgeleSayi+1));  
        System.out.println(arr[rastgeleSayi]);  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println("index Array'in sınırları dışında ");  
    } catch (StringIndexOutOfBoundsException e) {  
        System.out.println("index String'in sınırları dışında ");  
    }  
}
```

# Multiple Exceptions

Olusabilecek exception'lar arasında parent-child ilişkisi olursa, parent exception'in daha fazla yakalama kapasitesi, child exception'in ise spesifik olduğu için daha az yakalama kapasitesi olduğunu unutmamalıyız

Bu durumda ister throws keyword kullanarak method signature'ına ekleyelim, istersek de birden fazla catch blogu kullanalım, oncelik child exception'da olmalıdır.

Child exception, parent exception'dan sonra kullanılırsa, parent exception tüm exception'lari yakalayacaktır, child exception'a yakalayabilecegi bir exception kalmaz.



Parent exception, child exception'lari da yakalayacağı için child exception'i hiç yazmayabiliriz. Ancak ikisini de yasmak istiyorsak önce child exception, sonra parent exception yazılmalıdır.

NOT : Birden fazla catch blogu kullanıp child exception'i sonra yazarsak Java alttaki exception ulaşılabilir durumda değil diyerek CTE verir ve kodu calistirmaz.

# Multiple Exceptions

Ozellikle IO (File input ve output) ile ilgili kodlar yazdigimizda çok fazla ilintili exception oldugundan, parent-child hiyerarsisine dikkat edilmelidir.

veya tek exception yazip, tur olarak IOException veya Exception gibi tüm IO exception'larini kapsayan exception'lar secilmelidir.

Ornegin yandaki kod'da child olan FileNotFoundException once yazilmis olmasina ragmen IntelliJ onu gri (gereksiz) olarak isaretlemistir. IOException tumunu kapsadigi icin FileNotFoundException yazmasak da kod o exception'lari yakalayacaktir.

java.io  
**Class IOException**

java.lang.Object  
java.lang.Throwable  
java.lang.Exception  
java.io.IOException

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ChangedCharSetException, CharacterCodingException, CharConversionException, ClosedChannelException, EOFException, FileLockInterruptedException, FileNotFoundException, FilerException, FileSystemException, HttpRetryException, IOException, InterruptedByTimeoutException, InterruptedIOException, InvalidPropertiesFormatException, JMXProviderException, JMXServerErrorException, MalformedURLException, ObjectStreamException, ProtocolException, RemoteException, SaslException, SocketException, SSLException, SyncFailedException, UnknownHostException, UnknownServiceException, UnsupportedDataTypeException, UnsupportedEncodingException, UserPrincipalNotFoundException, UTFDataFormatException, ZipException

```
public static void main(String[] args) throws FileNotFoundException, IOException {  
  
    String path= "src/day37_exceptions/TextFile";  
  
    FileInputStream fis= new FileInputStream(path);  
  
    int k=0;  
  
    while ((k= fis.read()) != -1){  
        System.out.print((char)k);  
    }  
}
```

# Bazi Exception Turleri

## 1- NullPointerException

null olarak işaretlenmiş bir String'i string method'ları ile kullanmak istediğinizde ortaya çıkar.

## 2- StringIndexOutOfBoundsException

String'de olmayan bir index'i kullanmak istediğinizde ortaya çıkar.

## 3- ArrayIndexOutOfBoundsException

Array'de olmayan bir index'i kullanmak istediğinizde ortaya çıkar.

```
public static void main(String[] args) {  
  
    // 1- NullPointerException  
    String str=null;  
  
    System.out.println(str); // null  
  
    // System.out.println(str.concat("Ali")); // NullPointerException  
  
    // 2- StringIndexOutOfBoundsException  
  
    String str2="Java";  
    // System.out.println(str2.charAt(6)); // StringIndexOutOfBoundsException  
  
    // 3- ArrayIndexOutOfBoundsException  
  
    int[] arr={3,5,7,9};  
    System.out.println(arr[8]); // ArrayIndexOutOfBoundsException  
}
```

# Bazi Exception Turleri

## 4- NumberFormatException

```
public static void main(String[] args) {  
  
    // 4- NumberFormatException  
    //     parseInt () kullandigimizda string'de sayı disinda bir karakter olursa 56a vb..  
    String str= "567";  
  
    System.out.println(Integer.parseInt(str) +3); // NumberFormatException
```

```
// 5 - ClassCastException  uygun olmayan cast islemlerinde ortaya cikar  
  
String a= "Java";  
  
// Integer b= a;  farkli data turlerindeki variable'lari birbirine atayamayiz  
  
Object b = a; // Object class'i String'in parent class'i oldugundan kabul eder  
  
Integer c= (Integer)b;  
// object class'i Integer'in da parent'i oldugundan casting ile kabul etti  
  
System.out.println(c);  
}
```

## 5- ClassCastException

Uygun olmayan bir cast islemi yapmaya calistigimizda ortaya cikar

# throw Keyword

Java'da istedigimiz durumlarda biz de exception throw edebiliriz.

Kontrollu exception kullanimi kod'un belli islemler icin baska satirlara gecmesi, belli method'lari kullanmasi gibi amaclarla kullanilabilir.

```
public class C07_IllegalArgumentException {
    public static void main(String[] args) {

        Scanner scan= new Scanner(System.in);
        System.out.println("Lutfen yasinizi giriniz");

        int sayi=scan.nextInt();

        if (sayi<0){
            throw new IllegalArgumentException("Yas negatif olamaz");
        }
    }
}
```

```
Lutfen yasinizi giriniz
-10
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : Yas negatif olamaz
at day38_exceptions.C07_IllegalArgumentException.main(C07_IllegalArgumentException.java:14)

Process finished with exit code 1
```

# Exceptions

Soru : Asagidaki kod calistirilirsa, konsolda ne yazdirir ?

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

    String a=null;
    String b="";

    try{
        int c= a.length()+b.length();
        throw new RuntimeException();

    }catch (NullPointerException e){
        System.out.println("null pointer'dir, uzunlugu olmaz");

    }catch (RuntimeException e){
        System.out.println("throw keyword");
    }
}
```

# throw Keyword

Soru : Asagidaki kod calistirilirsa, konsolda ne yazdirir ?

```
public static void main(String[] args) {  
  
    String a=null;  
    String b="";  
  
    try{  
        int c= a.length()+b.length();  
        throw new RuntimeException();  
  
    }catch (NullPointerException e){  
        System.out.println("null pointer'dir, uzunlugu olmaz");  
  
    }catch (RuntimeException e){  
        System.out.println("throw keyword");  
    }  
}
```



# throw Keyword

Soru : Asagidaki kod calistirilirsa, konsolda ne yazdirir ?

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

    String a=null;
    String b="";

    try{
        try{
            int c= a.length()+b.length();

        }

    }catch (NullPointerException e) {
        if (b.length()==0){
            throw new RuntimeException();
        }
        System.out.println("null pointer'dir, uzunlugu olmaz");
    }
    }catch (RuntimeException e){
        System.out.println("throw keyword");
    }
}
```



# throw Keyword

Soru : Yandaki kod calistirilrsa,  
konsolda ne yazdirir ?

```
public static void main(String[] args) {  
  
    int a= 10;  
    int b= 2;  
  
    try{  
        System.out.println(a/b);  
    try{  
        Object c= a/b;  
        String d= (String)c;  
  
    }catch (ClassCastException e) {  
        throw new RuntimeException();  
    }  
    catch (RuntimeException e){  
        System.out.println("throw keyword");  
    }  
}
```

# throws Keyword

throws keyword'u bir class'da exception olusma olasılığının deklare etmek için kullanılır.

throws keyword kullanımı exception oluşturduğunda kodun durmasına engel olmaz, sadece Java'nın ve diğer kullanıcıların bu class'da exception riskinin farkında olduğumuzu bilmesine yarar.

throws keyword kullanıldığı method'un deklare edildiği satırda ( ) ile { } arasına yazılır.  
Method içinde kullanılamaz.

```
public static void main(String[] args) throws FileNotFoundException, IOException {  
  
    String path= "src/day37_exceptions/TextFile";  
  
    FileInputStream fis= new FileInputStream(path);  
  
    int k=0;  
  
    while ((k= fis.read()) != -1){  
        System.out.print((char)k);  
    }  
}
```

Birden fazla exception ihtimali varsa, throws keyword'den sonra aralarına virgül konarak tüm exception'lar yazılabilir

throws keyword ile birden fazla exception yazılacaksa ve aralarında parent-child ilişkisi varsa önce child, sonra parent exception yazılmalıdır.



# throw - throws

## throw

- 1- throw keyword kontrollu olarak bir exception throw etmek için kullanılır
- 2- throw keyword ile sadece bir exception throw edilebilir
- 3- throw keyword method içinde kullanılır
- 4- throw keyword yazdıktan sonra variable yazılır  
**Orn:** new  
IllegalArgumentException( );

-

## throws

- 1- throws keyword bir veya daha fazla exception'i deklare etmek için kullanılır
- 2- throws keyword bir veya daha fazla exception deklare edilebilir
- 3- throws keyword method'un deklare edildiği satırda kullanılır
- 4- throws keyword yazdıktan sonra exception class ismi yazılır  
**Orn:** FileNotFoundException



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-41

Exceptions

Errors

## Garbage Collector



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter

# finally keyword

Finally blogu try blogu ile calisir.

Bir try blogu ile istendigi kadar catch blogu ve en sonda bir tane finally blogu kullanilabilir.

Bir try blogu ile hic catch blogu kullanmadan da finally blogu kullanilabilir.

```
int a= 10;
int b= 0;

try {
    System.out.println(a/b);
} catch (ArithmaticException e) {
    System.out.println("bolecek sayı sıfır olamaz");
} finally {
    System.out.println("Finally blok çalıştı");
}
```

finally blogu her durumda calisir, kullanım amaci bir exception olussa da, mutlaka yapilmasi gereken database connection'inin durdurulmasi gibi gorevlerin yapilacagindan emin olmaktir.



# finally keyword

Soru : Yandaki kod calistirilirsa,  
konsolda ne yazdirir ?

```
public static void main(String[] args) {  
  
    String a= null;  
  
    try{  
        a.concat(str: "b");  
  
    }catch (NullPointerException e) {  
        a="";  
  
    }finally {  
  
        System.out.println(a.concat(str: "c"));  
    }  
}
```



# finally keyword

Soru : Yandaki kod calistirilirsa,  
konsolda ne yazdirir ?

```
public static void main(String[] args) {  
  
    String a= null;  
    String b="";  
  
    try {  
        try {  
            b = b + "a";  
            a.length();  
            b = b + "b";  
        } catch (NullPointerException e) {  
            b = b + "c";  
        } finally {  
            b = b + "d";  
            throw new RuntimeException();  
        }  
    }catch (Exception e) {  
        b=b+"e";  
    }  
    System.out.println(b);  
}
```

# Exceptions Tekrar Soruları

1) try blogu mutlaka catch blogu ile kullanilmalidir. ~~True / False~~

2) finally blogu mutlaka calisir. ~~True / False~~

3) Bir try blogu ile birden fazla catch blogu calistirilabilir. ~~True / False~~

4) Birden fazla catch blok varsa, child olan once yazilmalidir. ~~True / False~~

5) FileNotFoundException nedir?

Programimizda bir dosyayı okumaya çalışiyorken, dosya bulunamazsa olusur. IOException'in subclass'ıdır.

6) IOException nedir?

Programimizda bir file'a input/output yapiliyorsa ve program calisirken bir problem cikarsa olusur.

Checked exception'dir ve kod yazilirken mutlaka handle edilmelidir.



# Yeni Checked Exception Olusturma

- 1 – Checked Exception olusturmak istiyorsak Exception Class'ini parent class yapmaliyiz.
- 2 – Istedigimiz ismi verebiliriz ama genel kullanım ismin Exception kelimesi içerecek şekilde seçilmesidir.

```
public class InvalidEmailIdCheckedException extends Exception {  
  
    private static final long serialVersionUID = 1L;  
  
    public InvalidEmailIdCheckedException(String message) {  
        super(message);  
    }  
}
```

- 3 – Olusturdugumuz exception'a bir seri no vermeliyiz. Seri nosuna kolay erisim icin variable'i static yapip, degistirilmemesi icin final olarak tanimlayin.
- 4 – String paramertresi olan bir constructor olusturup ilk satirina super(message) ekleyin.



# Yeni Checked Exception Olusturma

Soru : Yeni bir class olusturalim, icinde mailDogrula(String eMail) olsun. Email adresi @gmail.com veya @hotmail.com icermiyorsa InvalidEmailIdCheckedException versin.

```
public class Test {  
  
    public static void main(String[] args) throws InvalidEmailIdCheckedException {  
        mailDogrula("ab@gmail1.com");  
    }  
  
    public static void mailDogrula(String eMail) throws InvalidEmailIdCheckedException {  
        if (eMail.contains("@hotmail.com") || eMail.contains("@gmail.com")) {  
            System.out.println(eMail);  
        } else {  
            throw new InvalidEmailIdCheckedException("Email adresi uygun degil");  
        }  
    }  
}
```



# Yeni Unchecked Exception Olusturma

- 1 – Checked Exception olusturmak istiyorsak RuntimeException Class'ini parent class yapmaliyiz.
- 2 – Istedigimiz ismi verebiliriz ama genel kullanım ismin Exception kelimesi içerecek şekilde seçilmesidir.

