

Bansel Somergül

07.07.2023

Istanbul

12i Systems

Summer Internship



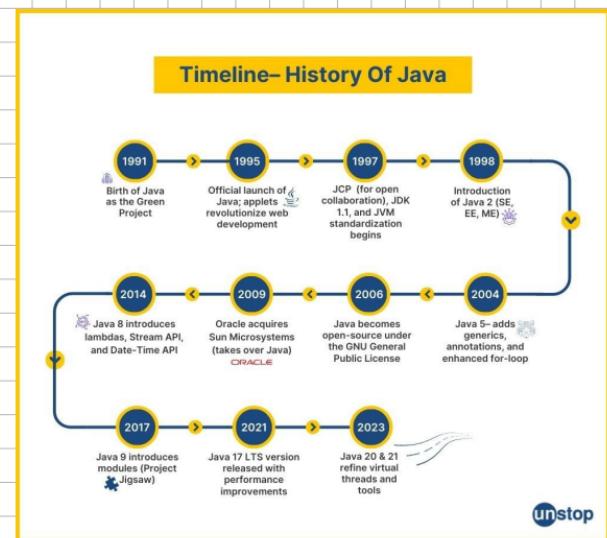
Java™

Java is a high-level, general-purpose, memory-safe, object-oriented programming language. It is intended to let programmers write once, run anywhere (WORA),^[18] meaning that compiled Java code can run on all platforms that support Java without the need to recompile.^[19] Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages.

Java gained popularity shortly after its release, and has been a popular programming language since then.^[20] Java was the third most popular programming language in 2022 according to GitHub.^[21] Although still widely popular, there has been a gradual decline in use of Java in recent years with other languages using JVM gaining popularity.^[22]

Java was designed by James Gosling at Sun Microsystems. It was released in May 1995 as a core component of Sun's Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GPL-2.0-only license. Oracle, which bought Sun in 2010, offers its own HotSpot Java Virtual Machine. However, the official

Java	
Paradigm	Multi-paradigm: generic, object-oriented (class-based), functional, imperative, reflective, concurrent
Designed by	James Gosling
Developer	Oracle Corporation
First appeared	May 23, 1995; 30 years ago ^[1]
Stable release	Java SE 24 ^{[2][3]} ✓ / 18 March 2025; 3 months ago
Typing discipline	Static, strong, safe, normative, manifest
Memory management	Automatic garbage collection
Filename extensions	.java, .class, .jar, .jmod, .war
Website	oracle.com/java/ ↗ java.com ↗ dev.java/ ↗
Influenced by	GLU ^[4] Simula ^[5] Lisp ^[4] Smalltalk ^[4] Ada 83, C++, ^[15] C# ^[16] Eiffel ^[7] Mesa ^[6] Modula-3, ^[9] Oberon, ^[10] Objective-C, ^[11] UCSD Pascal ^[12] , ^[13] Object Pascal ^[14]
Influenced	Ada 2005, ARKTS, BeanShell, C#, Chapel, ^[15] Clojure, ECMAScript, Fantom, Gambas, ^[16] Groovy, Hack, ^[17] Haxe, J#, JavaScript, JS++, Kotlin, PHP, Python, Scala, Seed7, Vals
Java Programming at Wikibooks	



Hangi IDE'ye tutturancağız

3'üncü de tutturancağız cuih
ide'ler belli tip uygulalar
geliştirmek için çok işe yarılır

IntelliJ Idea

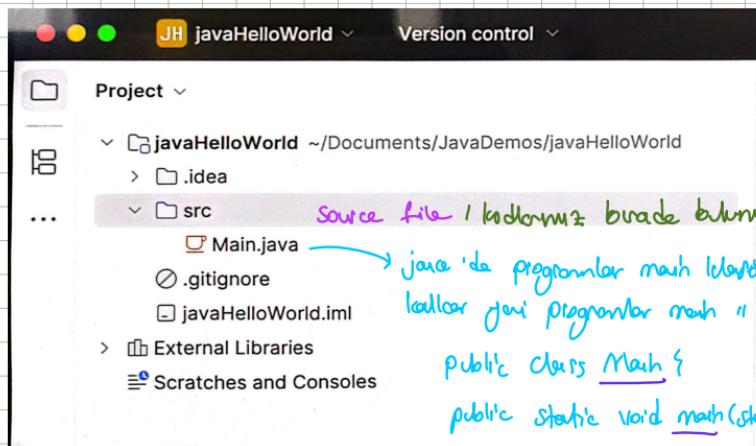
• Kullan gelistirme

Netbeans

• Swing gelistirme
cuih en iyi ide

Eclipse

• Kurucu uygulalar
cuih



Source file / kodlannız birade bulunur (kayıtlı)

Java'de programlar main metodundan başlar
kullanıcı programlar main'i başlatırlar

```
public class Main {  
    public static void main(String[] args) {  
        // Write code here  
    }  
}
```

Main class'indeki
main metodu

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

intellisense

3
3

main structure
for Java

programlama yapmayı
yapmak için bu.
isaret tutturır

```
System.out.println("Hello Java");
```

→ etenice bir sententile
Java yazıncaı doğrular

line denote

→ Java, case sensitive'dır. Jan buyrun - Jancale farklı bir dildir.

