**Mündəricat**

1. INTRODUCTION
2. IF ELSE
3. SWITCH CASE
4. LOOP (While and Do While / For loop)
5. BREAK AND CONTUNIE
6. ARRAY
7. METHODS
8. OBJECTS
9. VARİABLE TYPES
10. String metodları
11. REKURSİV metodlar
12. ABSTACT CLASS AND ABSTRACT METHODS AND INTERFACES
13. Constructor
14. Override and Overload
15. OOPs (Object-Oriented Programming System)
16. ACCESS MODİFİERS
17. STACK və HEAP yaddaşlar
18. GABBİDGE COLLECTOR
19. THREAD AND MULTİTHREAD (Synchronization and volatile)
20. Generics in Java
21. Java Reflection
22. Annotation və custom Annotation
23. Widening > boxing > varargs
24. Enum
25. Java Collection
26. String, StringBuilder, StringBuffer
27. Exception try catch(checked , unchecked)
28. Exception throws, throw, try with resources & Customize Exception
29. MySQL
30. GitHub and BitBucket

ToString

Hashcode

Lambda

**Session and cookies**

**Sesiya serverdə saxlanılır.**

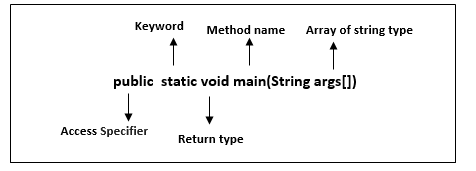
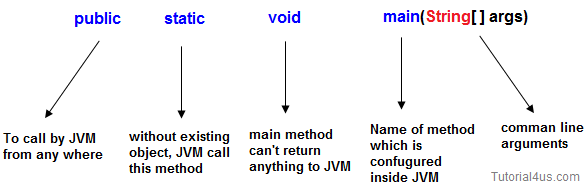
**Kukilər brouzerdə saxlanılır**

**INTRODUCTION**

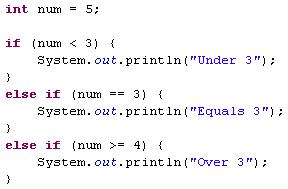
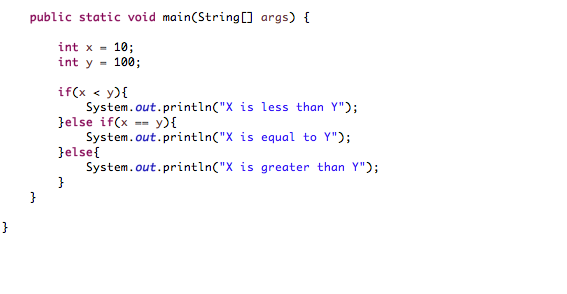
IDE – Integrated Development Environment / kod yazılması üçün platforma və ya editordu. (Eclipse / İntelij İdea / NetBeans and etc)

Java – 1995 may 23 Jams Gosling tərəfindən yaradılıb. (Oracle Corporation)

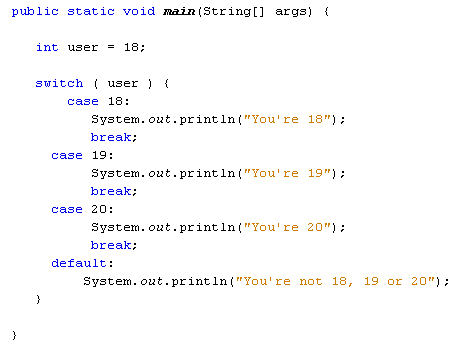
1. Java versiyasına baxmaq= CMD =java –version
2. Java Home quraşdırmaq üçün= echo %JAVA\_HOME%
3. Cd(change direction) cd yazıb arxasınca əməliyyatı edəcəyin ünvanı yaz.
4. CMD-də java compile etmək = javac Main.java = burada .class faylı yaranır. Daha sonra =java Main = enter və run edilir.
5. Javada Clasın içərisində System.out.println(“Hello” + arg[0]);
6. System.out.println(“Hello” + arg[1]);
7. Bunu CMD-də run etmək üçün = CMD = java Main Elton=enter
8. CMD= java Main Haldun
9. 3cü arqument yoxdursa o zaman dırnaq içində CMD= java Main “Nargiz” yazılmalıdır.



**IF ELSE**



**SWITCH CASE**

****

**LOOP (While and Do While / For loop)**

**//while**

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

**// do while**

int i = 0;  
do {

System.out.println(i);

i++;

}

while (i < 5);

**//For loop**

// Outer loop

for (int i = 1; i <= 2; i++) {

System.out.println("Outer: " + i); // Executes 2 times

// Inner loop

for (int j = 1; j <= 3; j++) {

System.out.println(" Inner: " + j); // Executes 6 times (2 \* 3)

}

}

**// for each loop**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

**BREAK AND CONTUNIE**

**//break**

for (int i = 0; i < 10; i++) {

if (i == 4) {

break;

}

System.out.println(i);

}

**//contunie**

for (int i = 0; i < 10; i++) {

if (i == 4) {

continue;

}

System.out.println(i);

}

**ARRAY**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

// Outputs Volvo

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

System.out.println(cars[0]);

// Now outputs Opel instead of Volvo

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);

// Outputs 4

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (int i = 0; i < cars.length; i++) {

System.out.println(cars[i]);

}

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

**// Multidimensional arrays**

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

System.out.println(myNumbers[1][2]); // Outputs 7

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

myNumbers[1][2] = 9;

System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7

public class Main {

public static void main(String[] args) {

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

for (int i = 0; i < myNumbers.length; ++i) {

for(int j = 0; j < myNumbers[i].length; ++j) {

System.out.println(myNumbers[i][j]);

}  
 }

}

}

**METHODS**

**Metodda void keywordü varsa return etmək olmur.**

public class Main {

static void myMethod() {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod();

myMethod();

myMethod();

}

}

// I just got executed!

// I just got executed!

// I just got executed!

**OBJECTS**

Javada obyektlər digər klasın metod funsiya və verilənlərindən istifadə etməyə imkan verir.

Animal animal = new Animal();

animal.name;

……

Soldakı animal referans adlanır.

Sağdakı new Animal obyekt adlanır.

**Qeyd:** Birbaşa metodsuz clasın daxilində yaradılan verilənlər instance variable adlanır. Metodun daxilində isə local variable adlanır.

