



# जावा प्रोग्रामिंग

Hindilearn द्वारा प्रकाशित



# जावा प्रोग्रामिंग

## जावा ट्यूटोरियल के बारे में

कंप्यूटर प्रोग्रामिंग के इतिहास में जावा यह एक महत्वपूर्ण लैंग्वेज है। जावा एक सरल भाषा है क्योंकि इसके syntax सरल, स्वच्छ और समझने में आसान है। जावा यह सी++ से मिलती-जुलती है क्योंकि यह भी ऑब्जेक्ट-ओरिएंटेड लैंग्वेज है। जावा के syntax अपनी अंग्रेजी भाषा से सम्बंधित होने के कारण प्रोग्रामर को इसे सिखने में रुचि मिलती है।

## ट्यूटोरियल किसे पढ़ना चाहिए

यदि आप कंप्यूटर साइंस / इंजीनियरिंग को आगे बढ़ाने की योजना बनाते हैं, तो सी और सी प्लस प्लस के बाद जावा को सीखना जरूरी है क्योंकि यह आपको प्रोग्रामिंग के अंतर्गत संकल्पनाओं को समझाने में मदद करता है।

## जावा सिखने के लिए अपेक्षित

जो जावा सीखना चाहता है तो उसके पास Computer (Desktop, CPU, Keyboard) या laptop JDK(Java Development Kit) और text editor को डाउनलोड करके इनस्टॉल करे। इस PDF File को Run करने के लिए किसी भी Browser या PDF Reader का उपयोग करे।

## अपने अधिकार

जो भी सामग्री आप इस PDF के माध्यम से उपयोग में ला रहे हैं वो Hindilearn.in की संपत्ति है जैसे कि छायाचित्र और पाठ। अगर आप इसकी व्यावसायिक रूप में नक़ल करने का प्रयास करेंगे तो आप पर राइट लॉज़ के अनुसार कार्रवाई भी हो सकती है।

## संपर्क

अगर आपको कोई गलत जानकारी या अशुद्धलेखन इस ट्यूटोरियल के माध्यम से प्राप्त होता है तो आप हमें [help@hindilearn.in](mailto:help@hindilearn.in) इस पते पर संपर्क कर सकते हैं।

## JAVA INTRODUCTION

11

*History for Java/Core Java*

11

*Java Basics*

13

*Java Hello Program*

14

*Reserved Keywords*

21

## JAVA DATA TYPES

23

*Primitive Data Types*

23

*short*

23

*float*

25

*long*

23

*double*

25

*character*

24

*boolean*

26

*Default Values for Primitive Data Types*

26

*Non-Primitive Data Types*

27

*Strings*

96

*Arrays*

99

## JAVA VARIABLES

28

*Variable Introduction*

28

*Types of Variables*

28

*Local Variable*

28

*Static Variable*

30

*Instance Variable*

29

## JAVA OPERATORS

31

*Arithmetic Operators*

31

*Relational Operators*

32

*Logical Operators*

34

*Bitwise Operators*

35

*Assignment Operators*

37

*Increment and Decrement Operators*

38

*Conditional Operators*

39

## JAVA LOOPS

40

*While Loop*

40

*Do While Loop*

41

*For Loop*

42

*Normal for Loop*

43

*Enhanced or*

44

*Foreach Loop*

## JAVA CONTROL STATEMENTS

45

*If Statement*

45

*If Else Statement*

46

*Else If Statement*

47

*Switch Case Statement*

48

*Break Statement*

50

*Continue Statement*

51

### JAVA ARRAYS

52

*Array Introduction*

52

*Types of Arrays*

55

*One-Dimensional Array* 55

*Two-Dimensional or Multi-Dimensional Array* 56

### JAVA STRINGS

58

*What is Strings*

58

*String Methods*

60

*contentEquals* 62  
*endsWith* 63  
*equals* 63  
*equalsIgnoreCase* 64  
*matches* 65  
*regionMatches* 66  
*startsWith* 69  
*getBytes* 70  
*charAt* 72  
*toCharArray* 72  
*subSequence* 73  
*compareTo* 74  
*compareToIgnoreCase* 75  
*indexOf* 76  
*lastIndexOf* 80

*length* 84  
*getChars* 85  
*copyValueOf* 86  
*valueOf* 88  
*concat* 89  
*intern* 89  
*replace* 90  
*replaceAll* 91  
*replaceFirst* 92  
*substring* 92  
*toLowerCase* 94  
*toUpperCase* 94  
*trim* 95  
*split* 95

### JAVA METHODS 99

*Methods Introduction* 99

*Date and Time Methods* 104

*Character Methods* 105

<i>isDigit</i>	106	<i>isWhitespace</i>	108
<i>isLetter</i>	106	<i>toLowerCase</i>	108
<i>isLowerCase</i>	107	<i>toUpperCase</i>	109
<i>isUpperCase</i>	107		

*Math Methods* 109

<i>abs</i>	111	<i>max</i>	119
<i>acos</i>	111	<i>min</i>	120
<i>asin</i>	112	<i>pow</i>	120
<i>atan</i>	113	<i>random</i>	121
<i>atan2</i>	113	<i>round</i>	121
<i>ceil</i>	114	<i>sin</i>	122
<i>cos</i>	115	<i>sinh</i>	123
<i>cosh</i>	115	<i>sqrt</i>	124
<i>exp</i>	116	<i>tan</i>	124
<i>floor</i>	116	<i>toDegrees</i>	125
<i>floorDiv</i>	117	<i>toRadians</i>	125
<i>log</i>	118		
<i>log10</i>	119		

### JAVA SPECIAL KEYWORDS 127

*Static Keyword* 127

<i>Static/Class Variable</i>	127	<i>Static/Class Method</i>	128
------------------------------	-----	----------------------------	-----

*Final Keyword* 130

<i>final Variable</i>	130	<i>final Class</i>	132
<i>final Method</i>	131		

<i>Super Keyword</i>			133
<i>For Variable</i>	133	<i>For Constructor</i>	135
<i>For Method</i>	134		
<i>This Keyword</i>			137
<b>JAVA EXCEPTION HANDLING</b>			<b>142</b>
<i>Introduction of Exception Handling</i>			142
<i>Checked Exception</i>	144	<i>Un-checked Exception</i>	145
<i>try and catch</i>			146
<i>finally, throw and throws</i>			150
<b>JAVA FILE HANDLING</b>			<b>153</b>
<i>Introduction of File Handling</i>			153
<i>Byte Streams</i>			153
<i>BufferedInputStream</i>	154	<i>FileOutputStream</i>	158
<i>BufferedOutputStream</i>	155	<i>FileInputStream</i>	159
<i>DataOutputStream</i>	156	<i>PrintStream</i>	160
<i>DataInputStream</i>	157		
<i>Character Streams</i>			161
<i>BufferedReader</i>	162	<i>FileWriter</i>	165
<i>BufferedWriter</i>	163	<i>FileReader</i>	166
<i>PrintWriter</i>	164		
<b>JAVA CLASSES AND OBJECTS</b>			<b>167</b>