II Variables

Degiskenler data tuttugunu işgorlardır. Bu işgorlарın sağesinde bellekte tutulan değişkenlerin adresleri bilinçde㈠ olsalar da, bu değişkenler ile ulaşılabilirliğiniz.

Ayrıca değişkenler sağesinde, bir dosya programının bir yerinde bir değişken bir kere atayıp sonradan o dosya içerisinde kullanılabilirliğiniza

- Değişkenlerinizi bilgilere **reusability** sağlar
- dataları değiştirdiğinizde bilgilere tek bir tekrar kullanılabilirlikle ictin değişkenleri güncelleriz
ama ana nedeniniz bellekten fazla ramda kullanılmaz

II Data Types

- Değişken türmlerken, değişkenin tuttuğu verinin tipini belirtmeniz gerekmektedir.
Cümlü: Java, type sensitivity bir dilidir.

Primitive Types

type name	kind of value	memory used	size range
boolean	true or false	1 byte	not applicable
char	single character (Unicode)	2 bytes	all Unicode characters
byte	integer	1 byte	-128 to 127
short	integer	2 bytes	-32768 to 32767
int	integer	4 bytes	-2147483648 to 2147483647
long	integer	8 bytes	-9223372036854775808 to 9223372036854775807
float	floating-point number	4 bytes	$-3.40282347 \times 10^{38}$ to $-1.40239846 \times 10^{-45}$
double	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

→ modern long data tipi var icin en çok kullanılır olanlar, bit neden
int, byte, short vb. ile yaygın mı? Her segi long türündedirken short
mi?

- Cihazın ve donanım açısından bir sonraki önemdeki en önemli
farkı "memory size" kılavuzunda ortaya çıkar:

byte	1 byte
short	2 byte
int	4 byte
long	8 byte

geneldeki üzere $\times 2$ 'lik
bir artışı mevcuttur

// Switch - Case

- genelde if iki kod parçasını farklılıklarını sınıtlamak için kullanılır
- program dallandırma amacıyla

switch (condition) {

case : _____
 break;

default : _____

}

- Varsayılmaktır
larda direkt mola kullan

ctrl + shift + alt + 1

// Loops

Bir programda, birbirine benzeyen ifadeler / genel ifadeleme teknikleri kullanılarak
yazılan işlemleri döngüler denir.

→ for → while → do-while → foreach

for (int baslangic ; bitis ; artim)

{

 // kodlar burası

}

while (dengesi saglanma)

{

 // kodlar burası

 // koşul artırmayı unutma

}

do {

 // kodlar burası

 // + kere koşul olurken döngü

} while (şart burası)

// Arrays

Benzersiz türdeki verilen bir arada tutmak istenilen eserlerin topluluğudur.

Data Type [] ArrayName = new Data Type [ArraySize]

for (String ogrenci : ogrenciler)
{

 // kodlar burası

}

II Multidimensional Arrays

Cüde boyutlu dizi ləndir.

`DataType[][] ArrayName = new DataType[size][size]`

II Stringler ile Çalışma

Stringler, vəzində karakter dizi ləndir. Buranın dələyi vətəndə belənən qeydiyi məntilişlər kəndə de gələnlərdir (indi's rəb'i).

- String mesaj = "Hello World!"
- mesaj.length → elemən sayıını verir
- mesaj.charAt() → girilen indexdəki eleməni verir
- mesaj.concat("Metin") → metin birləştirmə işləni yapar
çıktı: Hello World!Metin



→ bəzidə da bir
0 karakterdir.

* buradakı concat işləni sonuncuda orjinal string olun mesaj stringi
içinən deyilmesi

mesaj.concat("Metin") → Yeni bir string olustur, kubanələr iñən
başka deyiklərə attrak qəbulur

→ mesaj. startsWith(.) → parentet içindeki ifade ile başlıyor
bu ifadede oronacık ifade
" " ile oronur, '' ile değil | cuihü bir string ile
nw dyle barkar

bu ifadede oronacık ifade | cuihü bir karakter ile
" " ile oronur, '' ile değil | baslıyor durumunu sağlanır

→ mesaj. endsWith(.) → birne durumunu içindeler, mantık gibi

asagidaki ifade destination obdu, konuları daşıyıcı iken burası farklıdır

char [] karakterler = new char [5]

→ mesaj. getChars(srcBegin=0, srcEnd=5, karakterler, destBegin=0)

string içindelki belli
bir kuru almanıza
baglar

source
başlangıç
noktası

source
bitiş
noktası

dest
noktası

destination
başlangıç
noktası

5 dörtlü
editmet 0,1,2,3,4
5 tane

→ mesaj. indexOf(' ') : parentet içinde yazılanın kaçinci
(" ") karakter olduğuuhnun bulunduğu yer
indis bilgisini verir

→ mesaj. indexOf(' ') : istedeki ile beraber mantık one orone
(" ") en sağdan başlar

* her iliskide, oronan ifadesi bulduğu ilk end sonucu verir
ve oronayı bıralar.