**VARİABLE TYPES**

1. **public** **class** A
2. {
3. **static** **int** m=100;//static variable
4. **void** method()
5. {
6. **int** n=90;//local variable
7. }
8. **public** **static** **void** main(String args[])
9. {
10. **int** data=50;//instance variable
11. }
12. }//end of class

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Java Variable Example: Narrowing (Typecasting)

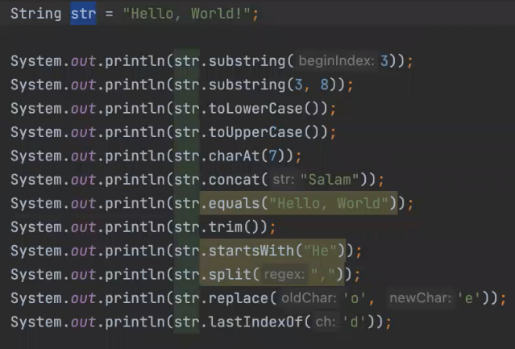
1. **public** **class** Simple{
2. **public** **static** **void** main(String[] args){
3. **float** f=10.5f;
4. //int a=f;//Compile time error
5. **int** a=(**int**)f;
6. System.out.println(f);
7. System.out.println(a);
8. }}

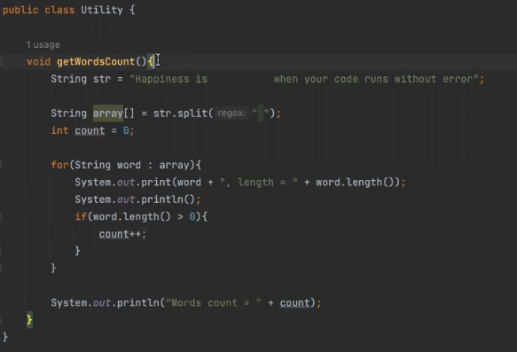
### Java Variable Example: Overflow

1. **class** Simple{
2. **public** **static** **void** main(String[] args){
3. //Overflow
4. **int** a=130;
5. **byte** b=(**byte**)a;
6. System.out.println(a);
7. System.out.println(b);
8. }}

### Java Variable Example: Adding Lower Type

1. **class** Simple{
2. **public** **static** **void** main(String[] args){
3. **byte** a=10;
4. **byte** b=10;
5. //byte c=a+b;//Compile Time Error: because a+b=20 will be int
6. **byte** c=(**byte**)(a+b);
7. System.out.println(c);
8. }}

**String metodları**

****

**REKURSİV metodlar**

1. **public** **class** RecursionExample3 {
2. **static** **int** factorial(**int** n){
3. **if** (n == 1)
4. **return** 1;
5. **else**
6. **return**(n \* factorial(n-1));
7. }
9. **public** **static** **void** main(String[] args) {
10. System.out.println("Factorial of 5 is: "+factorial(5));
11. }
12. }

**ABSTACT CLASS AND ABSTRACT METHODS AND INTERFACES**

Abstract classların obyekti yaradıla bilmir. Abstract classes are not instantiable.

Abstrakt klass final ola bilməz yəni final keyvordu azıla bilmez qarşısında. Daxilində Normal methodlar və abstract metodlar( public abstract void getModel(); ) yaradılır.

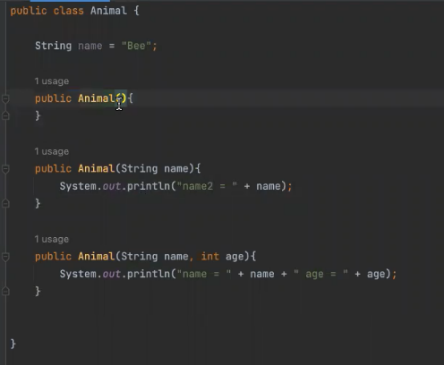
Digər classlar bu klasdan varislik alıb Override edir. Default class kimi də istifadə edilir.

Default metodunun qarşısında default static metodun qarşısında static keyvoru yazılmalıdır.

Interfeyslər – abstract classlar kimi obyekti yaradılmır. Lakin o halda obyekt yaradıla bilərik ki, mütləq override edilsin.

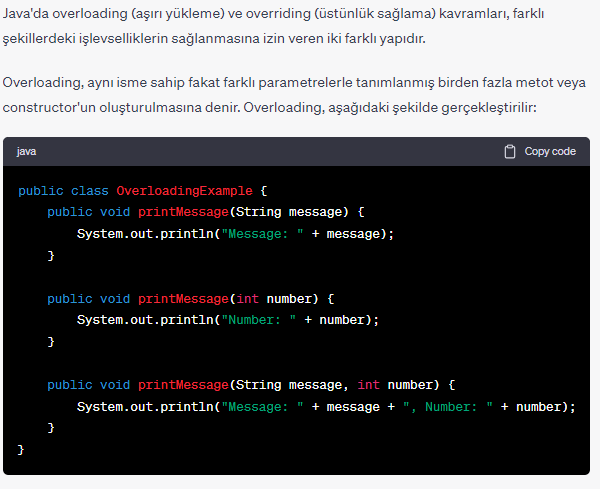
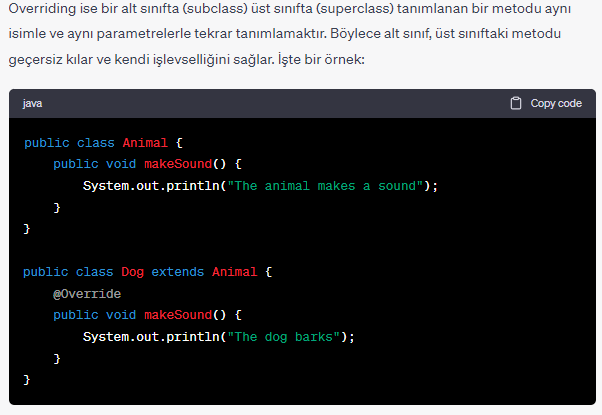
İnterfeys abstract klasdan fərqli olaraq bir clasa çox sayda interfeys implemets etmək olur.