```
public class InvalidEmailIdUnCheckedException extends RuntimeException {  
  
    private static final long serialVersionUID = 1L;  
  
    public InvalidEmailIdUnCheckedException(String message) {  
        super(message);  
    }  
}
```

- 3 – Olusturdugumuz exception'a bir seri no vermeliyiz. Seri nosuna kolay erisim icin variable'i static yapip, degistirilmemesi icin final olarak tanimlayin.
- 4 – String parametresi olan bir constructor olusturup ilk satirina super(message) ekleyin.



# Yeni Unchecked Exception Olusturma

**Soru :** Yeni bir class olusturalim, icinde mailDogrula(String eMail) olsun. Email adresi @gmail.com veya @hotmail.com icermiyorsa InvalidEmailIdUnCheckedException versin.

```
public class Test {  
  
    public static void main(String[] args)  {  
        mailDogrula("ab@gmail1.com");  
    }  
  
    public static void mailDogrula(String eMail)  {  
        if (eMail.contains("@hotmail.com") || eMail.contains("@gmail.com")) {  
            System.out.println(eMail);  
        } else {  
            throw new InvalidEmailIdUnCheckedException("Email adresi uygun degil");  
        }  
    }  
}
```

# Exceptions

Soru : Bosluklara ne yazılmalıdır ? Tüm uyan sıkıları işaretleyin

Which of the following pairs fill in the blanks to make this code compile? (Choose all that apply)

```
7: public void ohNo() _____ Exception {  
8:     _____ Exception();  
9: }
```

- A. On line 7, fill in throw
- B. On line 7, fill in throws
- C. On line 8, fill in throw
- D. On line 8, fill in throw new
- E. On line 8, fill in throws
- F. On line 8, fill in throws new

Cevap : B, D. In a method declaration, the keyword throws is used. To actually throw an exception, the keyword throw is used and a new exception is created.

# Exceptions

Soru : Kod calistirilirsa hangi exception olusur ?

Which exception will the following throw?

```
Object obj = new Integer(3);  
String str = (String) obj;  
System.out.println(str);
```

- A. ArrayIndexOutOfBoundsException
- B. ClassCastException
- C. IllegalArgumentException
- D. NumberFormatException
- E. None of the above.

Cevap : B. The second line tries to cast an Integer to a String. Since String does not extend Integer, this is not allowed and a ClassCastException is thrown.



# Exceptions

## Soru :

Which of the following can be inserted in the blank to make the code compile? (Choose all that apply)

```
public static void main(String[] args) {  
    try {  
        System.out.println("work real hard");  
    } catch (_____ e) {  
    } catch (RuntimeException e) {  
    }  
}
```

- A. Exception
- B. IOException
- C. IllegalArgumentException
- D. RuntimeException

Cevap : C, E. Option C is allowed because it is a more specific type than RuntimeException. Option E is allowed because it isn't in the same inheritance tree as RuntimeException. It's not a good idea to catch either of these. Option B is not allowed because the method called inside the try block doesn't declare an IOException to be thrown. The compiler realizes that IOException would be an unreachable catch block. Option D is not allowed because the same exception can't be specified in two different catch blocks. Finally, option A is not allowed because it's more general than RuntimeException and would make that block unreachable.



# Exceptions

## Soru :

What is the output of the following snippet, assuming a and b are both 0?

```
3:     try {
4:         return a / b;
5:     } catch (RuntimeException e) {
6:         return -1;
7:     } catch (ArithmaticException e) {
8:         return 0;
9:     } finally {
10:        System.out.print("done");
11:    }
```

- A. -1
- B. 0
- C. done-1
- D. done0
- E. The code does not compile.
- F. An uncaught exception is thrown.

**Cevap :** E. The order of catch blocks is important because they're checked in the order they appear after the try block. Because ArithmaticException is a child class of RuntimeException, the catch block on line 7 is unreachable. (If an ArithmaticException is thrown in try try block, it will be caught on line 5.) Line 7 generates a compiler error because it is unreachable code.

# Exceptions

Soru :

Cevap :

E. The `parseName` method is invoked within `main()` on a new `Dog` object. Line 4 prints 1. The try block executes and 2 is printed. Line 7 throws a `NumberFormatException`, so line 8 doesn't execute. The exception is caught on line 9, and line 10 prints 4. Because the exception is handled, execution resumes normally. `parseName` runs to completion, and line 17 executes, printing 5. That's the end of the program, so the output is 1245.

What is the output of the following program?

```
1: public class Dog {  
2:     public String name;  
3:     public void parseName() {  
4:         System.out.print("1");  
5:         try {  
6:             System.out.print("2");  
7:             int x = Integer.parseInt(name);  
8:             System.out.print("3");  
9:         } catch (NumberFormatException e) {  
10:             System.out.print("4");  
11:         }  
12:     }  
13:     public static void main(String[] args) {  
14:         Dog leroy = new Dog();  
15:         leroy.name = "Leroy";  
16:         leroy.parseName();  
17:         System.out.print("5");  
18:     } }  
A. 12  
B. 1234  
C. 1235  
D. 124  
E. 1245  
F. The code does not compile.  
G. An uncaught exception is thrown.
```

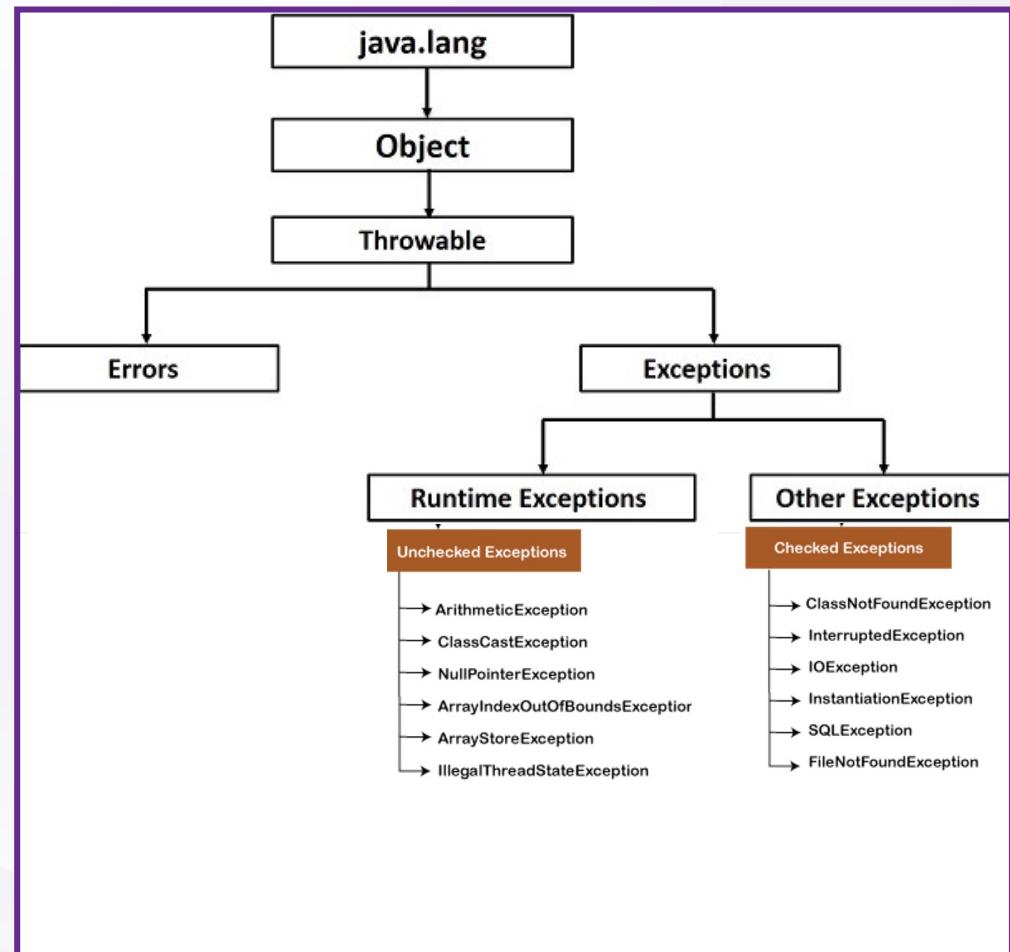
# Errors

Error class'i Throwable classinin child class'idir.

Error sistemsel sebeplerle ortaya cikan beklenmedik durumdur ve handle edilemez.

Error'lar onceden ongorulemezler, RunTime'da olusurlar yani Unchecked'dirler.

Error olustugunda program durdurulur, exception gibi handle edilemediginden data kayiplari olmasinin onune gecilemez.





# Errors

```
public static void main(String[] args) {  
  
    for (int i = 10; i >0 ; i++) {  
        System.out.print(i + " ");  
    }  
}
```

Error'lara örnek olarak **OutOfMemoryError**, **StackOverFlowError** veya **SystemCrashError** hatalarını verebiliriz.



# Errors vs Exceptions

## Karşılaştırma Tablosu

| Karşılaştırma için temel | Hata                                                                   | İstisna                                                                                                                                |
|--------------------------|------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Temel                    | Hata, sistem kaynaklarının eksikliğinden kaynaklanır.                  | Kod nedeniyle bir istisna ortaya çıkıyor.                                                                                              |
| Kurtarma                 | Bir hata düzeltilemez.                                                 | Bir istisna kurtarılabilir.                                                                                                            |
| Anahtar kelimeler        | Bir hatayı program kodıyla çözmenin bir yolu yoktur.                   | İstisnalar, "try", "catch" ve "throw" üç anahtar sözcüğü kullanılarak ele alınır.                                                      |
| sonuçlar                 | Hata tespit edildiğinde program anomal olormalı şekilde sonlandırılır. | Bir istisna tespit edildiğinde, karşılık gelen "atma" ve "yakalama" anahtar sözcükleri tarafından atılır ve yakalanır.                 |
| Türleri                  | Hatalar denetlenmeyen tür olarak sınıflandırıldı.                      | İstisnalar kontrol edilmiş veya kontrol edilmemiş tip olarak sınıflandırılır.                                                          |
| paket                    | Java'da hatalar "java.lang.Error" paketi olarak tanımlanmıştır.        | Java'da, bir istisna "java.lang.Exception" içinde tanımlanmıştır.                                                                      |
| Örnek                    | OutOfMemory, StackOverFlow.                                            | İşaretli İstisnalar: NoSuchMethod, ClassNotFoundException. İşaretlenmemiş İstisnalar: NullPointerException, IndexOutOfBoundsException. |

# Java - Platform Independent

Platform independent (platformdan bagimsiz) calisma, yazilan kodun her platformda calismasi demektir.

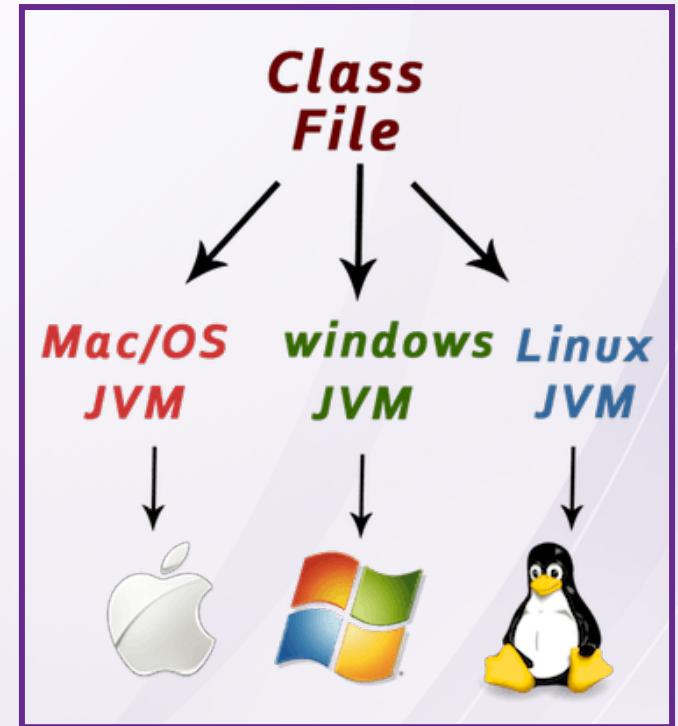
Hangi programlama dilini kullanırsak kullanalım, bizim için anlamlı for-loop, if-else, method, class.. gibi kelimeler kullanırız

Bu kelimelerin bilgisayar tarafından anlasılabilmesi için bir **compiler** tarafından makine diline(**byte code**) çevrilmesi gereklidir.

Java'da, **Javac** (**J**ava **p**rogramming **l**anguage **c**ompiler) yazdığımız kodları .class uzantılı byte kodlara çevirir.

Javac tarafından byte kodlara çevrilen bu kodlar **JVM** (**J**ava **V**irtual **M**achine) tarafından her türlü platformda çalıştırılır.

Boylece Java'da herhangi bir platformda yazılan kodlar tüm platformlarda çalıştırılabilir.





# Garbage Collector



Unused objects



Finalize Garbage



Destroy

Garbage Collection, Java'da daha once kullanılmış ama artık kullanılmayan objelerin temizlenerek, memory kullanımını düzenleyen sürecin adıdır.

Garbage Collection için Java'da oncelikle `finalize()` method'u çalışarak kullanılmayan objeleri işaretler. Sonra Garbage Collector devreye girer ve işaretlenmiş objeleri yokeder.

Java'da kullanılan memory miktarını gösteren bazı uygulamalar olsa da, garbage collection sürecini JVM yönetir. Biz istediğimiz zamanlarda `finalize()` ve Garbage Collector'un çalışmasını saglayabiliriz ancak bu çok kullanılmaz. JVM bu süreci sorunsuz bir şekilde yönetir.



# final – finally{ } – finalize( )



Java'da farklı amaclarla kullanılan bu 3 kavram genelde karıştırılır ve interview'lerde sorulabilir.

**final variable** : Degeri degistirilemeyecek variable

**final method** : Override edilemeyecek method

**final class** : Inherit edilemeyecek class

Garbage collection mekanizmasında görev alır. Garbage collector'den önce çalışarak kullanılmayan ve yok edilmesi gereken variable'lari işaretler. Garbage collector işaretlenmeyen objeleri toplamaz.

**final keyword**

**finalize ( )**

finally blogu try ile birlikte kullanılabilir.

Try blogu tek başına çalışmaz catch ve/veya finally blogu ile çalışmak zorundadır. finally blogu try-catch'e eklenirse, her ihtimalde çalışır. Sistem kapanmadan kritik önemdeki işlemleri yapmak için kullanılır.

**finally{ }**

# Iterator

Java'da coklu element iceren Array ve ArrayList gibi yapılarda, index kullanarak element'lere erisim ve update yapmak mumkundur. Ancak Java'da coklu element iceren tum yapilar index kullanmaz.

Iterator, index kullanma'dan collection elementleri üzerinde gezinmemizi ve elementler üzerinde degisiklik yapmamizi saglar.

## Method Summary

| All Methods       | Instance Methods | Abstract Methods                                                                                                                                                                                  | Default Methods |
|-------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Modifier and Type |                  | Method and Description                                                                                                                                                                            |                 |
| default void      |                  | <code>forEachRemaining(Consumer&lt;? super E&gt; action)</code><br>Performs the given action for each remaining element until all elements have been processed or the action throws an exception. |                 |
| boolean           |                  | <code>hasNext()</code><br>Returns true if the iteration has more elements.                                                                                                                        |                 |
| E                 |                  | <code>next()</code><br>Returns the next element in the iteration.                                                                                                                                 |                 |
| default void      |                  | <code>remove()</code><br>Removes from the underlying collection the last element returned by this iterator (optional operation).                                                                  |                 |

# List Iterator

Iterator sinirli sayida method'a sahiptir.

ListIterator interface'i Iterator interface'inin child'i olarak olusturulmustur ancak daha fazla method bulundurmaktadir.

## Method and Description

`add(E e)`

Inserts the specified element into the list (optional operation).

`hasNext()`

Returns true if this list iterator has more elements when traversing the list in the forward direction.

`hasPrevious()`

Returns true if this list iterator has more elements when traversing the list in the reverse direction.

`next()`

Returns the next element in the list and advances the cursor position.

`nextIndex()`

Returns the index of the element that would be returned by a subsequent call to `next()`.

`previous()`

Returns the previous element in the list and moves the cursor position backwards.

`previousIndex()`

Returns the index of the element that would be returned by a subsequent call to `previous()`.

`remove()`

Removes from the list the last element that was returned by `next()` or `previous()` (optional operation).

`set(E e)`

Replaces the last element returned by `next()` or `previous()` with the specified element (optional operation).

ListIterator ile collection elementleri üzerinde ileri veya geri hareket edebilir, elementleri update edebilir veya silme imkani tanir.



# List Iterator