<b>JAVA INHERITANCE</b>	<b>171</b>
<i>Types of Inheritance</i>	172
<i>Single Inheritance</i>	173
<i>MultiLevel Inheritance</i>	174
<i>Constructor</i>	178
<i>Default Constructor</i>	179
<i>Parameterized Constructor</i>	179
<i>Hierarchical Inheritance</i>	175
<i>Constructor Overloading</i>	180
<b>JAVA POLYMORPHISM</b>	<b>183</b>
<i>Introduction</i>	183
<i>Upcasting</i>	185
<i>Compile and Run Time Polymorphism</i>	186
<i>Compile-Time Polymorphism</i>	186
<i>Run-Time Polymorphism</i>	187
<b>JAVA BINDING</b>	<b>188</b>
<i>Static Binding</i>	189
<i>Dynamic Binding</i>	190
<b>JAVA ABSTRACT CLASS</b>	<b>191</b>
<i>Abstract Class and Method</i>	191
<i>Abstract Classes using Multilevel Inheritance</i>	193



<b>JAVA METHOD OVERLOADING AND OVERRIDING</b>	<b>197</b>
<i>Method Overloading</i>	197
<i>Method Overriding</i>	204
<b>JAVA ACCESS MODIFIERS</b>	<b>206</b>
<i>What is Access Modifier and its Types</i>	206
<i>default Access Modifier</i>	206
<i>public</i>	207
<i>private</i>	207
<i>protected</i>	208
<b>JAVA DATA ENCAPSULATION</b>	<b>210</b>
<b>JAVA INTERFACES</b>	<b>212</b>
<i>Introduction</i>	212
<i>Relation between Class and Interface</i>	215
<i>Rules of Interfaces</i>	218
<b>JAVA ENUMERATION</b>	<b>219</b>

## **JAVA MULTITHREADING**

**223**

<i>Multithreading and Life Cycle of Thread</i>	223
<i>Thread Priority</i>	225
<i>Class Thread and Methods</i>	227
<i>Creating Thread</i>	228
<i>Naming Thread</i>	230
<i>Joining Thread</i>	232
<i>Sleeping Thread</i>	236
<i>Yield() Method</i>	237
<i>Synchronization</i>	239

## **JAVA PACKAGES**

**243**

<i>Introduction</i>	243
<i>In-Built Packages</i>	243
<i>User Defined Packages</i>	244

# Introduction

## History for Java/Core Java

### History

Java ये Computer Language का 'Sun Microsystem' इस company के अंतर्गत 'James Gosling' ने अविष्कार किया |

लेकिन Java Language बनाने में Mike Sheridan और Patrick Naughton इन दोनों का भी बड़ा हात है | इन तीनों में 'James Gosling' ने महत्वपूर्ण कामगिरी निभाई | Java ये Language मूलतः Television के परस्पर सम्बन्ध के लिए बनायीं गयी थी | लेकिन वर्तमान में ये Language बहुत ही महत्वपूर्ण साबित हुई और भविष्य में इसका महत्त्व और भी बढ़ जाएगा |

Java का पहला नाम 'Oak' इस पेड़ से रखा गया | ये रखने का मतलब 'James Gosling' और उनके सहकारी जहां पर Java के लिए काम करते थे, वहां पर 'Oak' का पेड़ था और 'Oak' ये कई देशों का राष्ट्रीय पेड़ भी है | इसी लिए 'Oak' ये नाम रखा गया | ये 'Oak' नाम 1991 में रखा गया |

लेकिन ये 'Oak' नाम पहले से ही 'Oak Technologies' का था |

उसके बाद 'Oak' का नाम बदलकर 'Java' रखा गया |

Java ये नाम रखने की एक ही वजह थी कि, जब 'James Gosling' और उनके सहकारी जब काम करते थे तब 'Java' नाम के बीज की Indonesian coffee पीते थे और 'Java' ये नाम काफी नया भी था | इसीलिए 'Java' नाम रखा गया |

अभी 2010 में Sun Microsystem ने 'Oracle' को ये बेच दी |

जब java का प्रारंभ हुआ तब उसका पहला नमूना '1995' में आया |

निचे Java के सभी versions दिखाए गए हैं |

Java Versions	Released Dates
JDK Alpha and Beta	1995
JDK 1.0	23rd Jan, 1996
JDK 1.1	19th Feb, 1997
J2SE 1.2	8th Dec, 1998
J2SE 1.3	8th May, 2000
J2SE 1.4	6th Feb, 2002
J2SE 5.0	30th Sep, 2004
Java SE 6.0	11th Dec, 2006
Java SE 7.0	28th July, 2011
Java SE 8.0	18th March, 2014

# Introduction for Java

Java ये दुनिया भर में बहुत ही प्रसिद्ध programming Language है ।

Java के लिए तीन प्रकार बनाये गए हैं ।

Java SE (Standard Edition)

Java EE (Enterprise Edition)

Java ME (Micro Edition)

1. Java SE

Java SE ये Basic Programming के लिए इस्तेमाल किया जाता है, इसे Core Java भी कहा जाता है ।

2. Java EE

Java EE ये Advanced Programming है , इसे Advanced Java भी कहते हैं ।

3. Java ME

Java ME ये mobile programming के लिए उपयुक्त है, लेकिन इसका android से कोई सम्बन्ध नहीं है ।

Java का जब प्रारंभ हुआ तब उसे मुफ्त किया गया था और वर्तमान और भविष्य में भी मुफ्त रहेगा । Java का एक और महत्व है ये Write Once, Run Anywhere है, इसका मतलब ये किसी भी mobile-based, windows-based और web-based Application पर चलाया जाता है ।

## Java Features

Java Programming के लिए बहुत सारे Features के साथ बनाई गयी है ।

जैसे कि,

Simple

Secure

Object-Oriented

Independent

Portable

Robust

Interpreted

Multi-threaded

High Performance

Distributed

**Simple** : ये Language C++ से कुछ समान है । जो भी User C++ से अच्छा ज्ञान प्राप्त करता है, उसे Java सिखने में भी कोई दिक्कत नहीं होती ।

**Secure** : ये Language C और C++ से भी ज्यादा secured है । ये Language virus-free Language है ।

**Object-Oriented** : ये Language Class-based और Object-based है ।

**Independent** : ये Language independent है , अगर कोई software बनाना हो तो इसे किसी चीज की जरूरत नहीं होती ।

**Portable** : इस Language को किसी भी platform पर Run किया जा सकता है ।

**Robust** : ये Language और इसका Memory Management बहुत ही मजबूत है ।

**Robust** : ये Language और इसका Memory Management बहुत ही मजबूत है ।

**interpreted** : ये Language एक interpreted Language है ।

**Multi-threaded** : इस Language में एक से ज्यादा programs को एक साथ Run किया जा सकता है |High

**Performance** : Java के Compilers program execution के लिए ज्यादा समय नहीं लेते, इसीलिए ये एक High-performing Language मानी जाती है ।