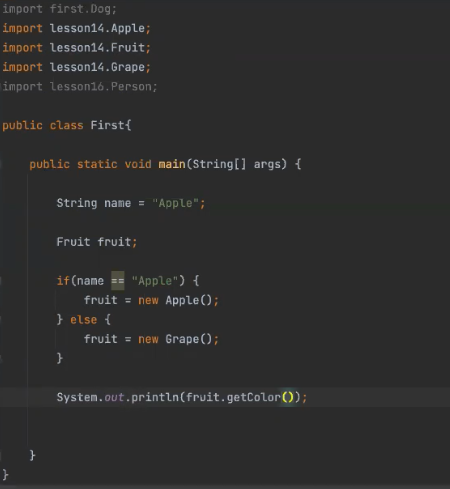
**Constructor**

**Clasın adı ilə eyni olan default metoda konstruktor deyilir. İstənilən sayda konstruktor yarada bilərik.**

****

**Override and Overload**

****

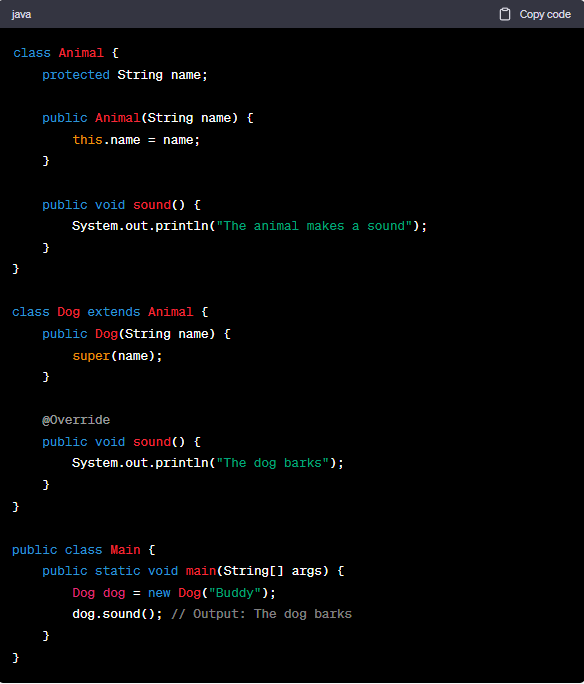
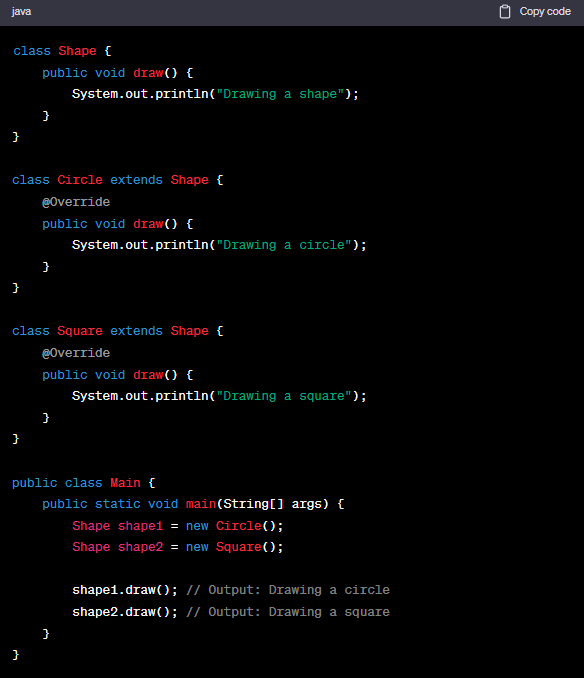
**Əgər dog klası animal klasdan varislik alıbsa o zaman upcasting edərək Animal animal = new Dog(); yaza bilərik. Və ya Downcasting Dog dog = (dog) animal yaza bilərik.**

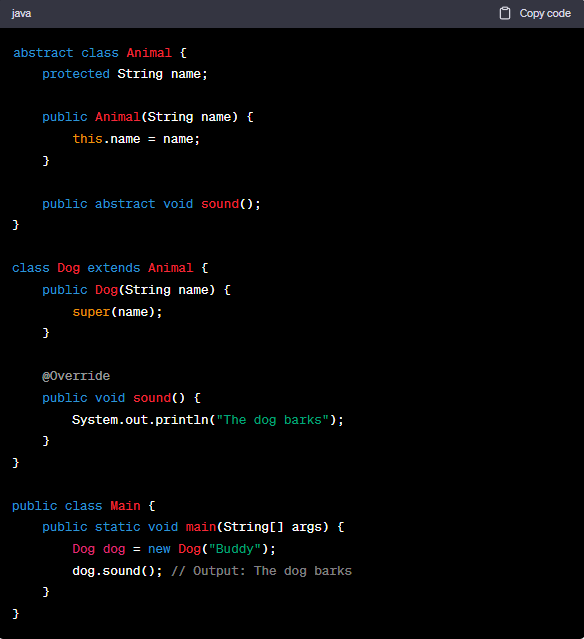
**Bu kodda Fruit interfeysdir. Apple və Grape claslar Friutdan implemets edib ovveride edir.**

**OOPs (Object-Oriented Programming System)**

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

Inheritance (Miras): Polymorphism (Çokbiçimlilik):

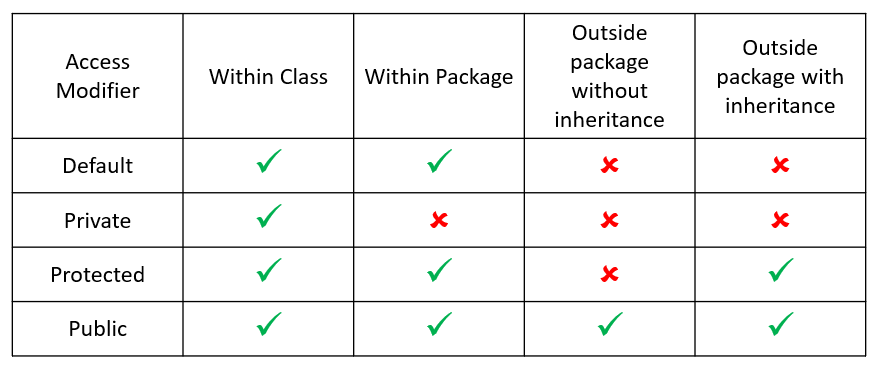
Abstraction (Soyutlama): Encapsulation (Kapsülleme):

burada getter setter hər zaman public olmalıdır.



**ACCESS MODİFİERS**

Class önündə public və default yazılır.



**STACK və HEAP yaddaşlar**

Hər hansı bir proqram təminatının hazırlanması zamanı mühim məsələlərdən biri də kompüterin yaddaşından doğru istifadə olunmasıdır. Bizim yazdığımız kodlar işə düşərəkən kompüterin RAM-ına ([Random Acces Memory](https://en.wikipedia.org/wiki/Random-access_memory)) yüklənir və proqram icra olunarkən RAM-in nə dərəcədə istifadə olunması proqramın məhsuldarlığına təsir edir. Kompüterin RAM yaddaşı iki əsas hissəyə ayrılır: ***Heap*** və ***Stack*** yaddaşları.

Heap və Stack anlayışları aparat təminatı ilə deyil proqram təminatı ilə bağlı anlayışlardır. Bu məsələni göz ardı etmək olmaz. Yəni bu iki yaddaş sahəsi RAM-da hər hansı fiziki yaddaş sahəsi deyil. Sadəcə olaraq yazdığımız proqram icra zamanı dəyişənləri saxlamaq üçün istifadə etdiyi yaddaşı iki hissəyə ayırır. Heap və Stack yaddaşılarının təyin olunması birbaşa bizim proqramın öhtəliyində, yəni dolayısı ilə bizim öhtəliyimizdə olan bir şeydir.

**Stack** yaddaşı kod yazarkən dəyəri statik olaraq təyin olunan [primitiv tipli dəyişənləri](https://en.wikipedia.org/wiki/Primitive_data_type) və obyekt referanslarını ([referance](https://en.wikipedia.org/wiki/Reference_(computer_science))) özündə saxlayır. Primitiv tipli dəyişənlərin istifadə edəcəkləri yaddaş sahəsi əvvəlcədən təyin olunur və yaddaşda müəyyən bir ünvana sahib olur. Belə dəyişənlərə əsasən [lokal dəyişənlər](https://www.guru99.com/local-vs-global-variable.html) aid edilir. Stack yaddaşında həmçinin primitiv tipli dəyişənlərdən istifadə edən funksiyalarda saxlanır. Stack yaddaşında saxlanan dəyişənlərə proqramın birbaşa müraciəti vardır və buna görə də stack yaddaşdan oxuma və yazma daha sürətlidir. Stack yaddaşın mənfi cəhəti onun həcminin *limitli* olmasıdır (ƏS`dən asılı olaraq).