```
public static void main(String[] args) {  
    List<Integer> sayilar= new ArrayList<>();  
  
    sayilar.add(10);  
    sayilar.add(20);  
    sayilar.add(30);  
    sayilar.add(40);  
  
    // Tum elementleri Iterator ile 5 artiralim  
    ListIterator lit = sayilar.listIterator();  
  
    while (lit.hasNext()){  
        lit.set((Integer)lit.next()+5);  
    }  
  
    System.out.println(sayilar);  
  
    // list elementlerini sondan basa yazdirin  
    // sondan basa yazdirmak icin once sona gitmeliyiz  
    // ancak ustteki loop'tan dolayi iterator zaten en sonda  
  
    while(lit.hasPrevious()){  
        System.out.print(lit.previous()+" ");  
    }  
}
```

Iterator veya ListIterator kullanirken dikkat etmemiz gereken en onemli konu, iterator'un nerede oldugu ve nasil hareket ettirilecegidir.

Ozellikle iterator.next( ) ve iterator.previous( ) method'larinin her kullanildiginda iterator'in hareket ettigini unutmamamiz gerekir.

Bir isleme baslamadan once iterator'un dogru konumda oldugundan ve uzerinde bir obje olup olmadiginden emin olmalıyız, aksi takdirde exception'larla karsilasabiliriz.

# Iterator

**Soru 1)** Bir listedeki istenen sayı aralığında olmayan elementleri silen bir program yazınız . (2. liste olusturmadan, gecerli liste üzerinde işlem yapınız)

Orn : [2,13,56,23,45,14,40] istenen aralık 20 ile 40 arası (sinirlar dahil)  
output: [23,40]

**Soru 2)** Bir listedeki elementleri iterator kullanarak sondan başa doğru yazdırın

**Soru 3)** (iterator ile index kullanımına örnek) Bir listedeki ilk n elemanı iterator kullanarak 5 artırın.

Orn :      list : [2,13,56,23,45,14,40]  
              artırılması istenen eleman sayısı : 3  
              output: [7,18,61,23,45,14,40]



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-42

### Collections



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

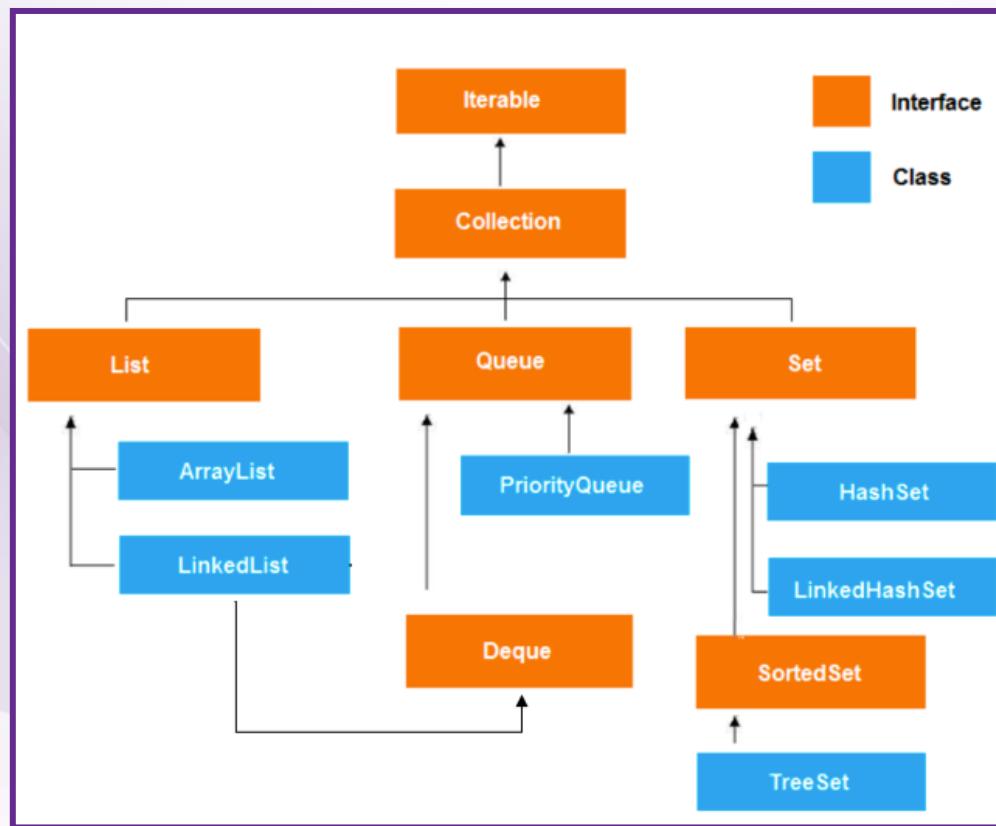
/wisequarter



# Collections

Java'da Collection Interface'i altında olusturulan, objelerden olusan bir topluluğu bir arada tutan yapılardır

Collections, gerçek hayatı ihtiyaçları karşılayabilmek için farklı özelliklere sahiptirler.



Tüm yapıyi ezberlemek yerine, bize collection'in ozelligini veren 5 kelimeye odaklanmak gereklidir.

1- Set (Kume)

2- Linked (Baglı)

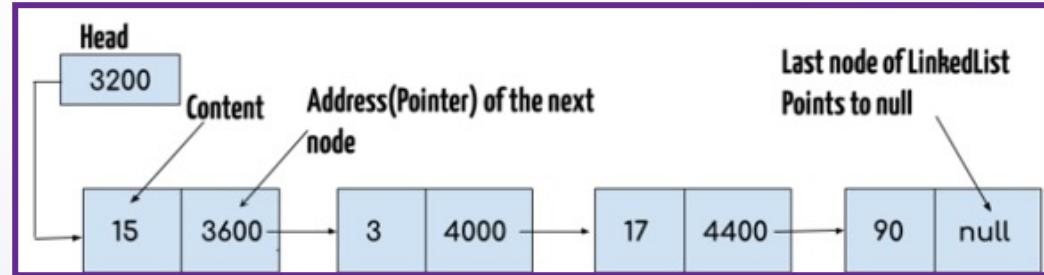
3- Queue (Sıra)

4- Tree (Doğal Sirali)

5- Hash

# Collections / Linked List

Linked List elementleri birbirine bagli bir zincir seklinde tutar

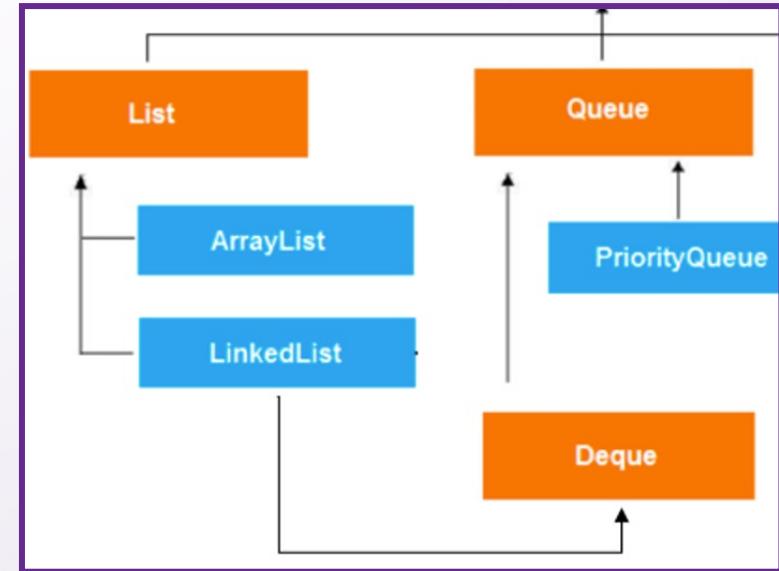


- Bir LinkedList olusturulduğunda ilk element olarak head olusturulur, head sadece 1.elementin adres bilgisini tutar.
- Sonraki elementler data ve kendinden sonraki elementin adresini tutan iki parcadan olusur
- Bu ikili yapinin herbirine **node** ismi verilir.
- Son **node'a** tail(kuyruk) denir ve adres bolumunde nullPointer bulunur. .
- Linked yapisindan dolayi element ekleme ve silme cok kolaydir ancak elementlere ulasmak yavastir.

# Collections / Linked List

Linked List 3 Interface'i implement etmistir.  
LinkedList constructor'i kullanarak  
olusturacagimiz objelerde data turu  
seçimimize gore kullanabilecegimiz method'lar  
da degisecektir.

```
LinkedList<Integer> ll1 = new LinkedList<>();
// List, Queue ve deque ozelliklerinin hepsini tasir
List<Integer> ll2 = new LinkedList<>();
Deque<Integer> ll3 = new LinkedList<>();
Queue<Integer> ll4 = new LinkedList<>();
```



Data turu secerken hangi collection'in ozelliklerini kullanmak isteniyorsa ona gore  
karar verilmelidir.

LinkedList class'i interface'lerin child'i olan bir concrete class oldugundan 3  
interface'deki method'larin tamamini override etmistir. Data turu olarak LinkedList  
secilirse 3 interface'in tum ozellikleri kullanilabilir.

# Collections / Linked List

Data turu List secilirse daha once kullandigimiz ArrayList ile benzer ozelliklere sahip olacaktir.

LinkedList yapisi geregi, olusturulurken secilecek data turune gore queue veya deque ozellikleri de gosterebilir. Onlar sonraki slaytlarda o konularin basligi altinda tekrar ele alınacaktır.

```
public static void main(String[] args) {  
  
    List<Integer> ll2 = new LinkedList<>();  
  
    ll2.  
        add(int index, Integer element)  
        retainAll(Collection<?> c)  
        add(Integer e)  
        remove(Object o)  
        subList(int fromIndex, int toIndex)  
    ll.remove(int index)  
    ll.hashCode()  
    Sy.set(int index, Integer element)  
    ll.get(int index)  
    Sy.addAll(int index, Collection<? extends Integer> c)  
    Sy.addAll(Collection<? extends Integer> c)  
    Sy.equals(Object o)  
    Sy.removeAll(Collection<?> c)  
    // clear()  
    // size()  
    In.contains(Object o)
```

# Collections / Linked List

list.hashCode( ) : olusturulan list'in o anki durumuna Java'nin tanımladığı hashCode'u döndürür.

Her eleman eklendiginde veya silindiginde değeri değişir.