→ mesaj. replace(" ", " ") : parentet içindelki ifadesi mesaj
değişkenin eski degeri ile değiştirir

oldchar: eski ifade veya target:

newchar: yeni ifade

replacement:

bu islemlerin sonucu
yeni metin verir
original ifade degismez

↳ substring: gelen metin icinden bir parca almak istir

mesaj.substring(begIndex: 3, endIndex: 8) 3-4-5-6-7
bu indexler olur

↳ mesaj.split(regex: "-"): orjinal metni çift tırnak içindeki
dunne gibi parçalar

Geçerli döşemeler, jaziklerde
için for kullandıkları

↳ mesaj.toLowerCase(): mesajı küçük harfe çevirir

mesaj.toUpperCase(): " " büyük " "

↳ mesaj.trim(): stringin başındaki - sonundaki boşlukları siler

```
① Main.java ×
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5
6         Scanner input = new Scanner(System.in); // Scanner object
7         System.out.print("Please enter a number: ");
8         int number = input.nextInt(); // To get a number
9
10
11         boolean isPrime = true;
12
13         // Prime numbers bigger than 1
14         if (number == 1) {
15             isPrime = false;
16         }
17
18         // The only even prime number is 2
19         else if (number == 2) {
20             isPrime = true;
21         }
22
23         else {
24             for (int i = 2; i <= Math.sqrt(number); i++) {
25                 if (number % i == 0) {
26                     isPrime = false;
27                     break;
28                 }
29             }
30
31             if (isPrime) {
32                 System.out.println("Prime number");
33             } else {
34                 System.out.println("Not Prime number");
35             }
36
37             input.close(); // Closing scanner object
38         }
39     }
40 }
```

əzələcəyi mi deyil mi?

Main.java

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5
6         Scanner input = new Scanner(System.in);
7
8         System.out.print("Please Enter a Character = ");
9         char character = input.next().charAt(0);
10
11         // I want to keep code simple. So i didn't check other characters like i,i etc.
12         // And also i didn't use try-catch to keep code simple and just learn essentials
13         if (character == 'a' || character == 'e' || character == 'o' || character == 'u') {
14             System.out.println("Girilen karakter sessli harftir");
15         }
16
17         else{
18             System.out.println("Girilen karakter sessiz harftir");
19         }
20
21         input.close();
22     }
23 }
24 }
```

Main.java

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5
6         // In number theory, a perfect number is a positive integer that is equal to the sum of its positive proper divisors,
7         // that is, divisors excluding the number itself. For instance, 6 has proper divisors 1, 2 and 3, and 1 + 2 + 3 = 6, so 6 is a perfect number.
8
9         Scanner input = new Scanner(System.in);
10        System.out.print("Please enter a number = ");
11        int number = input.nextInt();
12
13        int sum = 0;
14
15        for (int i = 1; i <= (number / 2); i++) {
16            if (number % i == 0) {
17                sum += i;
18            }
19        }
20
21        if (sum == number) {
22            System.out.println("The number is a perfect number ");
23        }
24
25        else {
26            System.out.println("The number is not a perfect number ");
27        }
28    }
29 }
```

```
1 import java.util.Scanner;
2 
3 public class Main {
4     public static void main(String[] args) {
5 
6         // Two numbers whose positive integer divisors, excluding themselves, are equal are called amicable numbers.
7 
8         Scanner input = new Scanner(System.in);
9         System.out.print("Please Enter First Number = ");
10        int firstNumber = input.nextInt();
11        System.out.print("Please Enter Second Number = ");
12        int secondNumber = input.nextInt();
13 
14        int sum1 = 0;
15        int sum2 = 0;
16 
17        for (int i = 1; i <= (firstNumber/2); i++) {
18            if (firstNumber % i == 0) {
19                sum1 += i;
20            }
21        }
22 
23        for (int i = 1; i <= (secondNumber/2); i++) {
24            if (secondNumber % i == 0) {
25                sum2 += i;
26            }
27        }
28 
29        if (sum1 == secondNumber && sum2 == firstNumber) {
30            System.out.println(firstNumber + " and " + secondNumber + " are friend number.");
31        }
32 
33        else {
34            System.out.println(firstNumber + " and " + secondNumber + " are not friend number.");
35        }
36 
37        input.close();
38    }
39 }
```

```
1 import java.util.Scanner;
2 
3 public class Main {
4     public static void main(String[] args) {
5 
6         int[] sayilar = new int[6];
7 
8         Scanner input = new Scanner(System.in);
9 
10        for (int i = 0; i < sayilar.length; i++) {
11            System.out.print("Please Enter " + (i+1) + ". Number: ");
12            sayilar[i] = input.nextInt();
13        }
14 
15        int aranacakSayi;
16        System.out.print("Please Enter Searching Number: ");
17        aranacakSayi = input.nextInt();
18 
19        boolean bulduduMu = false;
20 
21        for (int i = 0; i < sayilar.length ; i++) {
22            if (sayilar[i] == aranacakSayi) {
23                System.out.println("The Number is at " + i + ". index");
24                bulduduMu = true;
25            }
26        }
27 
28        if (!bulduduMu) {
29            System.out.println("The Number is not in the array");
30        }
31 
32        input.close();
33 
34    }
35 }
```