**Distributed** : Java के बनाये हुई program को एक से दुसरे computer पर distribute किया जाता है ।

## Java Basics

### JVM(Java Virtual Machine) क्या है ?

JVM ये Program execute करने के लिए इस्तेमाल किया जाता है । ये एक software है, पर ये machine की तरह काम करता है ।

जब C और C++ का program; लिखा जाता था तब program सिर्फ Compiler द्वारा compile किया जाता था । लेकिन Java में program; Compile और interpreted होता है ।

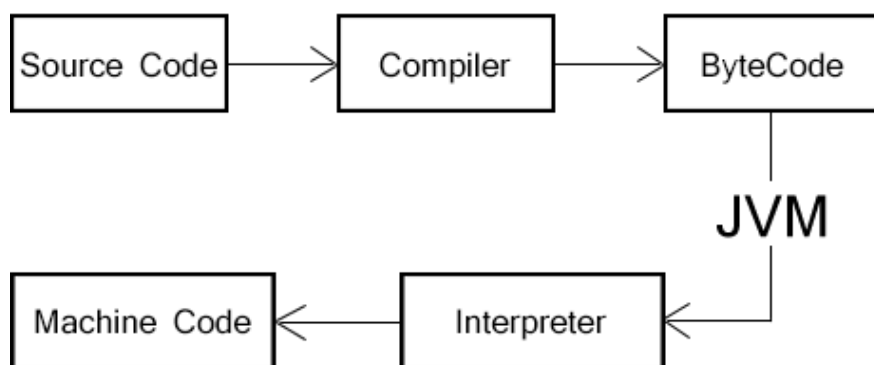
Java के program; run करने के लिए Computer के interpreter और Compiler का इस्तेमाल किया जाता है ।

Java का program जब run होता है, तब एक bytecode(class file); create करता है और उसके बाद interpreter उसे machine code में convert कर देता है ।

Bytecode ये portable है, ये किसी भी machine पर interpreted किया जा सकता है । for eg. अगर bytecode windows-based Application पर generate किया है , तो वो bytecode linux-based Application पर भी interpreted किया जा सकता है ।

Bytecode जब JVM पर आता है, तब JVM से bytecode; interpreter पर जाकर machine code में convert हो जाता है ।

Bytecode जब एक बार compile हो जाए तो वो किसी भी Computer के या Operating-system के JVM द्वारा interpreted हो सकता है ।



# Java का Source Code कहाँ पर लिखा जाता है ?

Java का Source Code लिखने के लिए किसी भी text-editor उपयुक्त रहता है ।

जैसे कि,

**Notepad/Notepad++** : Java का program Simple या जो user कुछ basic सिख रहा हो तो उसके लिए सही है ।

**Eclipse** : Java का program बनाने के लिए ये बहुत ही अच्छा Java IDE है, ये open-source और मुफ्त रहता है ।

**Netbean** : ये IDE भी Java के लिए काफी अच्छा है, ये भी open-source और मुफ्त रहता है ।

**Bluej** : ये IDE खास Java के लिए बनाया है , ये भी open-source और मुफ्त में रहता है । इस IDE को JDK(Java Development Kit) की जरूरत पड़ती है ।

## Java Hello Program

### Structure for Java Program

java का program पांच विभागों में बटा जाता है ।

java का hello program; सबसे पहला program है ।

<b>Package</b>	import java.io.*;
<b>class name and opening curly brace</b>	class Hello{
<b>Main Method and opening curly brace</b>	public static void main(String args[]){
<b>comment</b>	// comment or /* comment */
<b>statement</b>	System.out.println("Hello World");
<b>closing curly braces</b>	} }

**Package** : यहाँ पर program के अनुसार package को declare किया जाता है । यहाँ पर एक से अधिक packages भी declare किये जाते है । packages में classes, interfaces और sub-packages defined होते है । ये packages; program के सबसे ऊपर declare किये जाते है ।

**class name and opening curly brace** : यहाँ पर 'class' इस keyword के साथ class का नाम लिखा जाता है । लेकिन class का नाम कोई java का keyword नहीं होता ।

**Main Method and opening curly brace** : Main method में पांच हिस्से होते है ।

## What is public static void main(String args[]) ?

**public** : ये एक access specifier है , इसे access specifier भी कहा जाता है | public का मतलब किसी भी class से method को access किया जाता है |

**static** : static मतलब class को बिना object या instance से access किया जाए |

**void** : ये main method कुछ भी return ना करने की अनुमति देता है |

**main** : ये line में से मुख्य keyword है | JVM इसे ढूँढकर application का starting point दे देता है |

**(String args[])** : ये जो String है , ये कोई keyword नहीं है, ये एक class है चूँकि इसका पहला character; uppercase में है और args[] ये String class के object का array है | ये main function का argument है | इस main method के बाद इसको curly brace({) से open किया जाता है |

**comment** : ये एक comment tag है | इसे compiler द्वारा पढ़ा नहीं जाता | इसके दो प्रकार हैं |

- **One Line Comment** : इसे दो slashes के साथ comment दी जाती है | // comment
- **Multiple Line Comment** : इसे एक slash और asterisk(/\*) के साथ शुरू किया जाता है और asterisk और slash के साथ close किया जाता है (\*/). /\* Multiple line comment \*/

**statement** : statement के लिए तीन प्रकार हैं |

## What is System.out.println() ?

**System** : ये एक predefined class है |

**out** : ये PrintStream class का object है , इसे System class के अन्दर public static final इन keywords के साथ define किया गया है |

**println** : ये PrintStream class की method है | इसे public के साथ PrintStream class के अन्दर define किया गया है | ये print के साथ 'ln' है ये cursor को new line पर जाने की अनुमति देता है |

**closing curly braces** : यहाँ पर main method और class को close किया गया है |

Source Code :

```
//HelloProgram.java
class HelloProgram{
    public static void main(String []args) {
        System.out.println("Hello World!");
    }
}
```

Output :

```
Hello World!
```

# How to Run Java Program ?

Java का program जब notepad पर लिखा जाता है , तब .java इस extension के साथ दिया जाता है ।  
java के file का नाम class का नाम होता है । अगर एक java के file में एक से ज्यादा classes हो, तब जिस class के अन्दर main method होता है, तो उस class का नाम file का नाम होता है । किसी भी class का नाम और file का नाम पहला character; uppercase में होता है ।