```
public static void main(String[] args) {  
  
    List<Integer> ll2 = new LinkedList<>();  
    ll2.add(10);  
    ll2.add(20);  
  
    System.out.println(ll2); // [10, 20]  
    System.out.println(ll2.hashCode()); // 1291  
  
    ll2.add(40);  
    System.out.println(ll2.hashCode()); // 40061  
  
    ll2.remove(index: 1);  
    System.out.println(ll2.hashCode()); // 1311
```



# Collections / Linked List

list.**retainAll( list2 )** : list ile list2'yi  
karsilastirip, ortak olmayan elementleri  
list'den siler..

Silme islemi yaptiysa true, yapmadıysa  
false döner.

```
public static void main(String[] args) {  
  
    List<Integer> ll2 = new LinkedList<>();  
    ll2.add(10);  
    ll2.add(40);  
  
    System.out.println(ll2); // [10, 40]  
    List<Integer> ll5 = new LinkedList<>();  
    ll5.add(10);  
    ll5.add(20);  
    ll5.add(30);  
    System.out.println(ll5); // [10, 20, 30]  
  
    ll2.retainAll(ll5);  
  
    System.out.println(ll2); // [10]  
    System.out.println(ll5); // [10, 20, 30]  
  
    ll5.retainAll(ll2);  
    System.out.println(ll2); // [10]  
    System.out.println(ll5); // [10]  
}
```

# Collections / Linked List

**Soru :** Kullanicidan deger alarak iki String liste olusturun. Kullanici deger girmeyi birakmak icin O'a basmalidir.

Iki liste olustuktan sonra asagidaki sekilde bir output hazirlayin

liste1 : .....

liste2 : .....

ortak elementler : .....

**Soru :** Bir ogretmenden ogrencilerin notlarini girmesini isteyin, not grime islemi bittiginde Q'ya basmalidir. Not grime islemi bittikten sonra asagidaki sekilde output hazirlayin

not ortalamasi : .....

ogrenci sayisi : .....

ortalama altindaki ogrenci sayisi : ....

ortalamanin 10 puan alt ve ustunde olan ogrenci sayisi : ....



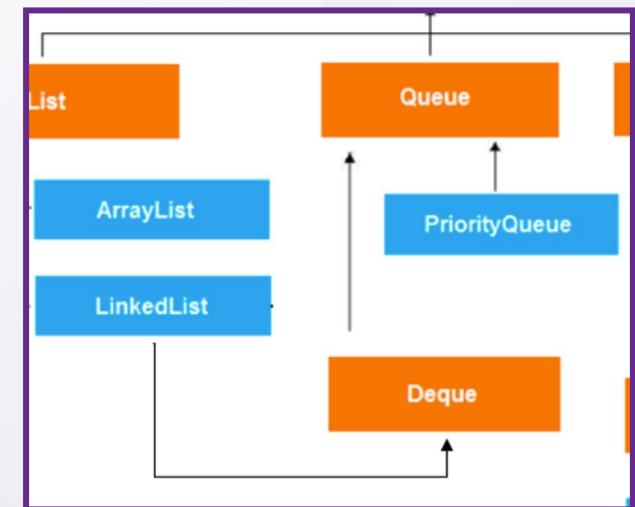
# Collections / Queue

Queue'nun temel ozelligi FIFO ( First In First Out ) kuralidir. Eczane ve yemekhane gibi tuketim suresi sinirli malzeme kullanan isletmeler icin onemli bir ozelliktir.

Queue Interface oldugu icin obje olusturmak mumkun degildir.

Queue ozelliklerine sahip bir obje olusturmak icin PriorityQueue veya LinkedList class'larindan constructor kullanilmalidir.

PriorityQueue Java'nin belirlemis oldugu bir oncelige gore siralama yapar, LinkedList ise ekleme sirasina gore bir list olusturur.





# Collections / Queue

queue.**element( )** : ilk elementi silmeden bize döndürür.

queue.**peek( )** : ilk elementi silmeden bize döndürür.

Aralarında sadece bir fark vardır. Bir queue için **peek( )** null sonucu donerken, **element ( )** exception oluşturur.

```
Queue<String> yemekSirasi= new LinkedList<>();  
  
yemekSirasi.add("ali");  
yemekSirasi.add("veli");  
yemekSirasi.add("ayse");  
yemekSirasi.add("kemal");  
  
System.out.println(yemekSirasi); // [ali, veli, ayse, kemal]  
System.out.println(yemekSirasi.element()); // ali  
System.out.println(yemekSirasi); // [ali, veli, ayse, kemal]  
  
System.out.println(yemekSirasi.peek()); // ali  
System.out.println(yemekSirasi); // [ali, veli, ayse, kemal]  
  
Queue<String> queDeneme= new LinkedList<>();  
System.out.println(queDeneme.peek()); // null  
// System.out.println(queDeneme.element()); // exception
```



# Collections / Queue

queue.**poll( )** : ilk elementi siler ve bize döndürür.

queue.**remove( )** : ilk elementi siler ve bize döndürür.

Aralarında sadece bir fark vardır. Bu bir queue için **poll( )** null sonucu donerken, **remove( )** exception oluşturur.

```
Queue<String> yemekSirasi= new LinkedList<>();  
  
yemekSirasi.add("ali");  
yemekSirasi.add("veli");  
yemekSirasi.add("ayse");  
yemekSirasi.add("kemal");  
  
Queue<String> queDeneme= new LinkedList<>();  
  
System.out.println(yemekSirasi.poll()); // ali  
System.out.println(yemekSirasi); // [veli, ayse, kemal]  
  
System.out.println(yemekSirasi.remove()); // veli  
System.out.println(yemekSirasi); // [ayse, kemal]  
  
System.out.println(queDeneme.poll()); // null  
// System.out.println(queDeneme.remove()); // exception
```



# Collections / Queue

queue.offer( ) : Bir kapasite sinirlamasi yoksa parametre larak girilen elementi listenin sonuna ekler.

```
public static void main(String[] args) {  
  
    Queue<String> yemekSirasi= new LinkedList<>();  
  
    yemekSirasi.add("ali");  
    yemekSirasi.add("veli");  
    yemekSirasi.add("ayse");  
    yemekSirasi.add("kemal");  
  
    System.out.println(yemekSirasi); // [ali, veli, ayse, kemal]  
    yemekSirasi.offer( e: "Hasan");  
    System.out.println(yemekSirasi); // [[ali, veli, ayse, kemal, Hasan]  
}
```

Islev acisindan queue.add( ) method'una benzer ancak kapasite sinirlamasi olan bir queue kullanildiginda add( ) yerine offer( ) tercih edilir.

# Collections / Deque

**Deque:** Double ended queue (iki taraflı kuyruk) demektir. Ekleme ve silme iki taraftan da yapılabilir.

Queue'da FIFO gecerli iken Deque'de hem FIFO hem de LIFO gecerlidir.

Double ended yapısından dolayı  
birçok method başa ve sona şeklinde  
iki şekilde kullanılmak üzere  
hazırlanmıştır.

**NOT :** deque null element kabul etmez.

**deque.pop( ) :** Bir deque'in ilk  
elementini siler ve bize döndürür.

**deque.push( ) :** Bir deque'in başına element ekler.

İşlev açısından queue.addFirst( ) method'una benzer ancak kapasite sınırlaması olan  
bir deque kullanıldığında addFirst( ) yerine push( ) tercih edilir.

```
Deque<String> stokList= new LinkedList<>();  
  
stokList.addLast( e: "Malz1");  
stokList.addFirst( e: "Malz2");  
System.out.println(stokList); // [Malz2, Malz1]  
  
System.out.println(stokList.pop()); // Malz2  
System.out.println(stokList); // [Malz1]  
  
stokList.push( e: "Malz3");  
System.out.println(stokList); // [Malz3, Malz1]
```

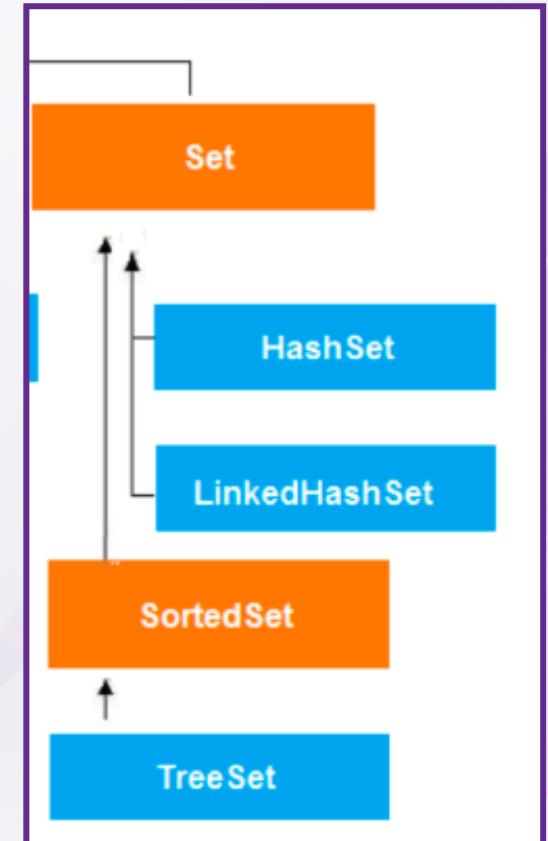
# Collections / Sets

Set tüm elementlerin unique olduğu collection'dır.

Matematikteki küme (set) mantığıyla çalışır. Okuldaki öğrenci numaraları, sosyal güvenlik numarası gibi dublication'in hatalara sebep olacağı elementleri store etmek için kullanılır.

Set matematikteki kume mantığıyla çalıştığı için sıralamaya bakmaz ve index yapısını desteklemez.

Sıralama önemli ise SortedSet interface'i altındaki TreeSet class'ı kullanabilir. TreeSet sıralama yaptığı için diğerlerine göre daha yavaş çalışır.



# Hash Mekanizmasi

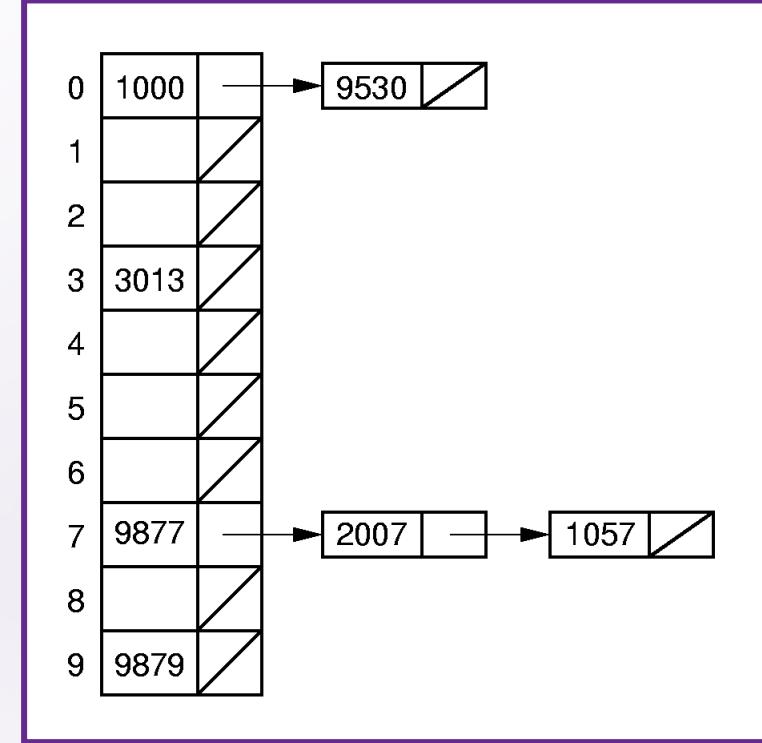
Hashing teknigi girdi ne olursa olsun belirlenen yapidaki bir output uretme islemidir.

Her hash algoritmasi, girdileri farkli sonuclarla donusturebilir. Ancak bir hash mekanizmasindan beklenen ayni girdi icin hep ayni sonucu uretmesidir.

Universitelerdeki numara algoritmasi hashCode'a iyi bir ornek olabilir.

Java'da kullanılan hash mekanizması da girilen tüm obje'ler için belirlenen yapıda bir hashCode üretir. Bu hashCode o objenin memory'de depolanması ve ihtiyac olduğunda ona ulaşılmasını kolaylaştırır.

Java'da hashCode kullanan HashSet gibi yapılar benzerlerine göre daha hızlı çalışmalar sayesinde one çıkarlar.



# Hash Mekanizması

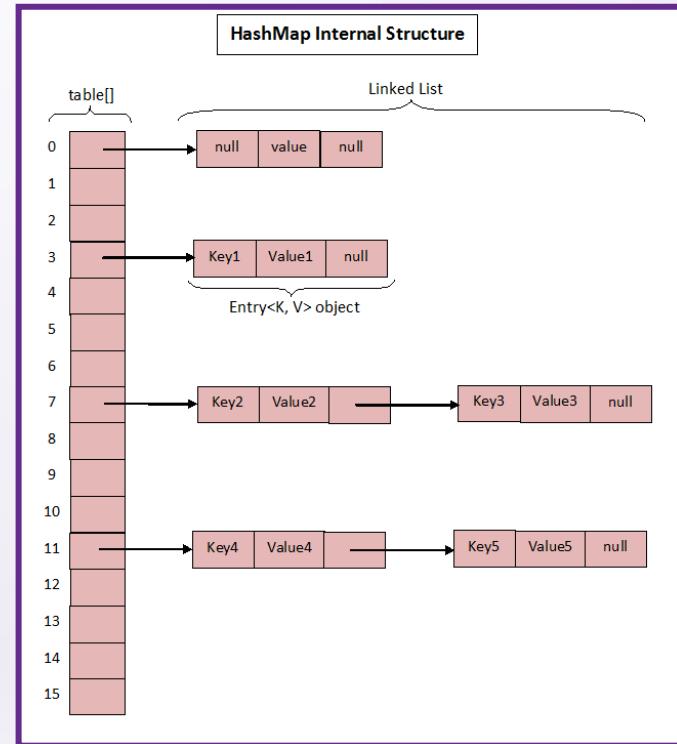
Bir HashCollection olusturuldugunda Java **16 bucket** olusturur ve elementleri bu bucket'lara yerlestirmeye baslar.

Olusturulan bucket'larin **%75'i doldugunda** Java 16 bucket daha olusturur. Buna **Load Factor** denir.

Java kullandigimiz herbir obje icin bir hash kod uretir. Eger uretilen hash kod daha once uretilen bir hash kod ile ayni ise buna **Hash Collision** denir

Hash Collision gerceklestiginde cozum icin 2 yol vardir.

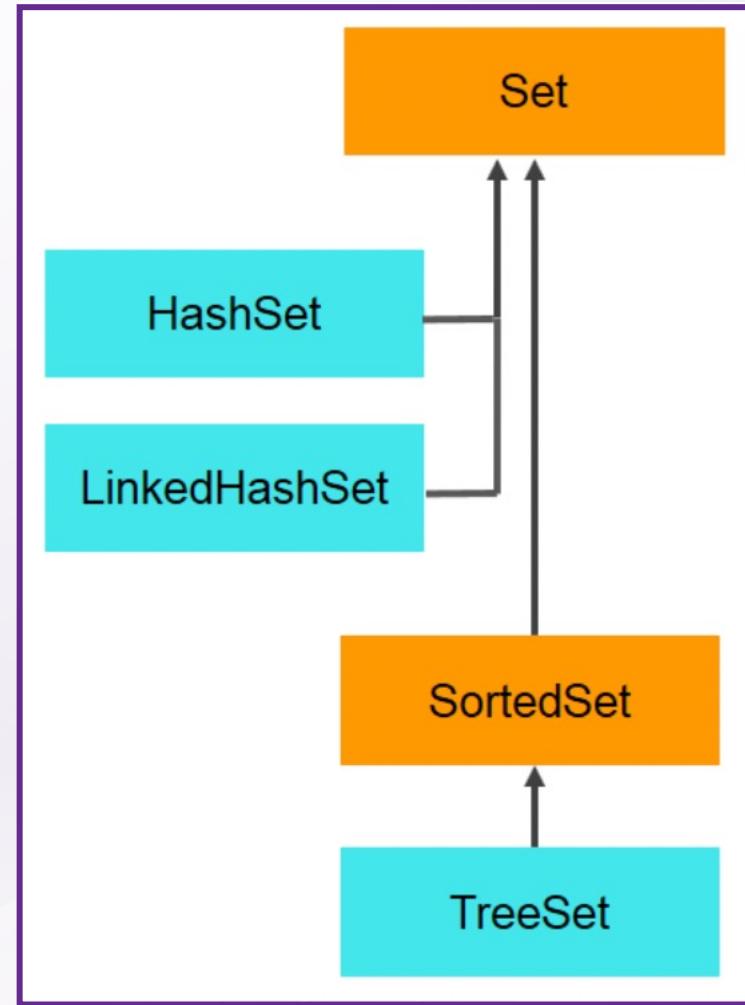
- A) **LinkedList kullanmak**
- B) **Formulle belirlenen yeni bir hash kod uretmek**



# HashSet

HashSet ;

- 1- Hash mekanizmasından dolayı hızlıdır.
- 2- Set olduğu için index yapısı yoktur, sıralama yapmaz. Elemanları yazdırdığımızda herhangi bir sırada gelebilir.
- 3- Set olduğu için bir değeri birden fazla girmenize (duplication) izin vermez. Aynı elementi bir daha girerseniz eskisini silip aynı olan yeni değeri ekler.
- 4- Eleman olarak null değeri kabul eder ancak set olduğu için birden fazla null ekleyemezsiniz
- 5- Index yapısını kullanmadığından tüm elemlere erişmek için for-each loop, değiştirmek için ise iterator kullanılabilir.



# HashSet Method'ları

Set ile kullanılan method'lar daha önce de kullandığımız ve bildigimiz method'lardır.

Burada unutulmamasi gereken konu, set'in index yapisini kullanmadigi, dolayisiyla elementlere index ile ulasmanin mumkun olmadigi ve elementleri cagirdigimizda rastgele bir sira ile gelecekleridir.