II Metodlarla Çalışma

Programlamada "Don't Repeat Yourself" prensibi mercattir. Yani

'Kendini tekrar etme' denilir. Yani, sonlu sayıda kod + kere
kullanmak istiyacını oluşturmak yerine!

Bu nedenle da "fonksiyonel programlama" mantığı derneğe gider. Bu
bir fonksiyonun görevi de aynı dikkat etmektedir.

→ Bir fonksiyonun görevi bir işlemi yapmak şeklinde yazılıp
gerekir. Cunku bir fonksiyon içeriğinde birden fazla işlem olursa ve
bu işlerden sadexe bir tanesini kullanmak istesek tüm parçası
kullanmamız gereklidir. Sadexe o kod parçasını kullanma sənədli

Oluşur. Yani kodu ve öz olsalı:

Bir fonksiyon → bir işlem

* main yapınız da bir method'dur ve jax'a de ille bize matn
(yazılı) verir.

Main.java x

```
1 import java.util.Scanner;  
2  
3 public class Main {  
4     public static void main(String[] args) {  
5         sayiBulmaca(); → soyi Bulmaca adında  
6         bir fonksiyon  
7     }  
8 }
```

10 public static void sayiBulmaca() { 1 usage }

fonsiyonun scope'u içinde de
kodlar mercattir

* soyi Bulmaca fonsiyonu
1 kere yazıldı. Article
istedigimiz zaman her
defter istedigimiz
zaman aqroabilidir.

→ main'den aqroabilen fonsiyon
fonsiyon istenildiğinde
icin carnel case
notasyonu =

public static void mesajVer(int aranacakSayi, int[] sayilar) {

fonksiyon özellikleri

fonsiyon
adı

Fonksiyon'a gelecek
olarak parametreler

Main.java ×

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5
6         // Call the function
7         sayiBulmaca();
8
9     }
10
11     // sayiBulmaca function
12     public static void sayiBulmaca() { 1 usage
13
14         int[] sayilar = new int[6];
15
16         Scanner input = new Scanner(System.in);
17
18         for (int i = 0; i < sayilar.length; i++) {
19             System.out.print("Please Enter " + (i+1) + ". Number: ");
20             sayilar[i] = input.nextInt();
21         }
22
23         int aranacakSayi;
24         System.out.print("Please Enter Searching Number: ");
25         aranacakSayi = input.nextInt();
26
27         mesajVer(aranacakSayi, sayilar);
28
29         input.close();
30     }
31
32     @
33     public static void mesajVer(int aranacakSayi, int[] sayilar) { 1 usage
34         boolean bulunduMu = false;
35
36         for (int i = 0; i < sayilar.length ; i++) {
37             if (sayilar[i] == aranacakSayi) {
38                 System.out.println("The Number is at " + i + ". index");
39                 bulunduMu = true;
40             }
41
42             if (!bulduMu) {
43                 System.out.println("The Number is not in the array");
44             }
45         }
46     }
}
```

* void metodlar, genye deger donduran jepibirdir

void metodlar, islen gereklilikte icin, bir enin genye getirmek icin vb. islen bozlu kullanilar.

→ void denebili fonksiyonlar genye deger donduran fonksiyonlardır. Bu fonksiyonlar icin return ifadesi kullanılır.

```
public static void guncelle() { no usages  
    System.out.println("Güncellendi");  
}
```

```
// a return function  
public static int topla(int a, int b) { 1 usage  
    return a + b;  
}
```

parametrelər
fonksiyon tipi ile return tipi
ayri olmala zənəbdür

Variable Arguments

fonksiyonlarda parametre olmali istedigimiz kəndar parametre göndərmək üçün variable arguments kullanılır

```
public static int topla( int... sayilar )
```

variable arguments füsmi,
bir nevi int array göndərilməsi gibi olur

```
// a return function  
public static int topla(int a, int b) { 1 usage  
    return a + b;  
}  
  