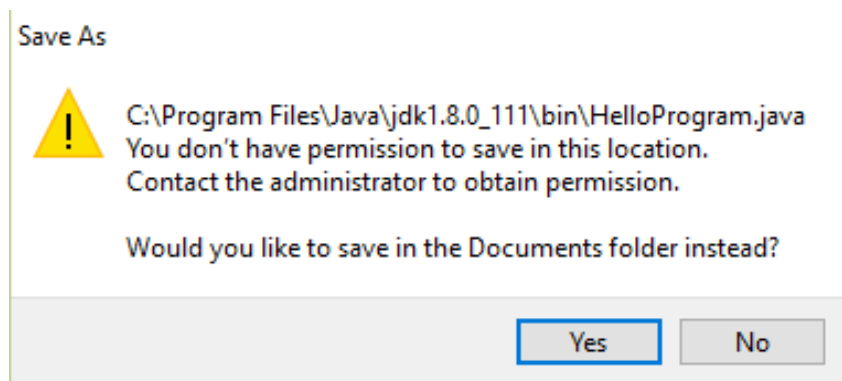
For Example,

**HelloProgram.java**

इस file को java के bin directory पर save किया जाता है ।

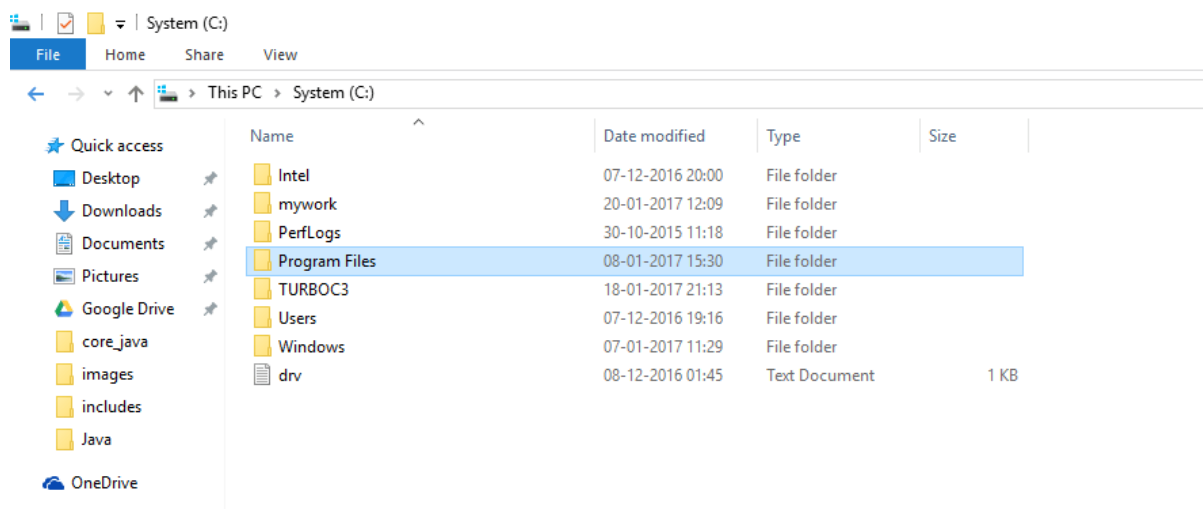
For Example. **C:\Program Files\Java\jdk1.8.0\_111\bin**

जब इस directory पर save करने की बारी आती है , तब file; save करने के लिए Operating system द्वारा permission नहीं दी जाती । इसके लिए निचे के pictures को देखे ।