```
Set<Integer> sayilarKumesi= new HashSet<>();  
  
sayilarKumesi.add(10);  
sayilarKumesi.add(30);  
sayilarKumesi.add(50);  
sayilarKumesi.add(40);  
sayilarKumesi.add(20);  
  
System.out.println(sayilarKumesi); // [50, 20, 40, 10, 30]
```

```
Set<Integer> sayilarKumesi= new HashSet<>();  
sayilarKumesi.|  
    ▾ add(Integer e)  
    ▾ addAll(Collection<? extends Integer> c)  
    ▾ hashCode()  
    ▾ remove(Object o)  
    ▾ sout  
    ▾ clear()  
    ▾ contains(Object o)  
    ▾ containsAll(Collection<?> c)  
System.out.|  
    ▾ equals(Object o)  
    ▾ isEmpty()  
    ▾ removeAll(Collection<?> c)  
    ▾ retainAll(Collection<?> c)  
    ▾ size()  
    ▾ toArray()  
    ▾ toArray(T[] a)  
    ▾ iterator()
```



# HasSets

Soru : Verilen bir array'deki tekrar eden elementleri silerek, her element'den sadece bir tane olacak sekilde yeni bir array olusturun.

```
public static void main(String[] args) {  
  
    int[] arr= {2,3,4,2,6,4,5,3,2,4,6,5,7,9};  
  
    Set<Integer> temp= new HashSet<>();  
  
    for (Integer each:arr  
         ) {  
        temp.add(each);  
    }  
  
    arr=new int[temp.size()];  
  
    int index=0;  
  
    for (Integer each: temp  
         ) {  
        arr[index] = each;  
        index++;  
    }  
  
    System.out.println(Arrays.toString(arr)); // [2, 3, 4, 5, 6, 7, 9]  
}
```



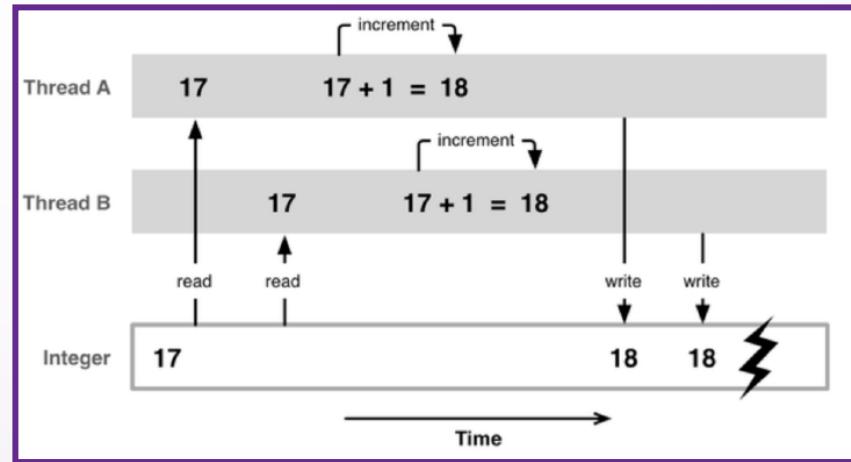
# HasSets

Soru : Bir HashSet ve TreeSet olusturun.  
Bir loop ile bu set'lere element ekleyip sureleri karsilastirin.

HashSet islem suresi : 7  
TreeSet islem suresi : 12

```
Set<Integer> hashSet= new HashSet<>();  
  
Set<Integer> treeSet= new TreeSet<>();  
  
Random rnd= new Random();  
int temp;  
  
Long hashSetBaslangic=System.currentTimeMillis();  
for (int i = 0; i <100000 ; i++) {  
    temp= rnd.nextInt( bound: 1000);  
    hashSet.add(temp);  
}  
Long hashSetBitis=System.currentTimeMillis();  
  
System.out.println(hashSetBitis-hashSetBaslangic);  
  
Long treeSetBaslangic=System.currentTimeMillis();  
for (int i = 0; i <100000 ; i++) {  
    temp= rnd.nextInt( bound: 1000);  
    treeSet.add(temp);  
}  
Long treeSetBitis=System.currentTimeMillis();  
  
System.out.println(treeSetBitis-treeSetBaslangic);
```

# Multi Thread



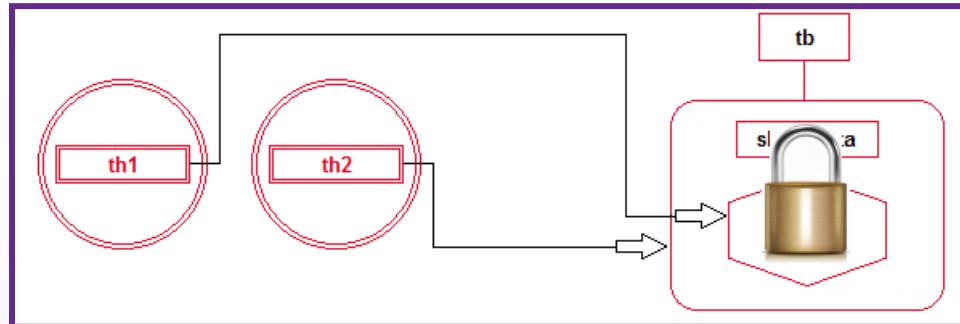
Multithreading CPU kullanımını en verimli hale getirmek için birden fazla thread'in aynı anda çalıştırılmasıdır.

Thread(is parçacığı) bir program sırasında çalışan her bir process'tır. JVM aynı anda birden fazla thread'in çalışmasına izin verir.

Birden fazla thread aynı anda çalışığında java'da kullanılan variable'larda kesismeler olusabilir.

Birden fazla thread aynı anda çalışığında JVM oncelikli thread'leri daha önce yapar fakat oncelik sıralaması yoksa thread'ler arasında bir senkronizasyon oluşturulmalıdır.

# Synchronizing



Multithread programlar calistiginda ayni variable'lara erismeye calisann birden fazla thread olabilir.

Yapilmaya calisilan process uzun olsa da thread'lerin kesisimi belirli bolgelerde yogunlasacaktir.

Java'da bu kesisim noktalari senkronize bloklara konulur ve bu bloklara ayni anda sadece bir thread'in erisimine izin verilir.

Java'da bu kesisim noktalari senkronize bloklara konulur ve bu bloklara ayni anda sadece bir thread'in erisimine izin verilir.



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-43

### Maps



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

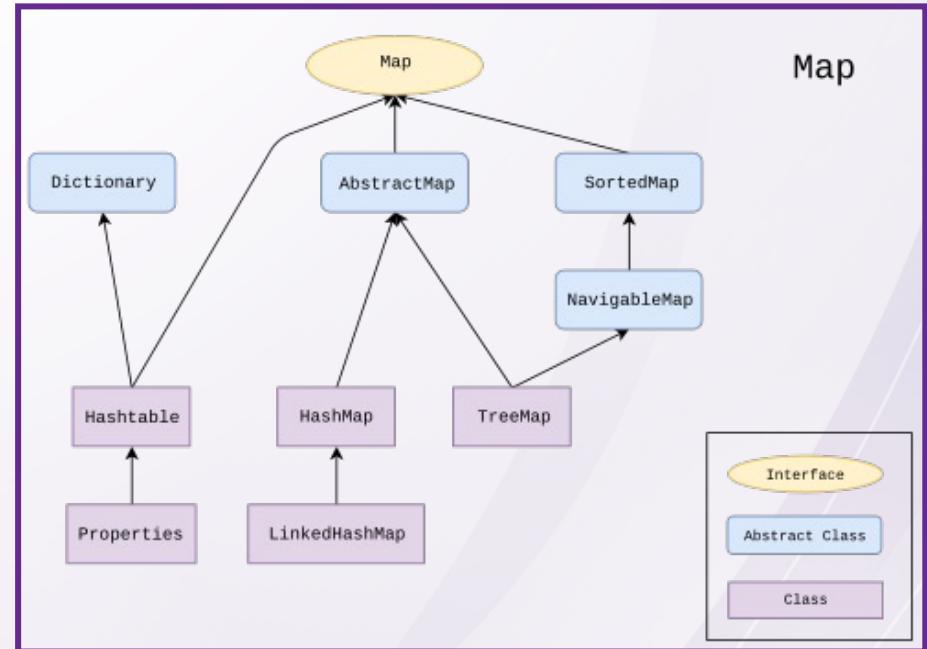
/wisequarter

# Maps

Map key-value pairs(ikilileri) kullanır.

Bu yapısı sayesinde reel uygulamalarda kullanılan database yapısına en uygun Java yapısıdır.

Her bir element için bir key ve bir value olmalıdır.



Elementler key değerleri ile tutuldugundan, key'lerin unique olması gereklidir.

Value için unique olma mecburiyeti yoktur. Value String bir ifadeye eklenecek bir çok bilgiyi içerebilir ama map value olarak tek bir değer store eder.

# Maps ve MultiThread

- HashMap ve TreeMap multi thread calismayi desteklemez yani thread-safe degildir. Synchronized ozelligi yoktur.
- Multi thread calisma icin HashTable kullanilabilir.





# Maps

Map ile aynı özelliklere sahip birden fazla objeyi store edebilirsiniz.

Objelere ait özelliklerini value olarak kaydederiz. Sonradan bu özelliklere ulaşabilmek için manipulation method'larını kullanacağımızdan, tüm objelerin value'ları aynı özellikleri aynı sırada barındırmalı, aralarında kullanacağımız ayıracı da standart olmalıdır.



No:101 Isim:Ali soyisim:Can sinif:10 sube:H bolum:MF

Ogrenci 1



No:102 Isim:Veli soyisim:Cem sinif:11 sube:M bolum:Soz

Ogrenci 2



No:103 Isim:Ali soyisim:Cem sinif:11 sube:H bolum:TM

Ogrenci 3