**Heap** yaddaş sahəsi kodda elan olunan və proqramın icrası zamanı hər hansı bir mərhələsində dinamik olaraq dəyişə bilən və yaddaşda primitiv tiplərə nəzərən daha çox yer tutan obyektləri saxlamaq üçün ayrılır. Biz kodda hər hansı dəyişən elan edərkən onun dəyərini təyin etmədiyimiz zaman bir “*pointer*”, yəni bir *referans* yaratmış oluruq. Məsələni daha da açıqlamış olsaq, bir ünvan yaradırıq. Lakin yaratdığımız bu ünvan heç bir yeri göstərmir. Bu zaman yaratdığımız ünvan stack yaddaşında qeyd olunur. Proqramın icrası zamanı irəliləyən mərhələlərdə bu ünvana hər hansı bir dəyər təyin etdikdə bu dəyər artıq heap yaddaşında saxlanır və stack yaddaşında olan ünvan ilə əlaqələndirilir. Yəni biz həmin dəyəri çağırarkən stack yaddaşında yerləşən ünvana müraciət edirik və o da bizə heap`dəki dəyəri gətirir. Beləliklə, heap yaddaşa bizim birbaşa müraciət etməyimiz mümkün olmur. Buna görə də heap yaddaşa müraciət stack yaddaşa nisbətən daha ləngdir və heap yaddaşın həcmi stack yaddaşa nisbətən daha çox olur.

Məsələn, primitiv tiplərdən int ([integer](https://en.wikipedia.org/wiki/Integer_(computer_science))) tipə malik “a = 32” dəyişəni birbaşa 32 dəyəri ilə stack yaddaşda saxlanır. Digər tərəfdən referans tipə malik [array](https://www.geeksforgeeks.org/array-data-structure/) (məs: b = array.array(‘i’, [1, 2, 3])) dəyişəninin qaşılığında bir array obyekti yarandığı üçün onu işarə edən referans stack yaddaşında, onun dəyəri olan obyekt isə heap yaddaşda saxlanır.

A picture containing text, screenshot, diagram, font

Description automatically generated

Heap və Satck yaddaşların sxemi

Heap və stack yaddaşlarının digər fəqli cəhəti isə odur ki, stack yaddaşı siz idarə etmirsiniz. Yaddaş [CPU](https://az.wikipedia.org/wiki/M%C9%99rk%C9%99zi_prosessor) tərəfindən səmərəli şəkildə idarə olunur. Yəni dəyişən proqram daxilində istifadə olunandan sonra əgər bir daha bizə lazım olmayacaqsa stack yaddaşından silinir. Artıq bizim bunun üçün əlavə nəsə etməyə ehtiyacımız qalmır. Amma heap yaddaş belə idarə olunmur. Lazımsız obyekt və dəyişənləri biz özümüz yaddaşdan təmizləməliyik. Əgər biz bunu etməsək proqramımız gərəksiz yerə yaddaşdan daha çox istifadə edəcək və məhsuldarlıq aşağı düşəcək. Bu problemi aradan qaldırmaq üçün bəzi proqramlaşdırma dillərində “[Garbage Collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science))” anlayışından istifadə edilir. Garbage collection heap yaddaşda referansdan qopmuş, yəni müraciət üçün heç bir ünvana sahib olmayan obyekt və dəyişənləri yaddaşdan silir. Heap`də yerləşən obyektlərə biz eyni zamanda bir neçə ünvan (referance) da bağlaya bilərik.

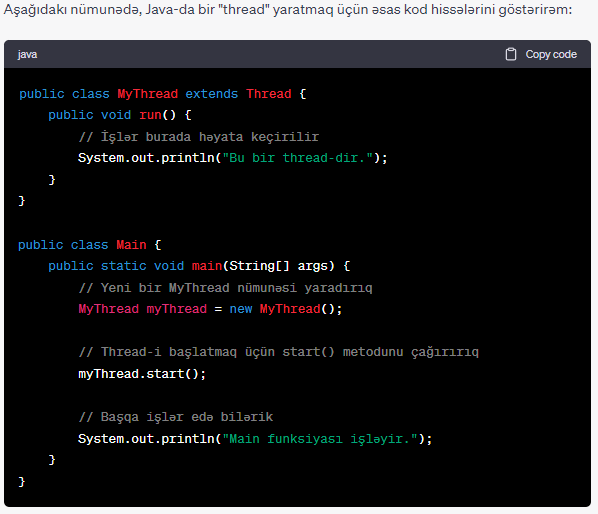
**GABBİDGE COLLECTOR**

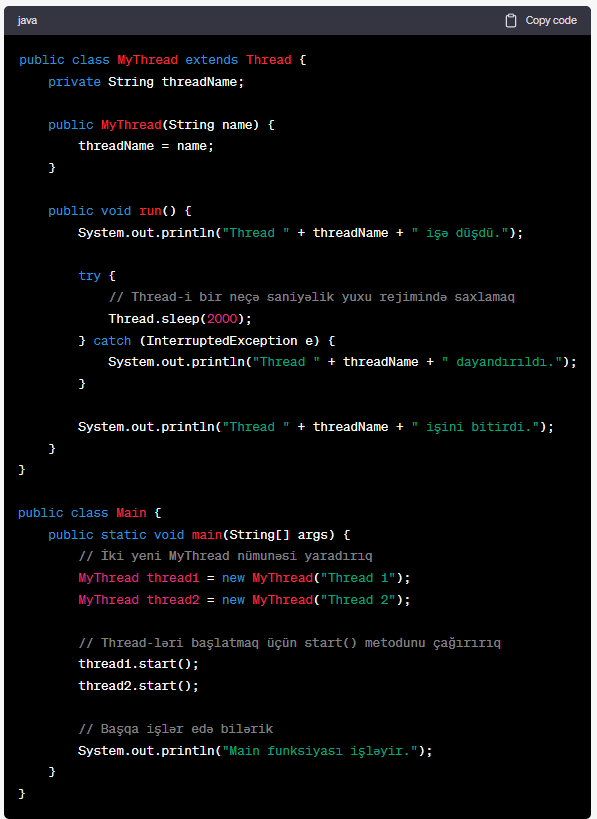
Gabbidge Collector - Obyektlərin referansları null və ya ümumiyyətlə obyektdən kəsilərsə boş obyektləri yaddaşda yer tutmamamsı üçün silir.

JVM has five types of GC implementations:

1. Serial Garbage Collector
2. Parallel Garbage Collector
3. CMS Garbage Collector
4. G1 Garbage Collector
5. Z Garbage Collector

**THREAD AND MULTİTHREAD (Synchronization and volatile)**

****

****

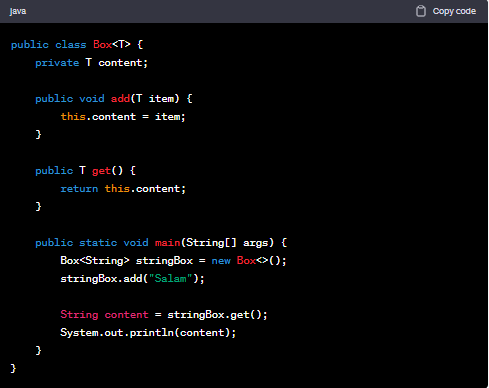
**Generics in Java**

Javada Generics, kodunuzda tip təhlükəsizliyini təmin etmək üçün istifadə olunan bir xüsusiyyətdir. Ümumi olaraq, bəlli bir veri tipini və ya veri tipi ailəsini istifadə etməyə imkan verir və beləliklə, işləmə vaxtında tip uyğunsuzluğu səhvlərini müəyyən etməyə imkan verir.