इस problem को solve करने के लिए निचे वाले pictures है ।

पहले C: drive के Program Files पर click करे ।

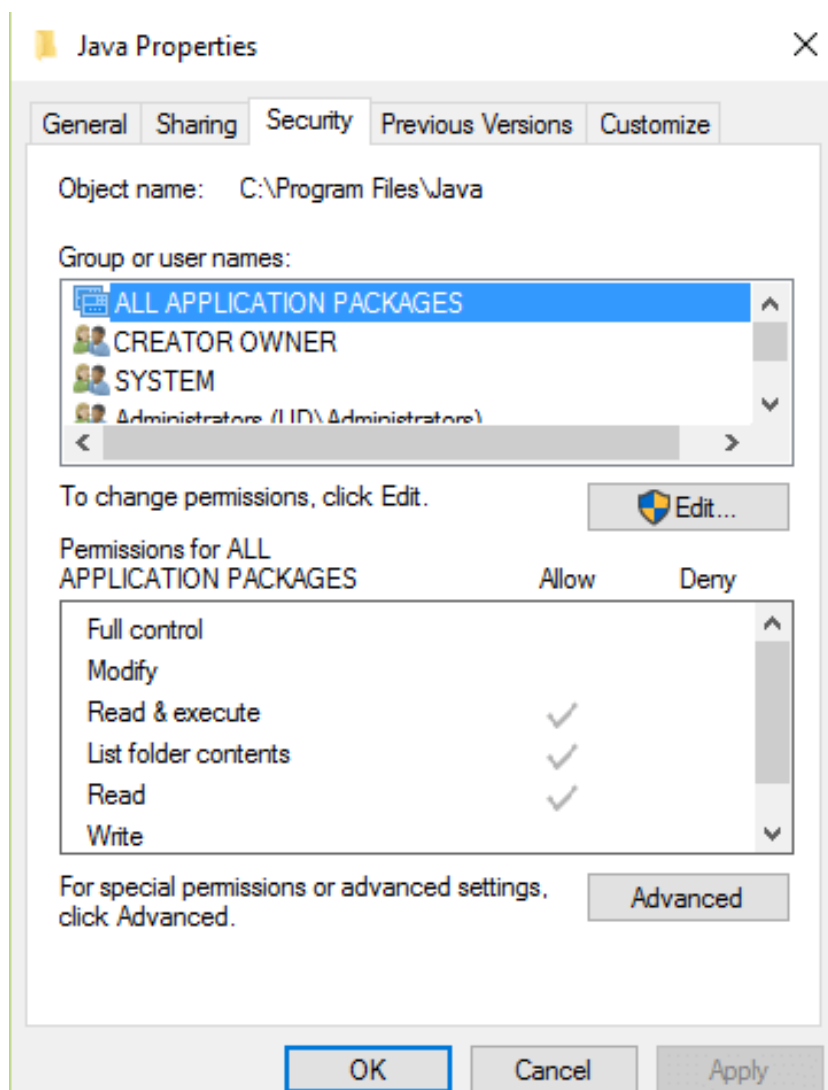


उसके बाद java इस folder पर right click करे।

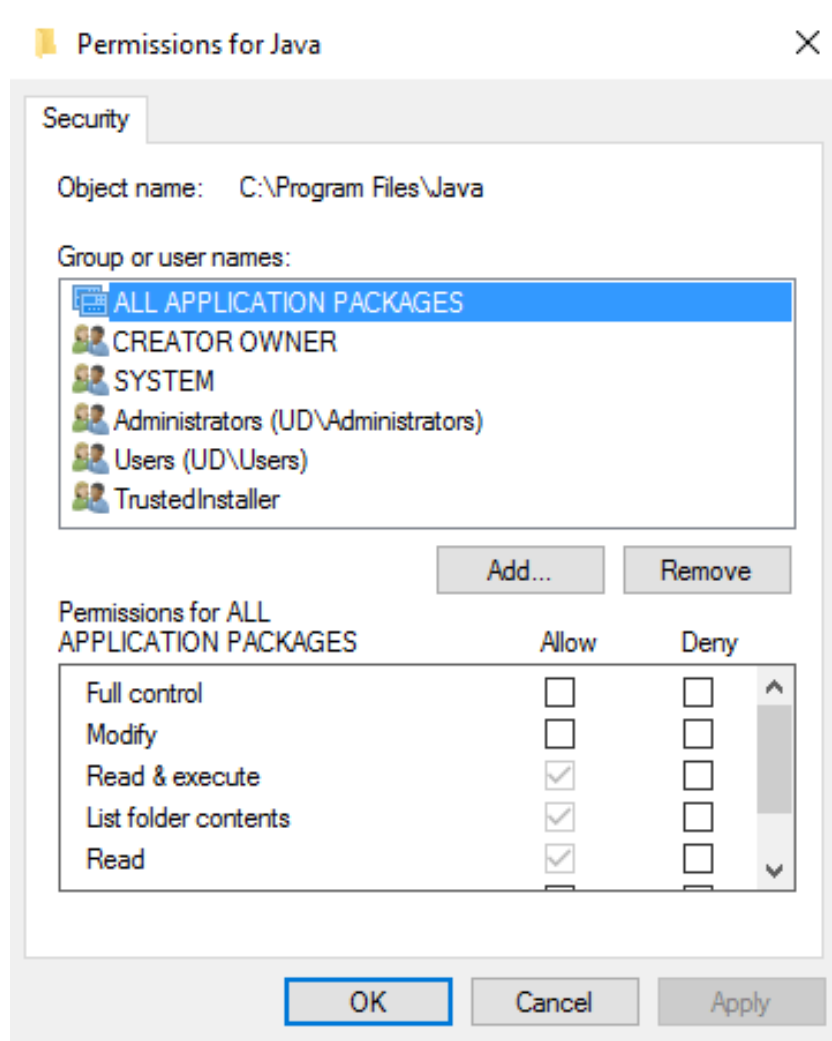


Name	Date modified	Type
CodeBlocks	06-01-2017 12:14	File folder
Common Files	18-01-2017 22:32	File folder
directx	19-12-2016 13:32	File folder
Google	10-12-2016 12:05	File folder
GreenTree Applications	08-01-2017 15:30	File folder
Hello	13-12-2016 23:17	File folder
Intel	07-12-2016 20:00	File folder
Internet Explorer	16-12-2016 22:50	File folder
Java	07-12-2016 23:02	File folder
Microsoft Office	07-01-2017 11:26	File folder
Microsoft Works	07-01-2017 11:26	File folder
Microsoft.NET	30-10-2015 11:18	File folder
Mozilla Firefox	16-12-2016 23:32	File folder
Mozilla Maintenance Service	16-12-2016 23:33	File folder
Realtek	07-12-2016 23:04	File folder
Windows Defender	10-12-2016 16:45	File folder
Windows Mail	10-12-2016 16:45	File folder
Windows Media Player	10-12-2016 16:45	File folder
Windows Multimedia Platform	10-12-2016 16:45	File folder
Windows NT	30-10-2015 11:18	File folder
Windows Photo Viewer	10-12-2016 16:45	File folder
Windows Portable Devices	10-12-2016 16:45	File folder
WindowsPowerShell	30-10-2015 11:18	File folder
WinRAR	19-12-2016 12:54	File folder

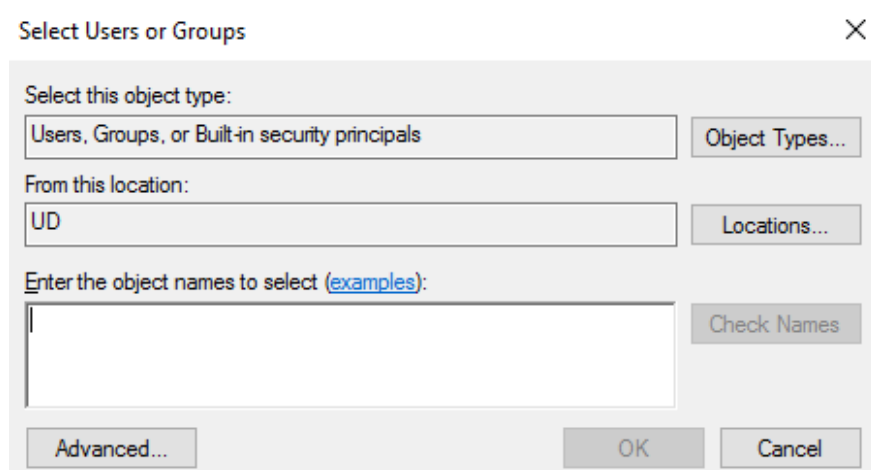
उसके बाद एक dialog box खुल जाएगा | उसके बाद properties पर जाए | उसके बाद security tab पर click करे |