```
Map<Integer,String> ogrenciMap = new HashMap<>();  
ogrenciMap.put(101, "Ali-Can-10-H-MF");  
ogrenciMap.put(102, "Veli-Cem-11-M-Soz");  
ogrenciMap.put(103, "Ali-Cem-11-H-TM");
```

{101=Ali-Can-10-H-MF, 102=Veli-Cem-11-M-Soz, 103=Ali-Cem-11-H-TM}

Ogrenci 1

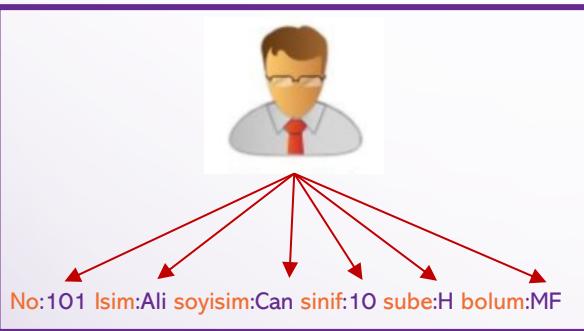
Ogrenci 2

Ogrenci 3

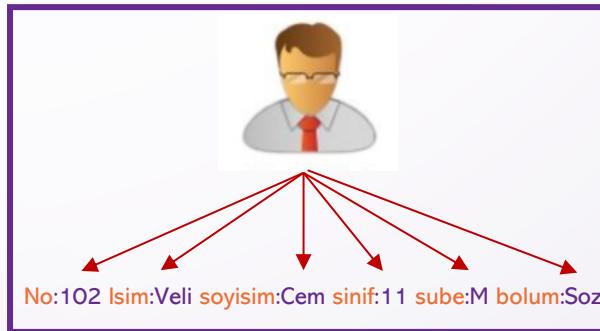


# Maps

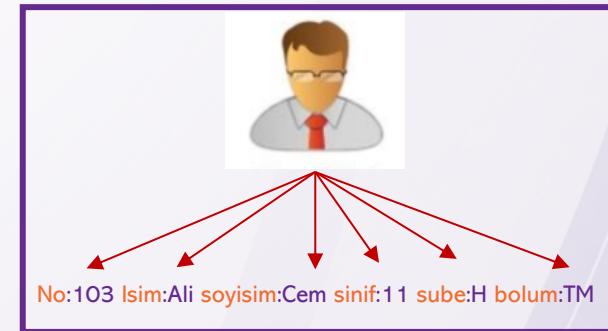
Map ile key ve value degerlerine ayri ayri method'lar ile ulasabiliriz.



Ogrenci 1



Ogrenci 2



Ogrenci 3

```
System.out.println(ogrenciMap.keySet());
```



```
[101, 102, 103]
```

keyset( ) ve values( ) method'lari Collection olarak key ve value degerlerini verir.

```
System.out.println(ogrenciMap.values());
```



```
[Ali-Can-10-H-MF, Veli-Cem-11-M-Soz, Ali-Cem-11-H-TM]
```

Ogrenci 1

Ogrenci 2

Ogrenci 3

# Maps

Map kompleks bilgi tuttugu olcude, bir bilgiye ulasmak da zorlasacaktir.

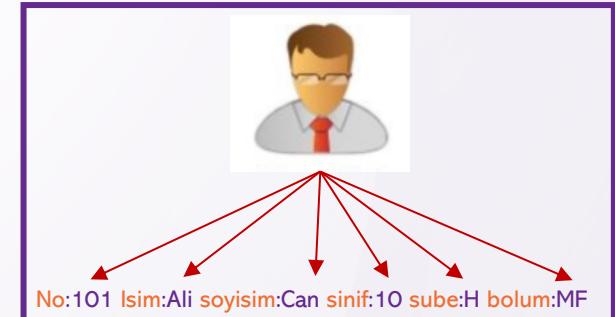
**Soru:** H subesindeki ogrencilerin isim ve soyisimlerini bir liste olarak yazdirin.

1- isimleri eklemek icin bos bir liste olustur.

2- Tum value'leri koyabilecegimiz bir Collection olustur ve value'leri store et.

3- For-each loop ile tum value'leri teker teker islenmek uzere ele al.

4- Her bir value'u ayirac'i kullanarak split et, bir array'e koyup, sube bilgisini tutan index'i kontrol et, istenen sube ise isim ve soyismi listeye ekle.



Ogrenci 1

Ali-Can-10-H-MF

```
Collection<String> ogrenciValueColl= ogrenciMap.values();
List<String> istenenSubedekiler= new ArrayList<>();
String[] valueArr;

for (String each: ogrenciValueColl
) {

    valueArr=each.split( regex: "-");
    if (valueArr[3].equalsIgnoreCase(sube)){
        istenenSubedekiler.add(valueArr[0]);
    }
}
```



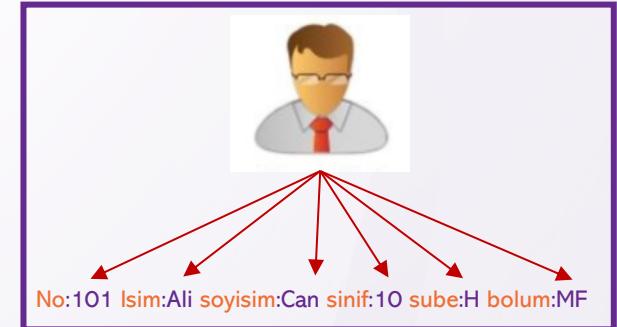
# Maps

**Value Update:** Update istenen element'in key'i belli ise once value'sune ulasir, onda istenen degisikligi yapip yeniden ayni key ve yeni deger ile map'e ekleyebiliriz.

**Soru:** 101 numarali ogrencinin soyismini Yan yapin

```
Map<Integer, String> ogrenciMap= ReusableMethods.ogrenciMapOlustur();
String ogrenciEskiBilgiler= ogrenciMap.get(101);
System.out.println(ogrenciEskiBilgiler); // Ali-Can-10-H-MF
String[] ogrenciValueArr=ogrenciEskiBilgiler.split( regex: "-");
System.out.println(Arrays.toString(ogrenciValueArr)); // [Ali, Can, 10, H, MF]
ogrenciValueArr[1]= "Yan";
System.out.println(Arrays.toString(ogrenciValueArr)); // [Ali, Yan, 10, H, MF]
String ogrenciYeniValue= ogrenciValueArr[0]+"-"+
    ogrenciValueArr[1]+"-"+
    ogrenciValueArr[2]+"-"+
    ogrenciValueArr[3]+"-"+
    ogrenciValueArr[4];

ogrenciMap.put(103,ogrenciYeniValue);
System.out.println(ogrenciMap);
// {101=Ali-Can-10-H-MF, 102=Veli-Cem-11-M-Soz, 103=Ali-Yan-10-H-MF, 104=Ayca-C}
```



Ogrenci 1

Ali-Can-10-H-MF

- 1- istenen key'e ait value'yu bir String'e kaydet.
- 2- Value'yu ayirac ile split yapip bir array'e kaydet
- 3- Array'de istenen update'i yap
- 4- Array'deki bilgileri birlestirip value formatina uygun bir String yap
- 5- Yeni value'yu key ile map'e ekle



Wise Quarter  
first class IT courses

# JavaTeam 120

## Ders-45

### Maps



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)

/wisequarter



# Onceki Dersten Aklimizda Kalanlar

- 1- Map : Map bugune kadar gordugumuz tum Java yapilarindan farklidir.
- 2- Map'i diger java yapilarindan ayiran en buyuk ozelligi key-value yapisinda olmasidir.
- 3- Java map'deki elementleri key'e gore tuttugu icin key degerleri unique olmalidir. Eger map'de var olan bir key ile yeni element eklemek istersek, eski value silinip yeni eklenen value o key ile iliskendirilir
- 4- Map guncel projelerde kullanilan data base'e en yakin Java objesidir. Ozellikle ihtiyac oldugunda nested map'ler kullanilarak bir cok uygulamanin java versiyonu yapilabilir.
- 5- Bir obje icerisinde tutulan data ne kadar kompleks olursa , o datalara erisim ve update o kadar zor olur.
- 6- Map'de bir bilgiye ulasmak ve update etmek bir cok islem gerektirdiginden, genellikle bu tur islemleri method olarak olusturmayi tercih ederiz.
- 7- Method'larda istedigimiz bilgiye ulasincaya kadar map'i parcalara ayirip, istedigimiz bilgiye ulastiktan veya update ettikten sonra yeniden map'e eklememiz ve map'i guncellememiz gerekir

# Maps / Entry Set

**Entry (Interface):** Bir map'deki key ve value ikililerini birlikte kullanmak istedigimizde Entry interface'indeki method'lari kullaniriz.

```
{101=Ali-Can-10-H-MF, 102=Veli-Cem-11-M-Soz, 103=Ali-Yan-10-H-MF, 104=Ayca-Can-11-B-MF, 105=Ayse-Cem-10-M-Soz}
```

maplsmi.entrySet( ) method'u entry'lerden olusan bir Set döndürür.

entry'ler map'deki data turunde olan key ve value'ler icerir. Entry'leri kaydetmek icin data turlerine dikkat ederek, entry'lerden olusan bir Set olusturmaliyiz..

```
Set<Map.Entry<Integer, String>> ogrenciEntrySeti=ogrencimap.entrySet();
```

entry'lerden olusan bu set'i for-each loop ile istedigimiz gibi isleyebiliriz.

```
101=Ali-Can-10-H-MF
102=Veli-Cem-11-M-Soz
103=Ali-Cem-11-B-TM
104=Ayca-Can-11-B-MF
105=Ayse-Cem-10-M-Soz
```

Entry interface'indeki getKey( ) ve getValue( ) method'lari ile entry'leri yandaki gibi yazdirabilir veya degistirdigimiz value'lari setValue(yeniDeger) method'unu kullanarak update edebiliriz.

# Maps

Soru : 10. siniftaki ogrencilerin no, isim, soyisim, bolumlerini bir liste olarak yazdirin

Soru'da key (ogrenci no) ve value birlikte istendigi icin entry kullanmak daha isabetli olacaktir.

entry kullanarak soruyu cozmek icin

- 1- Calisilacak map'deki key-value data turlerine uygun olarak, entry'lerden olusan bir Set olusturup, map'deki entryleri maplsmi.entrySet( ) method'unu kullanarak olusturulan Set'e store et.
- 2- For-each loop ile entry Set'indeki entry'leri islemek icin tek tek ele al.
- 3- Value'daki verilerden istenenleri kullanabilmek icin, value'yu ayirac ile split yapip bir array'e kaydet
- 4- Key ve Value'daki istenen bilgileri yazdir

```
// 10. siniftaki ogrencilerin no, isim, soyisim, bolumlerini
// bir liste olarak kullaniciya yazdirin
// Baslik satiri : No Isim Soyisim Bolum

Map<Integer, String> ogrenciMap= ReusableMethods.ogrenciMapOlustur();

Set<Map.Entry<Integer, String>> ogrenciEntrySeti=ogrenciMap.entrySet();
Integer tempKey;
String tempValue;
String[] tempValueArr;

for (Map.Entry<Integer, String> each: ogrenciEntrySeti
) {

    tempKey=each.getKey();
    tempValue= each.getValue();
    tempValueArr=tempValue.split( regex: "-"); // [Ali, Can, 10, H ,MF]

    if (tempValueArr[2].equals("10")){
        System.out.println(tempKey+ " "+
            tempValueArr[0]+ " "+
            tempValueArr[1]+ " "+
            tempValueArr[4]);
    }
}
```

# Maps

Soru : Map'deki soyisimleri buyuk harfe cevirin.

1- Calisilacak map'deki key-value data turlerine uygun olarak, entry'lerden olusan bir Set olusturup, map'deki entryleri maplsmi.entrySet( ) method'unu kullanarak olusturulan Set'e store et.

2- For-each loop ile entry Set'indeki entry'leri islemek icin tek tek ele al.

3- Value'daki verilerde istenen update'leri yapabilmek icin, value'yu ayirac ile split yapip bir array'e kaydet

4- Array'de istenen update'leri yap

5- Array'de yeniden map'e donmek icin, array'i String'e döndür, String'i **setValue(yeniDeger )** ile map'de update et.