Generics istifadə etmənin bəzi faydaları var:

* Tip təhlükəsizliyi: Generics, kodunuzdakı səhvləri təhlil vaxtında aşkar etmək üçün bəlli bir veri tipini və ya veri tipi ailəsini istifadə etməyə imkan verir. Bu, işləmə vaxtında mümkün olan tip uyğunsuzluqlarının qarşısını almağa kömək edir.
* Kodun təkrar istifadəsi: Generics, bir sinif və ya metodun fərqli veri tipləri ilə istifadə olunmasına imkan verir. Bu, eyni kodun fərqli veri tipləri ilə istifadəsinin asanlaşmasına və kodunuzun daha ümumi və təkrar istifadə edilə bilən olmasına kömək edir.
* Tipdən asılılıq: Generics, fərqli veri tipləri ilə işləmək üçün istifadə edilə bilər, beləliklə tipdən asılılığı olan kod yazmağa imkan verir. Bu kodunuzu elastik edir və daha geniş bir veri tipi spektrinə uyğunlaşdırmağa imkan verir.

İşte nümunə kod, Generics'in necə istifadə edildiyini göstərmək üçün:



Bu nümunədə, "Box" adlı bir sinif təyin etdik və "T" ilə cəmləşdirdik. Bu, sinifin fərqli veri tipləri ilə işləyə biləcəyi deməkdir. Ardından, "add" və "get" metodlarını istifadə edərək məzmunu əlavə edə və ala bilərik. "main" metodu, "Box" sinifini "String" veri tipi ilə istifadə edərək örnəkləyir və bir sətir məzmununu əlavə edir və alır.

Bu nümunə, "Box" sinifinin bir çox fərqli veri tipi ilə istifadə edilə biləcəyini və tip təhlükəsizliyini təmin etdiyini göstərir.

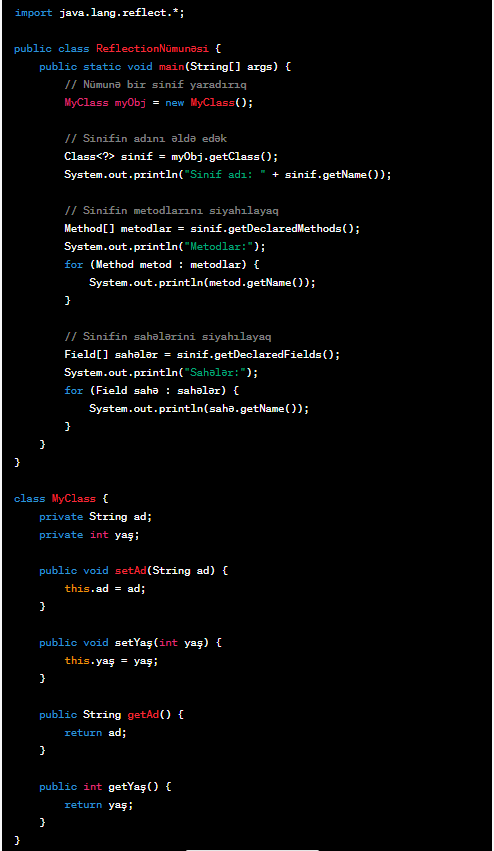
**Java Reflection**

Java Reflection, Java proqramında runtime zamanında bir sinifin veya obyektin yapısını araşdırmaq, məlumatlarını əldə etmək və hətta dəyişmək imkanı verən bir xüsusiyyətdir. Reflection, Javanın dinamik şəkildə işləməsinə imkan verir və proqramın işləmə vaxtında siniflər və obyektlər haqqında məlumat əldə etmək üçün istifadə edilə bilər.

Java Reflection aşağıdakı əməlləri həyata keçirməyə imkan verir:

* Sinifin adını, üst sinifini, tətbiq etdiyi interfeysləri və sahələrini əldə edə bilərsiniz.
* Sinifin və ya obyektin metodlarını və metodların parametrlərini çağırmaq mümkündür.
* Sinifin və ya obyektin sahələrinin dəyərlərini oxuya bilər və dəyişdirə bilərsiniz.
* Yeni bir obyekt yarada bilər və mövcud bir obyektin kopyasını yarada bilərsiniz.
* Sinifin və ya obyektin xüsusiyyətlərini, metodlarını və konstruktorlarını sıralaya bilərsiniz.

Aşağıda sadə bir Java Reflection nümunəsi verilmişdir:

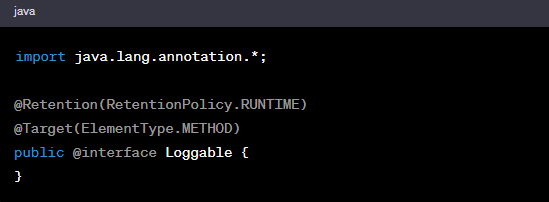


Yuxarıdakı nümunədə, MyClass adında bir sinif yaradırıq. ReflectionNümunəsi sinifində myObj adında bir MyClass obyekti yaradırıq. Ardından, getClass() metodu vasitəsi ilə MyClass sinifinin Class obyektini əldə edirik.

Daha sonra, getDeclaredMethods() və getDeclaredFields() metodlarını istifadə edərək sinifin metodlarını və sahələrini siyahılayırıq. Bu yolla, sinifin yapısal məlumatlarına runtime zamanında çatmaq mümkün olur.

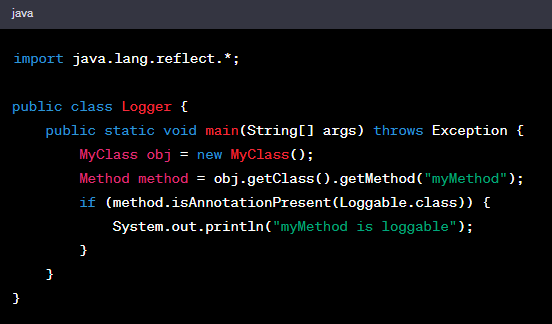
**Annotation və custom Annotation**

Java 5-dən bəri istifadə olunmaqdadır. Koda bəzi funksiyalar əlavə edir. Bunun vasitəsi ilə siniflərə,metodlara,paketlərə və s. barədə qeydlər əlavə etmək olur. İnterface tipindədir,içində sahələr(fields) yaradıla bilir. @ işarəsi ilə başlayır. C#-da attribute bunun analoqudur.



**A screen shot of a computer program

Description automatically generated with low confidence**

****

**@Target annotation**

| **İstifadə olunduğu sahələr** | **Açıqlama** |
| --- | --- |
| TYPE | Sinif,interface,abstrakt sinif başlarında |
| METHOD | Metod başlarında |
| FIELD | Global sahə başlarında |
| PARAMETER | Metod parametrlərində |
| CONSTRUCTOR | Constructor başında |
| ANNOTATION\_TYPE | Annotationların başında |
| PACKAGE | Packet decleration-ın başında |

package com.test;

import java.lang.annotation.Annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME) @interface melumat

{

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) melumat();

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) tarix();

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) muellif();

}

@melumat(melumat = "main class", tarix = "03.01.2016", yazar = "Tarlan")

@SuppressWarnings(value = { "" })

public class Main

{

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) throws [ClassNotFoundException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+classnotfoundexception)

{

Class<!--?--> c = Class.forName( "com.test.Main" );

[Annotation](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+annotation)[] ann = c.getDeclaredAnnotations();

 for( [Annotation](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+annotation) a : ann )

{

Class<!--?--> annType = a.annotationType();