// variable arguments  
public static int topla(int... sayilar) { no usages  
    int toplam = 0;  
    for (int sayı : sayilar) {  
        toplam += sayı;  
    }  
    return toplam;  
}
```

döni təzələmə fəxli

ayrı 'simde ora fəxli
iñi fonksiyon cunus' parametre sayilar fəxli

↳ variable arguments yapısı, parametre olarak ne gönderdiğimde kullanımda poteri kodu okuyabiliş olsun gibi içim test edilebilirliği oluşturmaktaadır

II class'lar ile Çalışma

Java, C#, C++ gibi dillerde nesne oluşturmak için bir sınıf ile örneklendirilebilir. Nesne oluşturmak için sınıfın içindeki metodları kullanır.

Sınırlı tutarlılığı: herhangi bir sınıfın içindeki metodları kullanır.

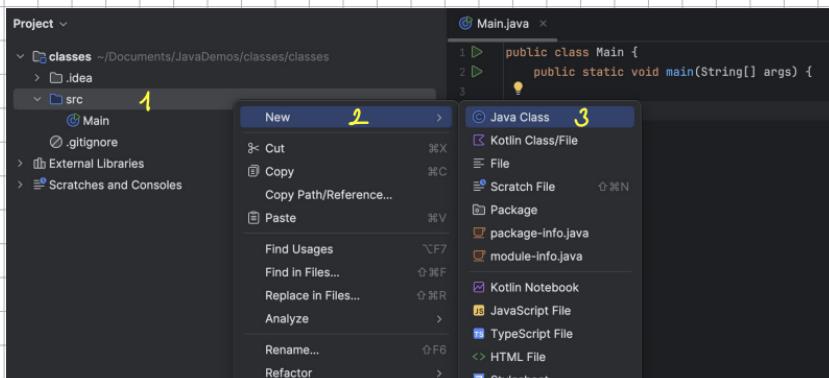
JAVA'da var olan sınımları, genelde metodları da kullanır bir sınıfın içinde kullanır.

• Bize en çok yardım eden bir class içinde bulduğumuz

```
public class Main { → main class'i içinde çağrılmış  
    public static void main (String[] args) {  
        }  
    }
```

• JAVADA her class 'tir / nesne 'dir

1. class'ların ilk ve temel özelliğinin genel olduğu



```

Main.java x CustomerManager.java → tımladığınız class
1 public class Main {
2     public static void main(String[] args) {
3
4         CustomerManager customerManager = new CustomerManager(); → class'ından object
5         bu class'tan bu isminde nesne şırt üretmektedir
6     }
7 }

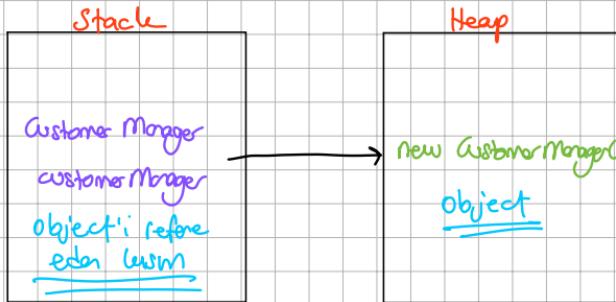
```

Yani burada class'ının bir örneğinin üretilmesiyle yani class'ının
bir object üretilmektedir.

Artık bu nesnenin kullanıldığı class'ının işlevini açıklabiliyoruz.

→ class'lar referans tipleridir

- bilgişayar belleği gibi ana blokuna gider:



```

Main.java x CustomerManager.java
1 public class Main {
2     public static void main(String[] args) {
3
4         // reference type
5         CustomerManager customerManager = new CustomerManager();
6         customerManager.Add();
7         customerManager.Remove();
8         customerManager.Update();
9
10
11         CustomerManager customerManager2 = new CustomerManager();
12         customerManager = customerManager2; // both object show same thing
13
14     }
}

```

Hedef referansı oluyan
Atesneler Garbage Collector
Jepisi seyfesinde
silmek

* Programlarda kullanığınız dört tipi iki şekilde dir.

1. Value type

- Stack'te leşmekte
- byte, short, int, long
- float, double
- char
- boolean

2. Reference type

- Stack'ta heap'te leşmekte
- string
- scanner, arraylist
- object
- interface
- array, enum

► Java'da veriler bellekte saklanma ve değişkenlere atama şekillerine göre temel olarak ikiye ayrılır: değer tipleri (value types) ve referans tipleri (reference types). Bu ayrılm, değişkenlerin nasıl davranışlarını ve bellekte nasıl yönetildiğini anlamak için çok önemlidir.

Değer Tipleri (Value Types) / İlkel Veri Tipleri (Primitive Data Types)

Java'da değer tipleri genellikle **ilkel veri tipleri (primitive data types)** olarak bilinir. Bu tipler, verinin kendisini doğrudan saklar. Yani bir değişken bir ilkel türde sahip olduğunda, o değişkenin bellekteki konumu doğrudan verinin değerini içerir.

Özellikleri:

- **Bellekte Doğrudan Değer Saklar:** Değişkenin bellekteki yeri, doğrudan o değerin kendisidir.
- **Sabit Boyut:** Her ilkel tipin bellekte kapladığı yer sabittir.
- **Stack Bellekte Saklanır:** Genellikle **stack (yığın) bellek** alanında saklanırlar. Bu, daha hızlı erişim sağlar.
- **null Değer Alamazlar:** İlkel tipler `null` (boş) değer alamazlar. Her zaman varsayılan bir başlangıç değerine sahiptirler (örneğin, sayılar için `0`, boolean için `false`).
- **Atama Yaptığınızda Kopyalama Olur:** Bir ilkel tip değişkenini başka bir ilkel tip değişkenine atadığınızda, değerin bir kopyası oluşturulur. Yani orijinal değişkenle yeni değişkenin bağımsız kopyaları olur. Birini değiştirmek diğerini etkilemez.

Java'daki İlkel Veri Tipleri (8 Tane):

1. Tam Sayılar:

- `byte` (8-bit)
- `short` (16-bit)
- `int` (32-bit, en sık kullanılan)
- `long` (64-bit)

2. Ondalık Sayılar (Kayan Noktalı):

- `float` (32-bit)
- `double` (64-bit, varsayılan ve daha yaygın)

3. Karakterler:

- `char` (16-bit Unicode karakter)

4. Mantıksal Değerler:

- `boolean` (`true` veya `false`)

Örnek:

```
Java

int sayi1 = 10;
int sayi2 = sayi1; // sayi1'in değeri olan 10, sayi2'ye kopulanır

sayi2 = 20; // sayi2'yı değiştirmek sayi1'i etkilemez

System.out.println("sayi1: " + sayi1); // Çıktı: sayi1: 10
System.out.println("sayi2: " + sayi2); // Çıktı: sayi2: 20
```

Referans Tipleri (Reference Types)

Referans tipleri, verinin kendisini doğrudan saklamazlar. Bunun yerine, verinin (yani nesnenin) bellekteki konumuna bir **referans (bellek adresi)** tutarlar. Asıl veri (nesne) **heap (yükün) bellek** alanında saklanır.

Özellikleri:

- **Bellekte Referans Saklar:** Değişkenin kendisi, objenin bellekteki adresini (referansını) tutar. Objenin verileri ise o adreste bulunur.

- **Heap Bellekte Saklanır:** Referans tiplerinin asıl verileri (objeleri) **heap bellek** alanında oluşturulur ve saklanır. Java'nın çöp toplayıcısı (Garbage Collector) bu alanı yönetir.
- **null Değer Alabilirler:** Bir referans tipi değişken, hiçbir objeyi işaret etmiyorsa `null` değerini alabilir.
- **Atama Yaptığınızda Referans Kopyalanır:** Bir referans tipi değişkeni başka bir referans tipi değişkenine atadığınızda, objenin kendisi kopyalanmaz; sadece objeye olan **referans (bellek adresi)** kopyalanır. Bu durumda, her iki değişken de bellekteki **aynı objeyi** işaret eder. Bir değişken üzerinden objede yapılan değişiklikler, diğer değişken üzerinden de görünür olur.

Java'daki Referans Tipleri:

- **Sınıflar (Classes):** `String`, `Scanner`, `ArrayList`, `Object` ve kullanıcı tanımlı tüm sınıflar (`Person`, `Car` vb.).
- **Arayüzler (Interfaces):**
- **Diziler (Arrays):** lüklü tipleri veya referans tipleri barındıran tüm diziler (`int[]`, `String[]`, `Person[]` vb.).
- **Enum'lar (Enumerations):**

Örnek:

```
Java

String metin1 = "Merhaba"; // metin1, "Merhaba" objesinin referansını tutar
String metin2 = metin1; // metin2 de aynı "Merhaba" objesinin referansını tutar

// Not: Stringler Java'da 'immutable' (değişmez) olduğu için bu örnek biraz yanlıltır.
// String bir değer ataması aslında yeni bir obje yaratır.
// Daha iyi bir örnek için kendi sınıfımızı kullanalım:

class Kutup {
    int x;
    Kutup(int x) { this.x = x; }
}
```

```

public class Main {
    public static void main(String[] args) {
        Kutup nokta1 = new Kutup(10); // nokta1, heap'te yeni bir Kutup objesine referans eder
        Kutup nokta2 = nokta1; // nokta2 de AYNI Kutup objesine referans eder

        nokta2.x = 20; // nokta2 üzerinden objenin x değerini değiştirdik

        System.out.println("nokta1.x: " + nokta1.x); // Çıktı: nokta1.x: 20 (Çünkü
        System.out.println("nokta2.x: " + nokta2.x); // Çıktı: nokta2.x: 20
    }
}

```

Temel Fark Özeti:

Özellik	Değer Tipleri (İlkel Tipler)	Referans Tipleri
Ne Saklar?	Verinin kendisini	Verinin (objenin) bellek adresini
Bellek Alanı	Genellikle Stack	Heap
Boyut	Sabit	Değişkenye göre değişir (objenin boyutu)
null Değer?	Alamaz	Alabilir
Atama Davranışı	Değer kopyalama (bağımsız kopyalar)	Referans kopyalama (aynı objeyi işaret eder)
Örnekler	int, boolean, double, char vb.	String, Array, Sınıf objeleri, Scanner vb.

Export to Sheets

Encapsulation

• Field ve Attribute

Class'lar ortak操作larını takip etmesinin yanı sıra bir diğer özelliti de "özellik" formasyonu. Bu özellitler 'field' ve 'attribute' olarak isimlendirilir

• data tutan yapıllerdir
• Variables'lerdir bir nesne

* Yazılım geliştiricilerin SOLID prensibi önemlidir. Bu prenziye iliskide deghizilebilir