```
Set<Map.Entry<Integer, String>> ogrenciEntrySeti= ogrenciMap.entrySet();

Integer tempkey;
String tempValue;
String[] tempValueArr;

for (Map.Entry<Integer, String> each: ogrenciEntrySeti
) {

    tempValue= each.getValue();
    tempValueArr=tempValue.split( regex: "-"); // [Ali, Can, 10, H ,MF]

    tempValueArr[1]=tempValueArr[1].toUpperCase(); // // [Ali, CAN, 10, H ,MF]

    tempValue=tempValueArr[0] +"-"+
        tempValueArr[1] +"-"+
        tempValueArr[2] +"-"+
        tempValueArr[3] +"-"+
        tempValueArr[4]; // // Ali-CAN-10-H-MF

    each.setValue(tempValue);
}
```



# Onceki Dersten Aklimizda Kalanlar

1- Map'ler kompleks bilgileri tuttuklari icin bilgilere ulasmak ve update etmek kolay olmayabilir.

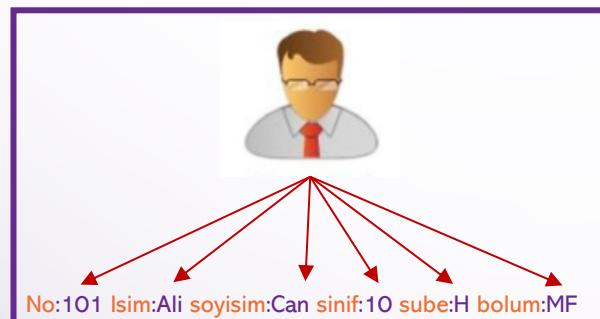
2- Map'de yapilacak islemler, genelde tek adimda yapilamayacagi icin, calistigimiz map ile ilgili yaptigimiz tum islemleri bir method olarak kaydetmek daha faydalı olacaktir. Boylece sonraki kullanimlarda method'umuz hazir olur

3- Map ile yapacagimiz yazdirma ve update islemleri icin once bilgiye ulasincaya kadar map'i parcalamaliyiz

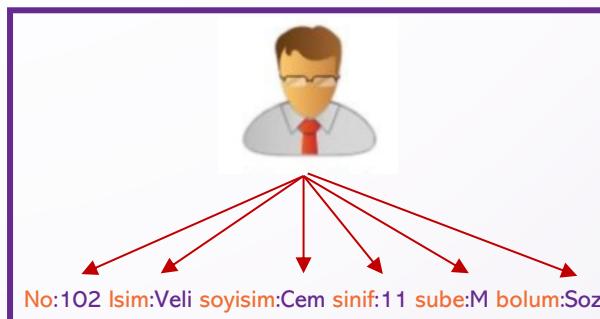
- EntrySet ile tum kayitlari entry seklinde ele alabiliriz
- for-each ile entrySet'deki her bir entry'i ele aliriz
- value nun durumuna gore bilgilere ulasabilecegimiz sekilde boleziz
- value ve key bilgilerinden istediklerimiz yazdirabiliriz.
- Eger update yapilirsa bilgileri map'e yeniden ekleuecek sekilde duzenleriz
- Duzenledigimiz value'yu setValue( ) ile map'de update ederiz



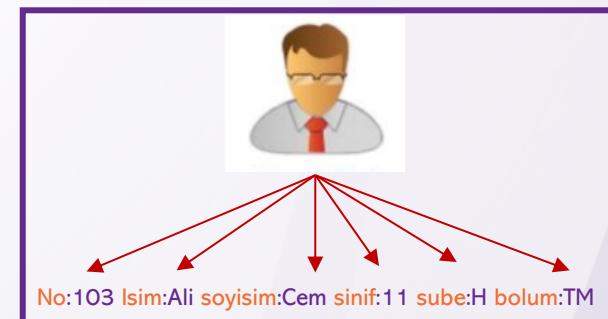
# Nested Maps



## Ogrenci 1



## Ogrenci 2



Ogrenci 3

Istenirse map içerisinde value olarak map kullanılabılır. Bu durumda yapının karmaşıklaşacağı ve erişmek için daha fazla manipülasyon gerekeceğini unutmamak gereklidir.

# Nested Maps

Map ile bir objeye ait farkli bilgileri kategorilerine gore ayirip depolayabiliriz

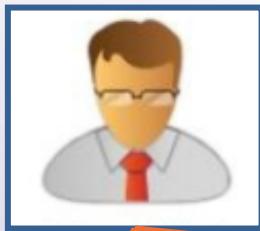


# Nested Maps



Java Bank

Map Bank = { M.No1={Musteri Map1}, M.No2={Musteri Map2}, ... }



Musteri : Ali Can

Musteri Map1 = {KisiselBilgiler={Kis.Bil.Map} , TIHesap={TIHesapMap}...}

TIHesapMap = {HesapNo="12345" , Kur="TL", AcTar="1.1.2021", Bakiye="0" }

Tc No:  
İsim :  
Soyisim :  
Telefon:

Hesap No:  
Kur :  
Acilis Tar. :  
Bakiye:

Hesap No:  
Kur :  
Acilis Tar. :  
Bakiye:

Hesap No:  
Kur :  
Acilis Tar. :  
Bakiye:

Kart No:  
Limit :  
Verilis Tar. :  
Kul.Limit :

Kisisel  
bilgiler

TL  
Hesap

Euro  
Hesap

Usd  
Hesap

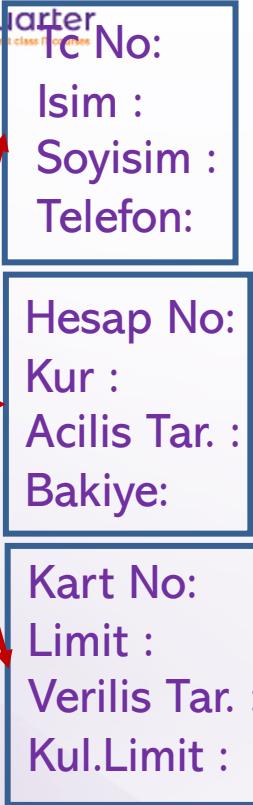
Kredi  
Karti



# Nested Maps



Musteri :  
Ali Can



## Kisisel bilgiler

{Soyisim=Can, TcNo=12345678901, İsim=Ali, Telefon=5553456789}

## Hesaplar

{Bakiye=0, Acilis Tarihi=12.12.2021, Kur=TL, HesapNo=1234-1}

{Bakiye=0, Acilis Tarihi=12.12.2021, Kur=Eur, HesapNo=1234-2}

{Bakiye=0, Acilis Tarihi=12.12.2021, Kur=Usd, HesapNo=1234-3}

## Kredi Karti

{Kul.Limit=11000, Limit=20000, KartNo=1234567890123456,  
Verilis Tarihi=12.12.2021}

{ KrediKarti={ Kul.Limit=11000, Limit=20000, KartNo=1234567890123456, Verilis Tarihi=12.12.2021 },

    UsdHesap={ Bakiye=0, Acilis Tarihi=12.12.2021, Kur=Usd, HesapNo=1234-2 },

    KisiselBilgiler={ Soyisim=Can, TcNo=12345678901, İsim=Ali, Telefon=5553456789 },

    EuroHesap={ Bakiye=0, Acilis Tarihi=12.12.2021, Kur=Eur, HesapNo=1234-2 },

    TLHesap={ Bakiye=0, Acilis Tarihi=12.12.2021, Kur=TL, HesapNo=1234-1 } }

# Maps

**Soru :** Verilen bir array'de kullanılan sayıları ve kaçar defa kullanıldığını yazdırın.

**Input :** {1,2,3,4,5,3,4,2,5,1,3,2,4,1}

**output :** 1 kullanımı : 3 adet

2 kullanımı : 3 adet

3 kullanımı : 3 adet

4 kullanımı : 3 adet

5 kullanımı : 2 adet

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

    int[] arr={13,5,6,54,32,3,4,6,2,4,5};
    Map<Integer,Integer> kullanımSayilarıMap= new HashMap(); // {}

    // he def  { 1=3, 2=3, 3=3, 4=3, 5=2}

    for (int each: arr
         ) {
        // each'in key olarak map'de olup olmadığını kontrol edelim
        if (kullanımSayilarıMap.containsKey(each)){
            // key map'de varsa
            kullanımSayilarıMap.put (each,(kullanımSayilarıMap.get(each) +1));
        }else {
            // key map'de yoksa (o zaman ekle)

            kullanımSayilarıMap.put(each,1);
        }
    }

    System.out.println(kullanımSayilarıMap); // {1=6, 2=3, 3=3, 4=3, 5=2}

    Set<Map.Entry<Integer,Integer>> kullanımSayilarıEntrySet=kullanımSayilarıMap.entrySet();

    for (Map.Entry<Integer,Integer> each: kullanımSayilarıEntrySet
         ) {
        System.out.println(each.getKey()+" kullanım : "+ each.getValue()+" adet" );
    }
}
```

# Maps

Soru : Verilen bir ogrenci map'inde her sinifta kacar ogrenci oldugunu yazdiran bir method olusturun.

```
// {101=Ali-Can-10-H-MF, 102=Veli-Cem-11-M-Soz, 103=Ali-Cem-11-B-TM,  
// 104=Ayca-Can-11-B-MF, 105=Ayse-Cem-10-M-Soz}  
// beklenen sonuc {10=2, 11=3}  
  
Map<String, Integer> sinifSayilariMap= new HashMap<>();  
// ogrenci map'ini entry'lere ayirip sinif kontrolu yapmaliyiz  
Set<Map.Entry<Integer, String>> ogrenciMapEntrySeti= ogrenciMap.entrySet();  
  
for (Map.Entry<Integer, String> entry: ogrenciMapEntrySeti  
) {  
// elimizde 101=Ali-Can-10-H-MF gibi entry'ler var  
String[] tempValueArr= entry.getValue().split( regex: "-"); // [Ali, Can, 10, H, MF]  
  
String sinifBilgisi=tempValueArr[2];  
// su an elimizde sinif bilgisi var  
// bunu yeni olusturdugumuz sinifSayilariMap'e ekleyecegiz  
if (sinifSayilariMap.containsKey(sinifBilgisi)){  
// onceden bu sinifdan ogrenci girilmiş demektir, ogrenci sayisini 1 artirmaliyim  
sinifSayilariMap.put(sinifBilgisi,sinifSayilariMap.get(sinifBilgisi)+1);  
}else{  
// daha once bu siniftan hic ogrenci girilmemiş demektir  
// bu sinifta 1 ogrenci oldu demem gerekir  
  
sinifSayilariMap.put(sinifBilgisi,1);  
}  
}  
System.out.println(sinifSayilariMap);
```

# Maps

**Soru :** Verilen bir ogrenci map'inde her sinifta kacar ogrenci oldugunu yazdiran bir method olusturun.

Soruda sinifBilgisi'nin output map'inde olup olmadigini kontrol edip varsa value'yu bir artirdik, yoksa value 1 olarak ekledik.

Bunu yapabilmek icin hazir method'lar da var.

```
String sinifBilgisi=tempValueArr[2];
// su an elimizde sinif bilgisi var
// bunu yeni olusturdugumuz sinifSayilariMap'e ekleyecegiz
if (sinifSayilariMap.containsKey(sinifBilgisi)){
    // onceden bu sinifdan ogrenci girilmis demektir, ogrenci sayisini 1 artirmaliyim
    sinifSayilariMap.put(sinifBilgisi,sinifSayilariMap.get(sinifBilgisi)+1);
}else{
    // daha once bu siniftan hic ogrenci girilmemis demektir
    // bu sinifta 1 ogrenci oldu demem gerekir

    sinifSayilariMap.put(sinifBilgisi,1);
}
```

```
String sinifBilgisi=tempValueArr[2];

sinifSayilariMap.computeIfPresent(sinifBilgisi,(k,v)->v+1);
sinifSayilariMap.putIfAbsent(sinifBilgisi,1);
```



# Maps Method'lari

mapismi.put(key, value) : Istenen key ve value'ye sahip elementi map'e ekler

```
Map<String, Integer> ornekMap=new HashMap<>();  
  
ornekMap.put("a",3);  
ornekMap.put("b",1);  
ornekMap.put("c",2);  
ornekMap.put("d",5);  
  
System.out.println(ornekMap); // {a=3, b=1, c=2, d=5}  
  
ornekMap.put("b",7); // var mi, yok mu kontrol etmeden direkt ekler  
ornekMap.putIfAbsent("a",7); // yoksa ekle dedigimizden a var oldugu icin eklemez  
ornekMap.putIfAbsent("e",3); // yoksa ekle dedigimizden e yok oldugu icin ekler  
  
System.out.println(ornekMap); // {a=3, b=7, c=2, d=5, e=3}
```

Eger eklenen key daha onceden map'de varsa, o key'e ait value yeni eklenen value olarak update edilir.

mapismi.putIfAbsent(key, value) : Eklenmek istenen key ve value map'de daha onceden mevcut ise ekleme yapmaz, map'de yoksa ekleme yapabilir.