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Annotation Type:"+annType);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(a.toString());

}

melumat infoAnn = c.getAnnotation( melumat.class );

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Məlumat:"+infoAnn.melumat());

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Tarix:"+infoAnn.tarix());

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Müəllif:"+infoAnn.muellif());

}

}

Output

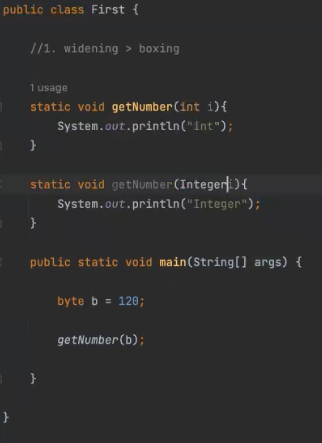
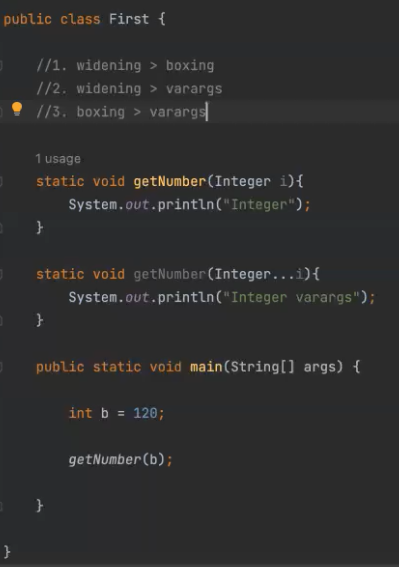
Annotation Type:interface com.test.melumat

@com.test.Bilgi(melumat=main class, tarix=03.01.2016, muellif=Tarlan)

Məlumat:main class

Tarix:03.01.2016

Müəllif:Tarlan

**Widening > boxing > varargs**

**Enum (Enumeration)**

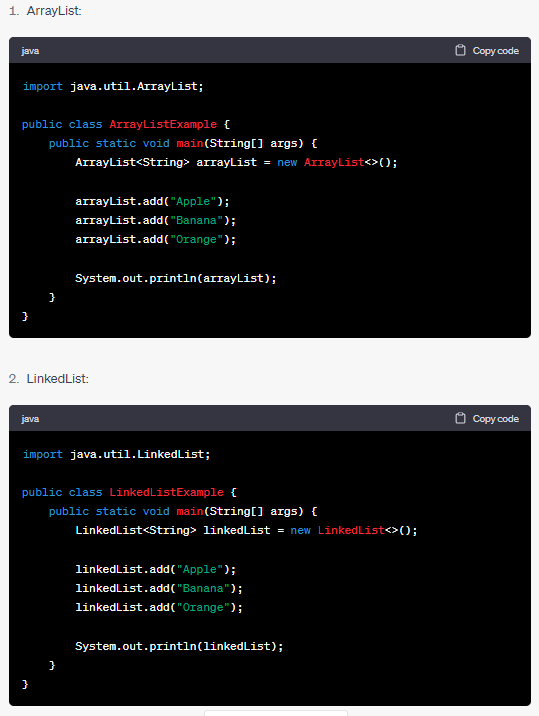
Java'da enumlar, bir dəyişkənin mümkün olan sabit dəyərlərini təmsil etmək üçün istifadə olunur. Bir enum tipi, ayrı ayrı dəyərlərə və bu dəyərləri istifadə etmək üçün gələn funksiyalara malik olur.

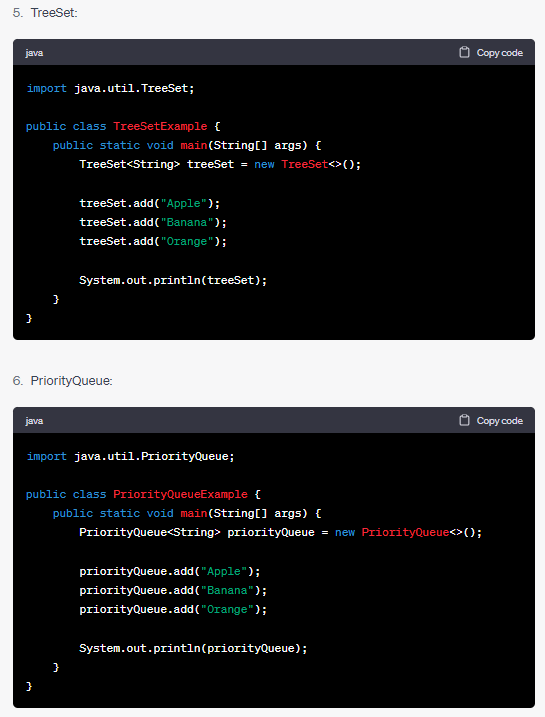


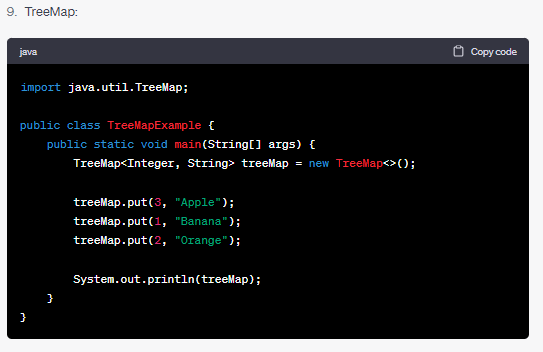
**Java Collection**

 A picture containing text, screenshot, diagram, font

Description automatically generated



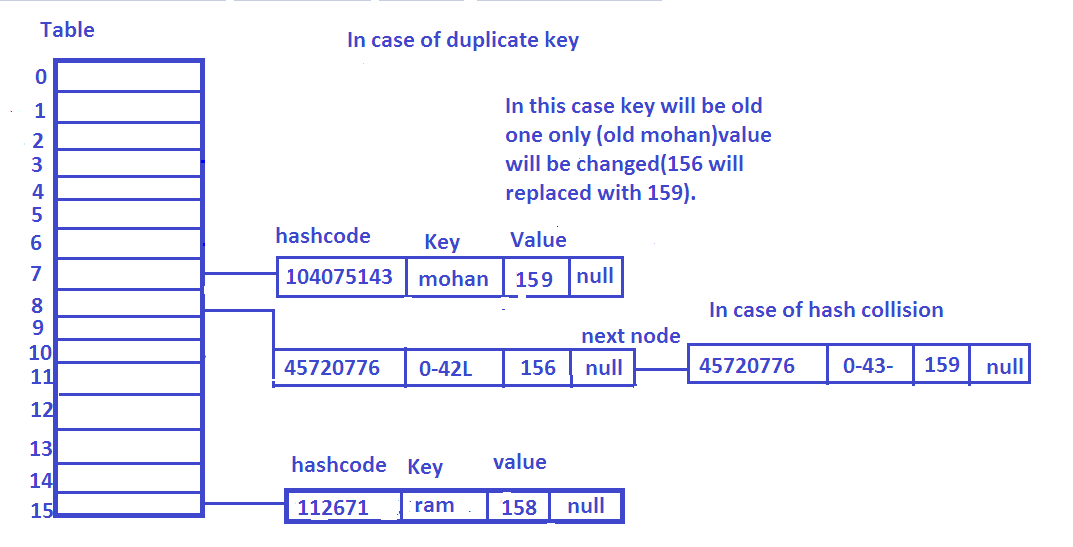




Arraylist-

Linketlist - (özündən əvvəl sonar pointi saxlayır)/ get əlverişsizdir.