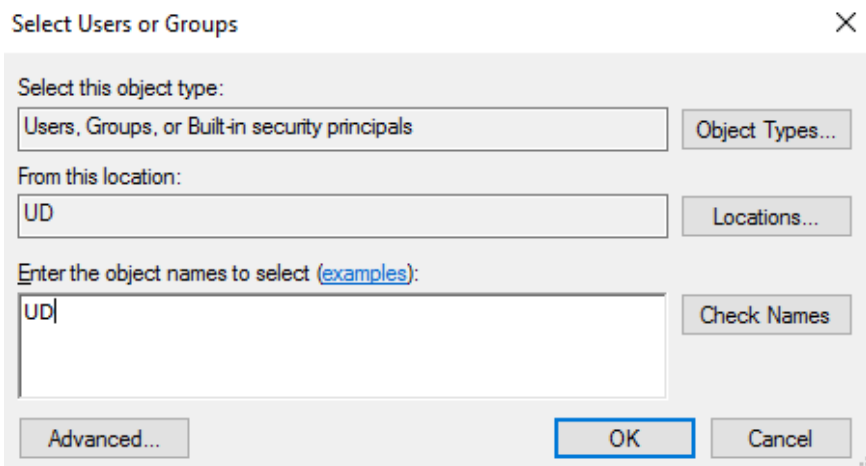
उसके बाद Edit button पर click करें



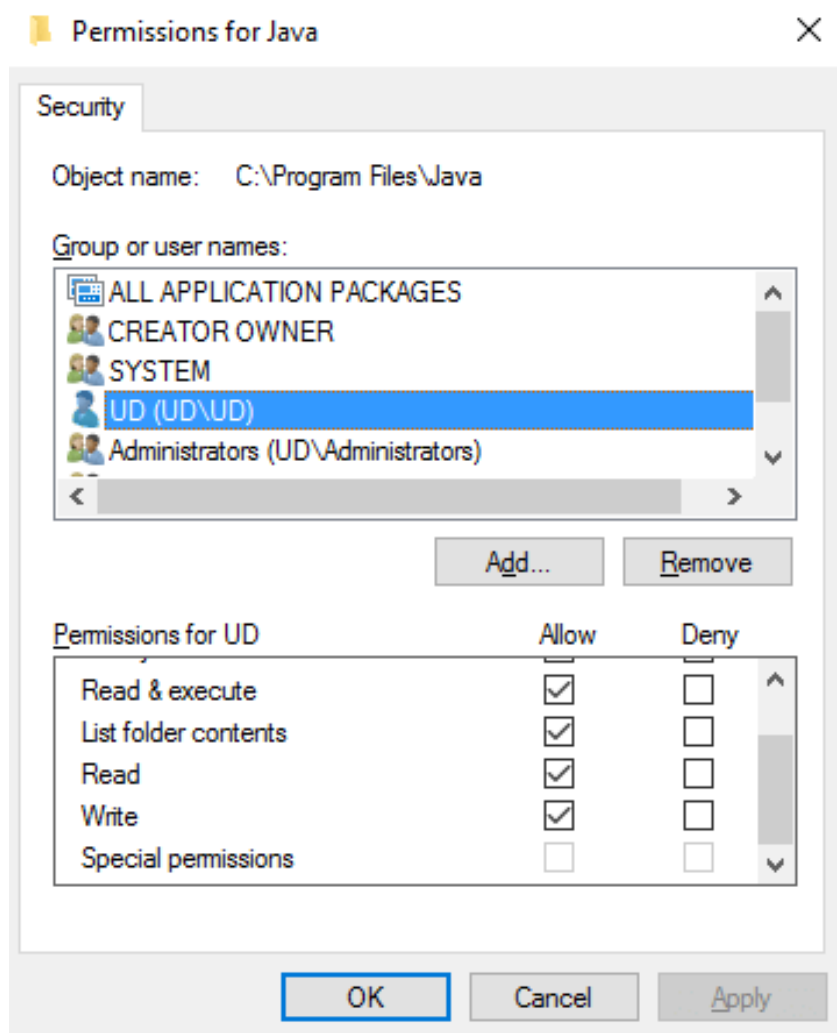
उसके बाद Add button पर click करें तो एक dialog box खुल जाएगा ।



वहाँ पर object को लिखिए । ये object का नाम computer का जो user का नाम होता है , वही दे ।

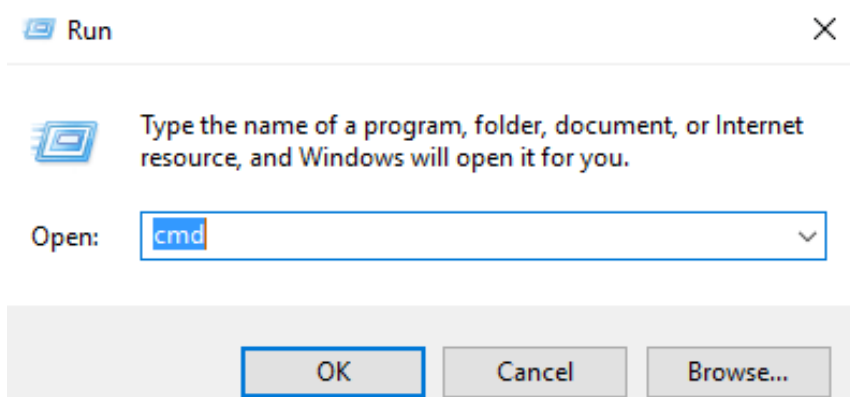


उसके बाद ok button पर click करे | अगर click किया जाए तो उसे write करने के लिए Allow को tick करे |



java का program; bin folder पर save किया जाता है |

Program को run करने के लिए command prompt को open करना पड़ता है | Command Prompt को open करने के लिए window+R दबाये उसके बाद cmd type करे |



उसके बाद ok button करे | click करने के बाद command prompt; open हो जाएगा |

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\UD>
```

Prompt; open होने के बाद उसे java के file का path दे |

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\UD>cd C:\Program Files\Java\jdk1.8.0_111\bin
C:\Program Files\Java\jdk1.8.0_111\bin>
```

बाद में javac के साथ filename और उसका extension दे |

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\UD>cd C:\Program Files\Java\jdk1.8.0_111\bin
C:\Program Files\Java\jdk1.8.0_111\bin>javac HelloProgram.java
C:\Program Files\Java\jdk1.8.0_111\bin>
```

उसके बाद java के साथ सिर्फ filename लिखे |

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\UD>cd C:\Program Files\Java\jdk1.8.0_111\bin
C:\Program Files\Java\jdk1.8.0_111\bin>javac HelloProgram.java
C:\Program Files\Java\jdk1.8.0_111\bin>java HelloProgram
Hello World!

C:\Program Files\Java\jdk1.8.0_111\bin>
```

# Reserved Keywords

Java के लिए 50 reserved keywords होते हैं। इन keywords को variables, classes और methods के नाम के लिए इस्तेमाल नहीं किया जाता।

Keywords	Description
abstract	इसका उपयोग Java Abstraction के लिए होता है।
assert	assert statement के लिए इस्तेमाल होता है। इससे debugging और testing में मदद होती है।
boolean	boolean के लिए 'true' और 'false' ये values हैं।
break	Loop को break करने के लिए इस्तेमाल किया जाता है।
byte	Numeric type के variable को declare करने के लिए इस्तेमाल होता है। ये integer के 8-bits hold करके रखता है।
case	Switch case statement को इस्तेमाल करने के लिए होता है।
catch	throw से exceptions handle करने के लिए इस्तेमाल होता है।
char	character variable declare करने के लिए इस्तेमाल होता है।
class	classes को declare करने के लिए इस्तेमाल होता है।
const	const ये एक स्थिर variable के लिए इस्तेमाल होता है।
continue	Loop को iterate किया जाता है।
default	Switch case statement के लिए इस्तेमाल होता है।
do	एक loop का प्रकार है। जिसके साथ while loop को इस्तेमाल किया जाता है।
double	floating-point data-type है।
else	if के साथ statement को इस्तेमाल किया जाता है।
enum	Enumeration data type का 'Keyword' है।
extends	Inheritance के लिए इस्तेमाल किया जाता है।
final	variable, class और method के साथ इस्तेमाल किया जाता है।
finally	try और catch के साथ exception handling के लिए इस्तेमाल किया जाता है।
float	Floating-point variable को declare करने के लिए इस्तेमाल होता है।
for	Loop का एक प्रकार है।