```

Main.java  ProductManager.java  Product.java
1 public class Main {
2     public static void main(String[] args) {
3         Product product = new Product();
4         product.name = "Laptop";
5         product.price = 36980;
6         product.description = "Asus ROG Strix 166";
7         product.id = 1;
8         product.stockAmount = 3;
9         System.out.println(product.name);
10
11     ProductManager productManager = new ProductManager();
12     productManager.Add(product);
13 }
14 }
```

```

Main.java  ProductManager.java  Product.java
1 public class ProductManager { no usages
2     public void Add(Product product) { // sending product object as a parameter no usages
3         // JDBC
4         System.out.println("Product Added " + product.name);
5     }
6 }
```

```

Main.java  ProductManager.java  Product.java
1 public class Product { no usages
2     // attributes veya field
3     int id; no usages
4     String name; no usages
5     String description; no usages
6     double price; no usages
7     int stockAmount; no usages
8 }
```

Java'da Encapsulation (Kapsülleme) Nedir?

Encapsulation (Kapsülleme), nesne yönelimli programmanın (OOP) dört temel prensibinden biridir. Temel olarak, veriyi (nitelikler/özellikler) ve o veri üzerinde işlem yapan metodları (davranıslar) bir araya getirme ve bu verİYE dışardan doğrudan erişimi kısıtlama sürecidir. En basit ifadeyle, bir nesnenin iç çalışma mekanizmasını dış dünyadan gizlemek ve sadece belirlenmiş arayüzler (metotlar) aracılığıyla erişim sağlamak.

Bir hapi veya kapsülü düşünebilirsiniz. Kapsülü içinde farklı maddeler bulunur ama siz sadece dışındaki kapsülü görür ve yutarsınız. İçindeki maddelerin ne olduğunu, nasıl çalıştığını bilmenize veya doğrudan onlara dokunmanız gereklidir. Encapsulation da benzer bir mantıkla çalışır.

Encapsulation'ın Temel Bileşenleri ve Nasıl Uygulanır?

Java'da encapsulation genellikle şu şekilde uygulanır:

1. Nitelikleri (`private`) Yapmak:

- Bir sınıfın verilerini (fields/attributes) doğrudan dışardan erişilemez hale getirmek için `private` erişim belirleyicisi kullanılır. `private` olarak tanımlanan bir niteliğe yalnızca aynı sınıfından erişilebilir. Bu, dışardan doğrudan müdahaleyi engeller ve verinin bütünlüğünü korur.

2. `public Getirici (Getter)` ve `Ayarlayıcı (Setter)` Metotları Kullanmak:

- `private` olarak tanımlanmış verilere dışardan kontrollü bir şekilde erişim sağlamak ve onları değiştirmek için `public` metotlar (genellikle getter ve setter metotları olarak adlandırılır) yazarısınız.
- `Getter Metotları (get...)`: Bir niteliğin değerini okumak (almak) için kullanılır.
- `Setter Metotları (set...)`: Bir niteliğin değerini değiştirmek (ayarlamak) için kullanılır. Bu metotlar içinde veri doğrulama (validation) ve iş kuralları gibi mantıklar da eklenebilir.

- Encapsulation, kullanıcıyı herhangi bir alanı kullanmayı kısıtlıyor
- Bir object'in iç contente ulaşırırmamızı dış dünyadan gizlemek ve sadece belirlenmiş metodlar aracılığıyla erişim sağlayacaktır.

Neden Encapsulation Kullanılır? (Faydalari)

1. Veri Gizleme (Data Hiding):

- Nesnenin iç durumunun dış dünyadan gizlenmesini sağlar. Kullanıcılar (diğer kod parçaları), bir nesnenin iç detaylarını bilmek sorunda kalmazlar, sadece o nesnenin sunduğu arayüzü (metotları) kullanırlar.

2. Veri Bütünlüğü ve Güvenliği:

- Niteliklere doğrudan erişimi engellediği için, verilerin yanlış veya geçersiz bir duruma düşmesini onler. Setter metotları aracılığıyla veri girişi kontrol edilebilir ve doğrulanabilir. Örneğin, bir yaş değerinin negatif olmasına engellebilirsiniz.

3. Esneklik ve Bakım Kolaylığı:

- Bir sınıfın iç yapısını değiştirdiğinizde (örneğin, bir niteliğin adını veya veri tipini değiştirdiğinizde), bu değişiklik dış kodu etkilemez. Çünkü dış kod sadexe getter/setter metodlarını kullanır, niteliğin kendisine bağımlı değildir. Bu da kodun bakımını ve evrimini kolaylaştırır.

4. Kullanım Kolaylığı (Usability):

- Sınıfın karmaşık iç yapısını dışarıya açmak yerine, sadece anlaşılır ve amaca yönelik metotlar sunar. Bu, sınıfı kullanan geliştiriciler için daha basit ve hata yapmaya daha az açık bir arayüz sağlar.