Hashmap-

****Linkedhashmap-

Treehashmap-(key null qəbul etmir / əlifba sırası)

Lambda expression

Interfeysdən clasa hec bir override etmirik.

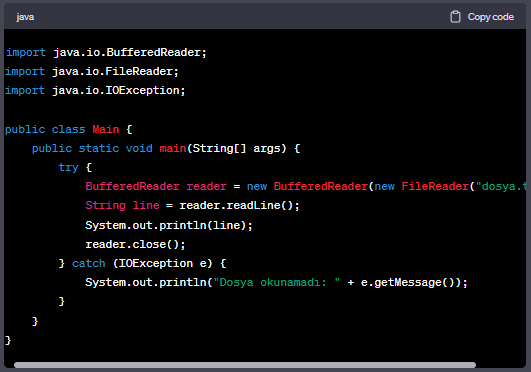
Lambda yalnız 1 abstrak metodu olan classla işleyir. Eyni adla metodu olanda lamda islemir.

Var keyvordu- local variable-lərdə istifadə edilir. Özü casting edir.

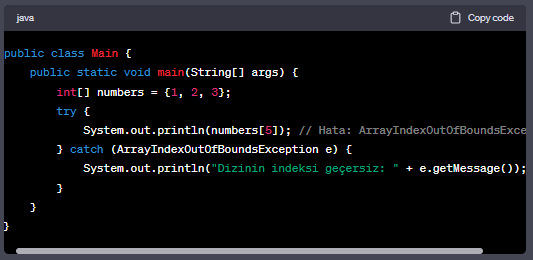
Stream Api -

**Exception try catch(checked , unchecked)**

Checked Exception (Mütləq Yoxlanan İstisnalar): Checked exception'lar, hata durumlarının programcı tarafından önceden tahmin edilebileceği durumlarda kullanılır. Bu tür hataların gerçekleşme ihtimali yüksek olduğundan, kodun hataya hazırlıklı olması gerekmektedir. Checked exception'lar try-catch blokları veya throws ifadesi kullanılarak yönetilir.

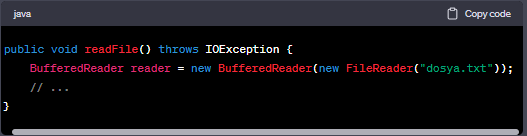


Unchecked Exception (Yoxlanmağa Ehtiyac Olmayan İstisnalar): Unchecked exception'lar, hata durumlarının önceden tahmin edilmesi zor veya imkansız olduğu durumlarda kullanılır. Bu tür hatalar genellikle programlama hatalarından kaynaklanır ve çoğunlukla çalışma zamanı (runtime) hatası olarak adlandırılır. Unchecked exception'ları kontrol etmek için try-catch bloğu kullanmak zorunlu değildir, ancak isteğe bağlı olarak kullanılabilir.

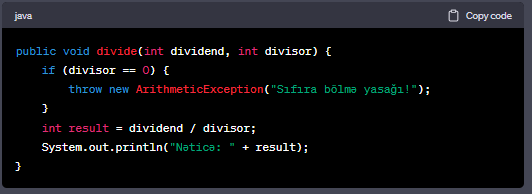


**Exception throws, throw, try with resources**

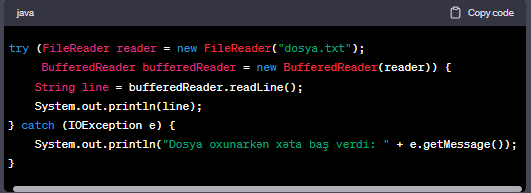
Throws (atmaq): throws ifadəsi, bir metodun içində meydana gələ biləcək xəta tiplərini bildirir və metodun çağırıldığı yerdə bu xətalara uyğun bir şəkildə işlənməsinin tələb olunduğunu bildirir. Bu, metodda fırladılan bir istisna obyektini (exception object) yönetmek üçün üst məsuliyyətin çağırılan yerdə olduğunu göstərir.



Throw (atmaq): throw ifadəsi, istisna obyektlərini yaratmaq və onları proqramın müxtəlif bölgələrindən fırlatmaq üçün istifadə olunur. İstisna fırlatmaq, proqramın normal işleyişini dayandırmaq və xətanın uyğun şəkildə idarə edilməsini təmin etmək üçün istifadə olunur.

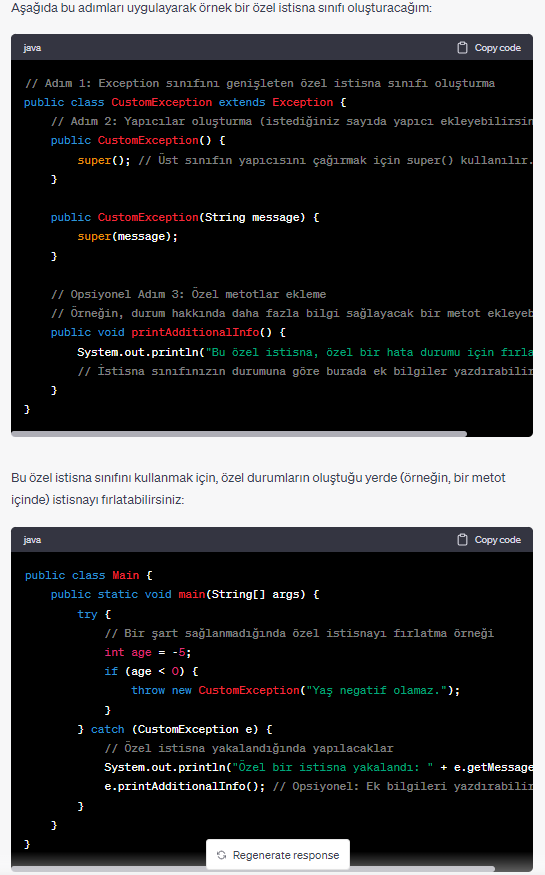


Try-with-resources: try-with-resources ifadəsi, Java 7 ilə birlikdə əlavə olunan bir syntaxdır və resursları avtomatik olaraq bağlamaq üçün istifadə edilir. Bu, bir resursun istifadəsi bitdikdə avtomatik olaraq bağlanmasını təmin edir. Bu, resursların (məsələn, dosya açmaq kimi) düzgün şəkildə qapadılmasını və sərbəst qoymağını təmin edir.



Yukarıdakı kod bloku, try-with-resources ifadəsini göstərir. FileReader və BufferedReader obyektləri resurs kimi istifadə olunur və blok bitdikdə avtomatik olaraq bağlanır. Buna görə də, bu resurslar müxtəlif səbəblərdən bağlamaq lazım olsa belə, proqramçının onları mənfi təsirlənməsinə ehtiyacı yoxdur.