goto	एक statement है , जिसमे label होता है ।
if	एक statement है   जिससे condition सही है या गलत इसका पता चलता है ।
implements	interfaces को implements करने के लिए इस्तेमाल होता है ।
import	packages को import करने के लिए इस्तेमाल होता है ।
instanceof	instanceof operator का इस्तेमाल run-time पर object का type; check करने के लिए होता है ।
int	integer variable को declare करने के इस्तेमाल होता है ।
interface	ये एक pure abstarct class है ।
long	long integer variable को declare करने के इस्तेमाल होता है ।
native	ये एक method specifier है ।
new	class का object ये instance create करने के लिए इस्तेमाल होता है ।
package	packages को declare करने के लिए इस्तेमाल होता है ।
private	ये एक access spacificier है ।
protected	इसे inheritance के लिए इस्तेमाल किया जाता है ।
public	ये एक access specifier है ।
return	value को return करने के लिए इस्तेमाल किया जाता है ।
short	short integer variable को declare करने के इस्तेमाल होता है ।
static	Variable के method के लिए इस्तेमाल किया जाता है ।
switch	एक से अधिक condition का statement है ।
synchronized	class के एक ही thread को access करने के लिए इस्तेमाल होता है ।
this	current object को refer करने के लिए इस्तेमाल किया जाता है ।
throw	Exception Handling के लिए इस्तेमाल किया जाता है ।
throws	ये भी Exception Handling के लिए इस्तेमाल होता है ।
try	Exception Handling के लिए इस्तेमाल किया जाता है ।
void	ये कुछ भी return नहीं करता ।
while	ये एक loop का प्रकार है ।

## Primitive Data Types

### short

short data type का size 2 bytes का होता है ।  
Range  $-32,768$  ( $-2^{15}$ ) से  $32,767$  ( $2^{15}-1$ )  
Default value '0' होती है ।

Source Code :

```
//Sample.java
class Sample{
    public static void main(String[] args){

        short s = 32767;
        System.out.println(s);
    }
}
```

Output :

```
32767
```

### long

integer में सबसे बड़ा long integer data type है ।  
long data type का size 8 bytes होता है ।  
Range  $-9,223,372,036,854,775,808$  ( $-2^{63}$ ) से  $-9,223,372,036,854,775,807$  ( $2^{63}-1$ ) तक होती है ।  
Default value '0L' होती है ।

Source Code :

```
//Sample.java
class Sample{
    public static void main(String[] args){

        long l = 458L;
        System.out.println(l);
    }
}
```

Output :

458

## character

char data type का size 2 bytes होता है ।

Range '\u0000' या 0 से '\uffff' या 65535 तक होती है ।

char data type पर एक ही character होता है ।

character को single quote ( ' ' ) में लिखा जाता है ।

Source Code :

```
//Sample.java
class Sample{
    public static void main(String[] args){

        char c = 'A';
        System.out.println(c);
    }
}
```

Output :

A

character data type की size '2 bytes' क्यों होती है ?

C और C++ में char 1 byte का होता है । इसकी range -128 से 127 या unsigned(0 से 255) तक होती है । पर C और C++ में Unicode System के character support नहीं करता ।

Java में character के लिए unicode system होता है । ये सभी international languages के लिए होता है ।



# float

float data type का size 4 bytes होता है ।  
Range  $3.4e-038$  से  $3.4e+038$  तक होती है ।  
Default value '0.0f' होती है ।

Source Code :

```
//Sample.java
class Sample{
    public static void main(String[] args){

        float f = 0.8f;
        System.out.println(f);
    }
}
```

Output :

0.8

# double

double data type का size 8 bytes होता है ।  
Range  $1.7e-308$  से  $1.7e+038$  तक होती है ।  
Default value '0.0d' होती है ।

Source Code :

```
//Sample.java
class Sample{
    public static void main(String[] args){

        double d = 45.8d;
        System.out.println(d);
    }
}
```

Output :

45.8

# boolean

'boolean' keyword का इस्तेमाल किया जाता है ।

boolean data type के लिए 'true' और 'false' ये दो values होती है ।

Default value 'false' होती है ।

Source Code :

```
//Sample.java
class Sample{
    public static void main(String[] args){

        boolean b = true;
        System.out.println(b);
    }
}
```

Output :

```
true
```

## Default Values for Primitive Data Types

Source Code :

```
class Sample{
    static byte by;
    static int i;
    static short s;
    static long l;
    static char c;
    static double d;
    static float f;
    static boolean b;
    public static void main(String[] args) {
        System.out.println("Byte : " + by);
        System.out.println("Integer : " + i);
        System.out.println("Short : " + s);
        System.out.println("Long : " + l);
        System.out.println("Character : " + c);
        System.out.println("Double : " + d);
        System.out.println("Float : " + f);
        System.out.println("Boolean : " + b);
    }
}
```

Output :

```
Byte : 0  
Integer : 0  
Short : 0  
Long : 0  
Character :  
Double : 0.0  
Float : 0.0  
Boolean : false
```

## Non-Primitive Data Types

### Strings



Strings के पाठ में देखे

### Arrays



Arrays के पाठ में देखे

# Variables

## Variable Introduction

Variables ये memory locations के नाम होते हैं | जो values; variables को दी जाती हैं, वो उस location पर store हो जाती है |

### Syntax for Variable Declaration

```
data_type_name variable_name; //or  
data_type_name variable_name1, variable_name2;
```

For Example,

```
int a;  
int b, c;
```

### Syntax for Variable Definition

```
data_type variable_name = variable_value;
```

For Example,

```
int a = 5;  
int b = 10, c = 15;
```

## Types of Variables

Java के लिए Variables के तीन प्रकार होते हैं |

1. Local Variable
2. Instance Variable
3. Static Variable

### 1. Local Variable

Local Variables block, methods और constructor के अन्दर होते हैं |

Local Variable का scope; local होता है | ये सिर्फ methods और constructor के अन्दर visible होते हैं |

जब Local Variables; methods और constructor के बाहर जाते हैं, तब destroyed हो जाते हैं |

Source Code :

```
class Sample{

    void display(){
        int a = 5; //Local Variable
        System.out.println("Value of a : " + a);
    }

    public static void main(String arg[]){
        Sample s = new Sample();
        s.display();
    }
}
```

Output :

```
Value of a : 5
```

## 2. Instance Variable

Instance Variables; class के अन्दर होते हैं और methods और constructor के बाहर होते हैं ।  
Instance Variables non-static variables होते हैं ।

Source Code :

```
class Sample{
    int a = 5; //Instance Variable
    void display(){
        System.out.println("Value of a : " + a);
    }

    public static void main(String arg[]){
        Sample s = new Sample();
        s.display();
    }
}
```

Output :

```
Value of a : 5
```

### 3. Static Variable

Static Variables को Class Variables भी कहते हैं।

ये Instance Variable के तरह class के अन्दर और methods और constructor के बाहर होते हैं।

'static' keyword के साथ इनका इस्तेमाल किया जाता है।

Source Code :

```
class Sample{
    static int a = 5; //Static Variable
    void display(){
        System.out.println("Value of a : " + a);
    }
    public static void main(String arg[]){
        Sample s = new Sample();
        s.display();
    }
}
```