# Maps Method'lari

mapismi.**compute(key, (k,v)-> yeni deger)** : istenen key'e sahip elementin value'sunu hesaplama veya atama yontemiyle update eder.

mapismi.**computeIfPresent(key, (k,v)-> yeni deger)** : istenen key'e sahip element map'de varsa value'sunu hesaplama veya atama yontemiyle update eder, map'de yoksa birsey yapmaz.

```
System.out.println(ornekMap); // {a=2, b=7, c=2, d=5, e=3, g=3, h=4}
```

```
ornekMap.compute(key: "a", (k,v)->2*v); // a'nin degerini 4 yapacak
```

```
System.out.println(ornekMap); // {a=4, b=7, c=2, d=5, e=3, g=3, h=4}
```

```
ornekMap.computeIfPresent(key: "c", (k,v)-> 20);
```

```
ornekMap.computeIfPresent(key: "k", (k,v)->30);
```

```
System.out.println(ornekMap); // {a=4, b=7, c=20, d=5, e=3, g=3, h=4}
```

```
ornekMap.computeIfAbsent(key: "c", v->15); // islem yapmaz
```

```
ornekMap.computeIfAbsent(key: "m", v->12); // ekler
```

```
System.out.println(ornekMap); // {a=4, b=7, c=20, d=5, e=3, g=3, h=4, m=12}
```

mapismi.**computeIfAbsent(key, (k,v)-> yeni deger)** : istenen key'e sahip element map'de yoksa value'sunu hesaplama veya atama yontemiyle update eder, map'de varsa birsey yapmaz.



Wise Quarter  
first class IT courses

# JavaTeam 116

## Ders-34

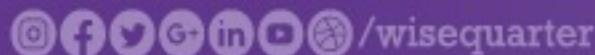
### Nested Maps



The future at your fingertips

+1 912 888 1630

[www.wisequarter.com](http://www.wisequarter.com)



# Maps Method'lari

mapismi.containsKey(key) : istenen key map'de varsa true, yoksa false döndürür.

mapismi.containsValue(value) : istenen value map'de varsa true, yoksa false döndürür.

```
Map<Integer, String> ogrenciMap= ReusableMethods.ogrenciMapOlustur();
System.out.println(ogrenciMap);
// {101=Ali-Can-10-H-MF, 102=Veli-Cem-11-M-Soz, 103=Ali-Cem-11-B-TM,
// 104=Ayca-Can-11-B-MF, 105=Ayse-Cem-10-M-Soz}

System.out.println(ogrenciMap.containsKey(102)); // true

System.out.println(ogrenciMap.containsValue("Ali")); // false
// containsValue("IstenenDeger") method'u tam olarak value girilirse true doner
// ancak value icerisindeki bir degeri aratırsanız false doner
System.out.println(ogrenciMap.containsValue("Veli-Cem-11-M-Soz")); // true
```

Eger bu method'da tum value degil, value'nun bir parcasini value olarak girerseniz false döndürür.

Bu method'un döndürmesi icin girilen value'nun map'de olan value'lerden biri ile tamamen ayni olmasi gerekir.



# Maps Method'lari

`mapismi.replace(key,yeniDeger)` : istenen key'in degerini update edip, eski degerini döndürür.

`mapismi.replace(key, eskiDeger, yeniDeger)` : istenen key'in eski degeri yazdigimiz gibi ise update edip, true döndürür.

Eski value yazdigimizdan farkli ise islem yapmaz ve false döndürür.

```
System.out.println(ornekMap); // {a=3, b=1, c=2, d=5}
System.out.println(ornekMap.replace("d", 10)); // 5 eski degerini döndürür
System.out.println(ornekMap); // {a=3, b=1, c=2, d=10}

System.out.println(ornekMap.replace( key: "a", oldValue: 5, newValue: 8)); // false
System.out.println(ornekMap.replace( key: "b", oldValue: 1, newValue: 2)); // true
System.out.println(ornekMap); // {a=3, b=2, c=2, d=10}

System.out.println(ornekMap.get("b")); // 2
System.out.println(ornekMap.get("z")); // null
System.out.println(ornekMap.getOrDefault( key: "a", defaultValue: 6)); // 3
System.out.println(ornekMap.getOrDefault( key: "m", defaultValue: 0)); // 0
System.out.println(ornekMap); // {a=3, b=2, c=2, d=10}
```

`mapismi.get(key)` : Girilen key map'de varsa, o key'e ait value'yu, girilen key map'de yoksa null döndürür.

`mapismi.getOrDefault(key, defaultValue)` : Girilen key map'de yoksa null degil belirledigimiz bir deger(ornegin 0) döndürmesini istiyorsak bu method kullanılır.

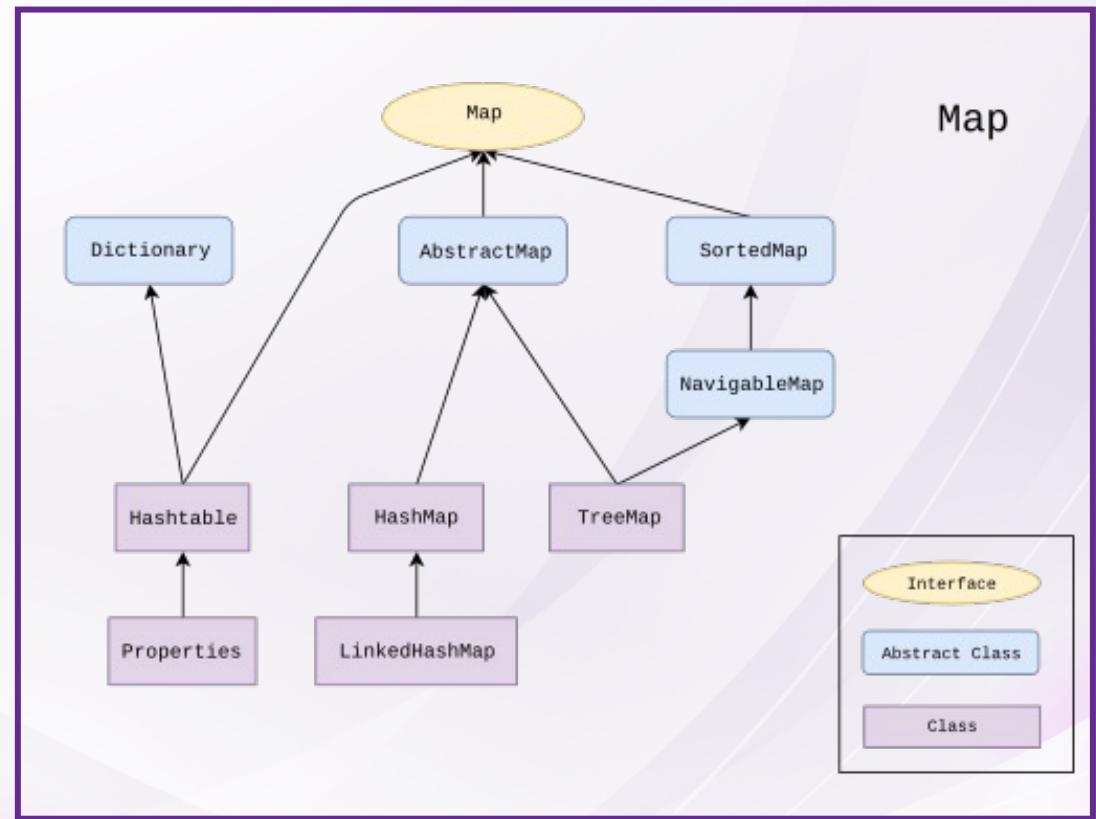


# Tree Maps

Tree map NavigableMap ve SortedMap Abstract class'larini inherit ettiginden Map interface'inde olmayan bazi ekstra method'lara sahiptir.

Inheritance kurallari geregi bu ozellikleri kullanabilmesi icin data turu olarak Map degil, SortedMap, NavigableMap veya TreeMap secilmelidir.

```
TreeMap<String, Integer> ornekMap=new TreeMap<>();  
  
ornekMap.  
    put(String key, Integer value)  
ornekMap.  
    higherKey(String key)  
ornekMap.  
    headMap(String toKey)  
ornekMap.  
    descendingMap()  
ornekMap.  
    headMap(String toKey, boolean inclusive)  
ornekMap.  
    floorEntry(String key)  
    firstKey()  
System.  
    ceilingKey(String key)  
    ceilingEntry(String key)  
System.  
    clear()  
System.  
    clone()  
System.  
    comparator()  
System.  
    containsKey(Object key)  
    containsValue(Object value)  
    descendingKeySet()
```





# Tree Maps

`treeMapismi.ceilingEntry(key)` : verilen key treeMap'de varsa, o key'e ait entry'yi, verilen key treeMap'de yoksa, verilen key'den büyük olan ilk key'e ait entry'yi döndürür.

Eğer verilen key treeMap'de yoksa ve girilen key degerinden büyük başka key yoksa null döndürür.

`treeMapismi.ceilingKey(key)` :  
verilen key treeMap'de varsa, o key'i, verilen key treeMap'de yoksa verilen key'den büyük olan ilk key'l döndürür.

```
System.out.println(ornekMap); // {a=3, b=5, k=2, r=1}
```

```
System.out.println(ornekMap.ceilingEntry("b")); // b=5
System.out.println(ornekMap.ceilingEntry("c")); // k=2
System.out.println(ornekMap.ceilingEntry("m")); // r=1
System.out.println(ornekMap.ceilingEntry("t")); // null
```

```
System.out.println(ornekMap.ceilingKey("b")); // b
System.out.println(ornekMap.ceilingKey("c")); // k
```

```
System.out.println(ornekMap.floorEntry("b")); // b=5
System.out.println(ornekMap.floorKey("d")); // b
System.out.println(ornekMap.floorKey("k")); // k
System.out.println(ornekMap);
```

`treeMapismi.floorKey(key)` :

`treeMapismi.floorEntry(key)` : verilen key treeMap'de yoksa verilen key'den küçük olan ilk key'l döndürür.



# Tree Maps

treeMapismi.ceilingEntry(key) : verilen key treeMap'de varsa, o key'e ait entry'yi, verilen key treeMap'de yoksa, verilen key'den büyük olan ilk key'e ait entry'yi döndürür.

Eğer verilen key treeMap'de yoksa ve girilen key değerinden büyük başka key yoksa null döndürür.

```
System.out.println(ornekMap); // {a=3, b=5, k=2, r=1}

System.out.println(ornekMap.ceilingEntry( key: "b")); // b=5
System.out.println(ornekMap.ceilingEntry( key: "c")); // k=2
System.out.println(ornekMap.ceilingEntry( key: "m")); // r=1
System.out.println(ornekMap.ceilingEntry( key: "t")); // null

System.out.println(ornekMap.ceilingKey("b")); // b
System.out.println(ornekMap.ceilingKey("c")); // k
```

treeMapismi.ceilingKey(key) : verilen key treeMap'de varsa, o key'i, verilen key treeMap'de yoksa verilen key'den büyük olan ilk key'i döndürür.



# Tree Maps

Tree Map elementlerini key'e gore dogal sirali olarak dizdigi icin, ilk ve son elementlere ulasilabilir.

treeMapismi.pollFirstEntry( ) :

treeMapismi.pollLastEntry( ) : istenen entry'yi silip, bize döndürür.

```
System.out.println(ornekMap); // {a=3, b=5, k=2, r=1}
System.out.println(ornekMap.firstKey()); // a
System.out.println(ornekMap.firstEntry()); // a=3

System.out.println(ornekMap.lastKey()); // r
System.out.println(ornekMap.lastEntry()); // r=1
```

```
System.out.println(ornekMap.pollFirstEntry()); // a=3 silip getirdi
System.out.println(ornekMap.pollLastEntry()); // r=1 silip getirdi
System.out.println(ornekMap); // {b=5, k=2}
```

```
System.out.println(ornekMap.descendingMap()); // {k=2, b=5}
System.out.println(ornekMap); // {b=5, k=2}
```

treeMapismi.descendingMap( ) : gecici olarak tree map'i descending siralar.



# Maps

treeMapismi.headMap(key) : verilen key exclusive olarak oncesindeki entry'leri verir  
treeMapismi.headMap(key, true) : verilen key inclusive olarak oncesindeki entry'leri verir

```
System.out.println(ornekMap); // {b=5, k=2, t=2, y=3}

System.out.println(ornekMap.headMap( toKey: "m")); // {b=5, k=2}
System.out.println(ornekMap.headMap( toKey: "t")); // {b=5, k=2}
System.out.println(ornekMap.headMap( toKey: "t", inclusive: true)); // {b=5, k=2, t=2}

System.out.println(ornekMap.higherKey("k")); // t
System.out.println(ornekMap.higherEntry( key: "t")); // y=3

System.out.println(ornekMap.lowerEntry( key: "m")); // k=2
System.out.println(ornekMap.lowerKey("c")); // b
```

treeMapismi.higherKey(key) : Verilen key'den buyuk olan ilk key'i döndürür.  
treeMapismi.higherEntry(key) : Verilen key'den buyuk olan ilk key'e ait entry'yi döndürür.  
treeMapismi.lowerKey(key) : Verilen key'den kucuk olan ilk key'i döndürür.  
treeMapismi.lowerEntry(key) : Verilen key'den kucuk olan ilk key'e ait entry'yi döndürür.

# The END

EN İYİ KOD ...  
CALISAN KOD'DUR.

ONDAN DA İYİSİ...  
CALISAN VE ANLASILIR KOD YAZMAKTIR.



OMUR BOYU YUZUNUZDEKI GULUMSEME EKSIK OLMASIN...