```
public class Ogrenci {  
    // 1. Nitelikler 'private' olarak tanımlanır (veri gizleme)  
    private String ad;  
    private int yas;  
    private String ogrenciNo;  
  
    // Kurucu metod (Constructor) - Nesne oluşturulurken başlangıç değerlerini ayarlar  
    public Ogrenci(String ad, int yas, String ogrenciNo) {  
        this.ad = ad;  
        // Setter metodu ile yaşı doğrulaması yapılmıştır  
        setYasi(yas); // Constructor içinde de setter çağırılabilir  
        this.ogrenciNo = ogrenciNo;  
    }  
  
    // 2. 'public' Getter Metotları - Niteliklerin değerlerini okumak için  
    public String getAd() {  
        return ad;  
    }  
  
    public int getYas() {  
        return yas;  
    }  
  
    public String getOgrenciNo() {  
        return ogrenciNo;  
    }  
  
    // 3. 'public' Setter Metotları - Niteliklerin değerlerini güvenli bir şekilde değiştirmek için  
    public void setAd(String ad) {  
        // İsteğe bağlı olarak burada veri doğrulama yapılabilir  
        if (ad != null && ad.trim().isEmpty()) {  
            this.ad = ad;  
        } else {  
            System.out.println("Hata: Ad boş olamaz!");  
        }  
    }  
  
    public void setYasi(int yas) {  
        // Yaş için veri doğrulama Örneği: Yaş 0 ile 120 arasında olmalı  
        if (yas > 0 && yas <= 120) {  
            this.yas = yas;  
        } else {  
            System.out.println("Hata: Yaş 0 ile 120 arasında olmalıdır!");  
        }  
    }  
  
    // OgrenciNo genellikle değişmez, bu yüzden sadece getter ekleyebiliriz (setter eklemeyebiliriz)  
    // Eğer bir niteliğin sadece okunmasını istiyorsak, sadece getter ekleriz.  
    // Eğer dışarıdan erişimini tamamen engellemek istiyorsak, ne getter ne de setter eklemeyiz.  
}
```

Kullanım Örneği:

```
Java

public class Main {
    public static void main(String[] args) {
        Ogrenci ogrenci1 = new Ogrenci("Aysel Yilmaz", 20, "12345");

        // Veriye doğrudan erişim yok (çünkü 'private')
        // System.out.println(ogrenci1.ad); // Hata verir!

        // Getter metotları ile verilere güvenli erişim
        System.out.println("Ögrenci Adı: " + ogrenci1.getAd());
        System.out.println("Ögrenci Yaşı: " + ogrenci1.getYas());

        // Setter metotları ile veriyi güvenli bir şekilde değiştirme
        ogrenci1.setYas(22); // Geçerli bir yaşı
        System.out.println("Yeni Yaşı: " + ogrenci1.getYas());

        ogrenci1.setYas(-5); // Geçersiz yaşı, setter hata mesajı verir ve yaşı deg
        System.out.println("Son Yaşı: " + ogrenci1.getYas()); // Hala 22
    }
}
```

Encapsulation, kodunuzu daha modüler, güvenli ve bakımı kolay hale getiren temel bir OOP prensibidir. Java'da genellikle getter ve setter metotları aracılığıyla uygulanır.

Main.java ProductManager.java Product.java

```
1  public class Product { 3 usages
2  	// attributes veya field
3  	// tüm bu attribute'ler public'tır ve her yerden erişilebilir
4  	// ama encapsulation gereği bunlar private yapılmıştır
5  	// int id;
6  	// String name;
7  	// String description;
8  	// double price;
9  	// int stockAmount;
10
11  // private sadece tanımlandığı sınıfta geçerlidir
12  private boolean onSale; no usages
13
14  // field'ları - işte böyle
15  private int _id; 1
16  private String _name;
17  private String _description;
18  private double _price;
19  private int _stockAmount;
20
21  public int getId() {
22      return _id;
23  }
24
25  // set edilecek de
26  public void setId(int id) {
27      _id = id; // türkçe
28      // yani burada
29  }
30
31 }
```

Show Context Actions

- 1 Paste
- Copy / Paste Special
- Column Selection Mode
- Find Usages
- Go To
- Folding
- Analyze
- Rename...
- Refactor **1**
- Generate...
- Toggle Field Watchpoint
- Open In
- Local History
- Git
- Compare with Clipboard
- Diagrams
- Create Gist...

2 Encapsulate Fields...

- Migrate Packages and Classes
- Invert Boolean...

16:23 LF UI

→ field'ları
encapsulation
metodlarını böyle
yerleştirmek bu
genellikle kullanılır