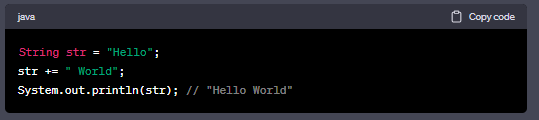
Customized Exception



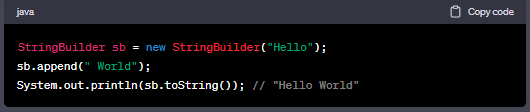


**String, StringBuilder, StringBuffer**

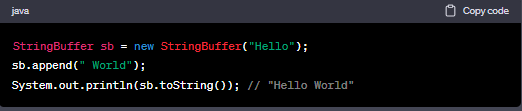
Özetle, String sınıfı değiştirilemez (immutable) metinleri temsil ederken, StringBuilder ve StringBuffer sınıfları değiştirilebilir (mutable) metinleri temsil eder. StringBuilder, performans açısından tercih edilen bir seçenekken, StringBuffer senkronizasyon gerektiren çoklu thread ortamlarında kullanılır.



String: String sınıfı, değiştirilemez (immutable) metinleri temsil eder. Yani, bir kez oluşturulduktan sonra değeri değiştirilemez. Herhangi bir değişiklik yapılmak istendiğinde, yeni bir String nesnesi oluşturulur. String sınıfı bellekte yer kapladığı için bu durum performans açısından dezavantajlı olabilir.



StringBuilder StringBuilder sınıfı, değiştirilebilir (mutable) metinleri temsil eder. Yani, metin üzerinde değişiklik yapılabilir. StringBuilder nesneleri, daha fazla performans sağlamak için tercih edilir, çünkü mevcut bir StringBuilder nesnesi üzerinde yapılan değişiklikler doğrudan aynı nesneyi günceller.



StringBuffer: StringBuffer sınıfı, StringBuilder gibi değiştirilebilir (mutable) metinleri temsil eder. StringBuilder ile aynı işlevselliği sağlar, ancak StringBuffer nesneleri senkronize edilmiştir ve birden fazla thread tarafından güvenli bir şekilde paylaşılabilir. Bu özelliği, çoklu thread ortamlarında kullanılması gereken senkronizasyon gerektiren durumlarda tercih edilmesini sağlar.

IP 192.168.1.1 0-dan 255ə qədər olur.

**PROYEKT**

**1 Servises APİ**

**2 Authentication and authorization**

**3 Validation**

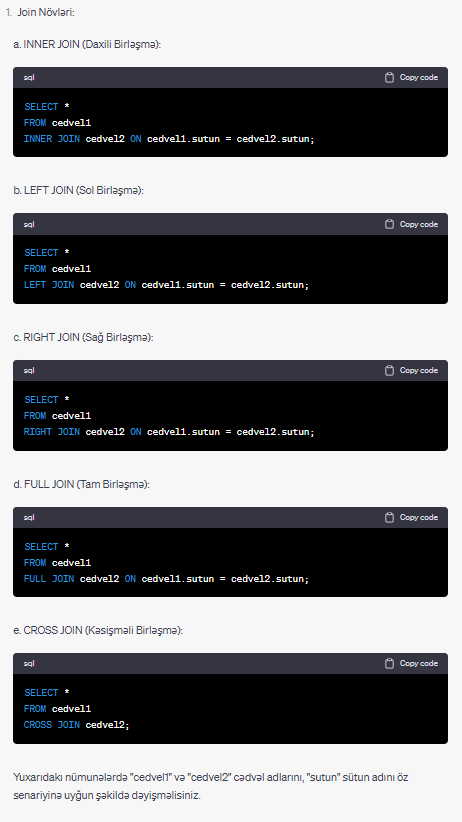
**4 Swagger**

**5 Spring**

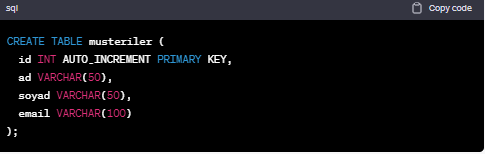
**6 Database**

**7 CRUD operation**

**MySQL**

MySQL-də veritabanı yaratma, cədvəl yaratma (id sütunu avtomatik artan) və CRUD (Create, Read, Update, Delete) əməliyyatları üçün nümunələr aşağıda verilmişdir.







**GitHub and BitBucket**

GitHub, bir şirkət və ehtiyatlı bir kod paylaşım və versiya nəzarəti platformasıdır. İstifadəçilərə kodun müxtəlif versiyalarını saxlamaq, kodu başqaları ilə bölüşmək və birgə çalışmaq imkanı verir. Aşağıda GitHub-dakı əsas əmrlərdən bəzilərini nümunələrlə təqdim edirəm:

* git clone: Bir repozytoriyanın tam nüsxəsini yükləmək üçün istifadə olunur.

Nümunə: **git clone https://github.com/openai/chatgpt**

* git pull: Uzak bir repozytoriyadan son dəyişiklikləri lokal repozytoriyaya əldə etmək üçün istifadə olunur.

Nümunə: **git pull origin master**

* git add: Dəyişiklikləri commit etmək üçün faylları seçmək üçün istifadə olunur.

Nümunə: **git add dosya.py**

* git commit: Dəyişiklikləri yerli repozytoriyada təsdiqləmək üçün istifadə olunur.

Nümunə: **git commit -m "Yeni funksiya əlavə edildi"**

* git push: Yerli repozytoriyadakı təsdiqləmələri uzak repozytoriyaya göndərmək üçün istifadə olunur.

Nümunə: **git push origin master**

Bu əmrlər yalnız bir neçəsinə nümunə olaraq göstərilmişdir və GitHub-dakı daha çox əmri öyrənmək üçün ətraflı dokumentasiyadan istifadə edə bilərsiniz.

Əvvəlcədən yüklədiyiniz Git proqramını yüklədiyinizdən və GitHub hesabınızı qurduğunuzdan əmin olun. Ardından aşağıdakı ardıcıllığı izləyərək Java kodunuzu GitHub-a yükləyə bilərsiniz:

1. İstifadə etdiyiniz Java proyektinin olduğu qovluğa keçin: **cd /proyektinizin\_qovluğu**
2. Git'i başlatmaq üçün aşağıdakı əmri daxil edin: **git init**
3. GitHub'dan aldığınız repozytoriyanın URL-ni əldə edin. Bu, GitHub saytında sağ üstdə "Code" düyməsini basaraq əldə edə bilərsiniz.
4. Uzak repozytoriya bağlantısını qoşmaq üçün aşağıdakı əmri daxil edin: **git remote add origin https://github.com/sizin\_hesabınız/repozytoriyanız.git**

(Yuxarıdakı URL-ni öz GitHub hesabınıza və repozytoriya adınıza uyğun şəkildə dəyişməlisiniz)

1. Bütün faylları yükləmək üçün aşağıdakı əmri daxil edin: **git add .**
2. Əlavə etdiyiniz faylların dəyişikliklərini təsdiqləmək üçün aşağıdakı əmri daxil edin: **git commit -m "Java kodu əlavə edildi"**
3. Yerli repozytoriyanızdakı təsdiqləmələri GitHub'a göndərmək üçün aşağıdakı əmri daxil edin: **git push origin master** (Əgər "master" yerinə başqa bir şaxələyiniz varsa, onu uyğun şəkildə dəyişməlisiniz)

Bu ardıcıllıq, Java kodunuzu GitHub-a yükləmək üçün əmrlərin sıralanmasıdır. Bu əmrlərlə kodunuzu yerli repozytoriyada təsdiqləyəcək və onu GitHub'a göndərəcəksiniz.