Output :

```
Value of a : 5
```

# Operators

## Arithmetic Operators

Operators	Explanation
+ (Addition)	ये दो Operands को add करता है ।
- (Subtraction)	ये right operand से left operand को निकाल देता है ।
* (Multiplication)	ये दो Operands को multiply करता है ।
/ (Division)	ये right operand द्वारा left operand को divide करता है ।
% (Modulus)	ये right operand द्वारा left operand को divide करके remainder निकालता है ।

Source Code :

```
//Sample.java
class Sample{

public static void main(String args[]) {
int a = 10, b = 5, c;
c=a + b;
    System.out.println("Addition of a and b is " + c);
c=a - b;
    System.out.println("Subtraction of a and b is " + c);
c=a * b;
    System.out.println("Multiplication of a and b is " + c);
c=a / b;
    System.out.println("Division of a and b is " + c);
c=a % b;
    System.out.println("Remainder of a and b is " + c);
}
}
```

Output :

```
Addition of a and b is 15
Subtraction of a and b is 5
Multiplication of a and b is 50
Division of a and b is 2
Remainder of a and b is 0
```

# Relational Operators

Operators	Explanation
< (less than)	एक Operand की value दूसरे Operand से कम हो तो ये true return करता है । for eg. num1=5; num2=6; num1 < num2
> (greater than)	एक Operand की value दूसरे Operand से ज्यादा हो तो ये true return करता है । for eg. num1=6; num2=5; num1 > num2
<= (less than or equal to)	एक Operand की value दूसरे Operand से कम हो या बराबर (equal) हो तो ये true return करता है । for eg. num1=5; num2=5; num1 <= num2
>= (greater than or equal to)	एक Operand की value दूसरे Operand से ज्यादा हो या बराबर (equal) हो तो ये true return करता है । for eg. num1=5; num2=5; num1 >= num2
== (equal to)	दो Operands जब बराबर(equal) होते हैं, तब ये true return करता है ।
!= (not equal to)	दो Operands जब एक-दूसरे से अलग होते हैं, तब ये true return करता है ।

Source Code :

```
//Sample.java
class Sample{
public static void main(String args[]){
int a = 6, b = 5;
if(a < b){
    System.out.println("a is less than b");
}
else{
    System.out.println("a is greater than b");
}
```



```
if(a <= b){  
    System.out.println("a is less than b");  
}  
else{  
    System.out.println("a is greater than b");  
}  
if(a > b){  
    System.out.println("a is greater than b");  
}  
else{  
    System.out.println("a is less than b");  
}  
if(a >= b){  
    System.out.println("a is greater than b");  
}  
else{  
    System.out.println("a is less than b");  
}  
if(a == b){  
    System.out.println("a is equal to b");  
}  
else{  
    System.out.println("a is not equal to b");  
}  
}  
}
```

Output :

```
a is greater than b  
a is greater than b  
a is greater than b  
a is greater than b  
a is not equal to b
```

# Logical Operators

Operators	Explanation
&& (logical &&)	अगर दोनों conditions true हो तो ये true return करेगा   for eg. (5<6) && (6>5)
(logical OR)	अगर दोनों में से एक भी true है , तो ये true return करेगा   for eg. (5<6)    (6>5)
! (logical not)	अगर condition true हो तो ये उसे false कर देता है   for eg. !((5<6) && (6>5)) ! ((5<6)    (6>5))

Source Code :

```
//Sample.java
class Sample{
public static void main(String args[]){

if((5 < 6) && (6 > 5)){
    System.out.println("Condition is true.");
}
else{
    System.out.println("Condition is false.");
}
if((5 < 6) || (6 > 5)){
    System.out.println("Condition is true.");
}
else{
    System.out.println("Condition is false.");
}
if(!((5 < 6) && (5 > 6))){
    System.out.println("Condition is true.");
}
else{
    System.out.println("Condition is false.");
}
}
}
```

Output :

```
Condition is true.
Condition is true.
Condition is true.
```

# Bitwise Operators

Truth Table for &, |, ^

a	b	a & b	a   b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

## Operation on AND (a&b)

अगर a = 20, b = 12 हो तो,

Decimal Value	Binary Value
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
4	0 0 0 0 0 1 0 0

## Operation on AND (a&b)

अगर a = 20, b = 12 हो तो,

Decimal Value	Binary Value
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
28	0 0 0 1 1 1 0 0

## Operation on XOR(a^b)

अगर a = 20, b = 12 हो तो,

Decimal Value	Binary Value
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
24	0 0 0 1 1 0 0 0

## Binary Left Shift( << ) and Right Shift( >> )

Left Shift(<<) for e.g. a=20; /\* 0001 0100 \*/ a << 2 में numeric value के binary value में हर binary number को 2 binary numbers left side से shift करता है | for e.g.a=20; /\* 0001 0100 \*/ तो इसका 0101 0000 मतलब 80 हो जायेगा |

Right Shift(>>) for e.g. a=20; /\* 0001 0100 \*/ ये Left shift से बिल्कुल उलट है | Right Shift a>> 2 में numeric value के binary value में हर binary number को 2 binary numbers right side से shift करता है | for e.g.a=20; /\* 0001 0100 \*/ तो इसका 0000 0101 मतलब 5 हो जायेगा |

## Complement Operator (~)

### Operation on Complement( ~ )

Decimal Value	Binary Value
~20	0 0 0 0 1 1 0 0
243	1 1 1 1 0 0 1 1

यहाँ पर Output -13 आने के बजाय 243 आया ऐसा क्यों ?

2's Complement of 243 -(reverse of 243 in binary + 1)

### Operation on 2's Complement ( ~ )

Decimal Value	Binary Value	2's Complement
243	1111 0011	-(0000 1100+1) = -(0000 1101) = -13(output)

Source Code :

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

int a=20; /* 0001 0100 */
int b=12; /* 0000 1100 */
int c;
c=a&b;
System.out.println("value of c is " + c); /* 4 = 0000 0100 */
c=a|b;
System.out.println("value of c is " + c); /* 28 = 0001 1100 */
c=a^b;
System.out.println("value of c is " + c); /* 24 = 0001 1000 */
c=a<<2;
System.out.println("value of c is " + c); /* 80 = 0101 0000 */
c=a>>2;
System.out.println("value of c is " + c); /* 5 = 0000 0101 */
}
}
```

Output :

```
value of c is 4
value of c is 28
value of c is 24
value of c is 80
value of c is 5
```

# Assignment Operators

Assignment Operators ग्यारह प्रकार के होते हैं।

Operators	Example
= (assignment)	c = a + b
+= (add assignment)	c += a same as c = c + a
-= (subtract assignment)	c -= a same as c = c - a
= (multiply assignment)	c *= a same as c = c * a
/= (divide assignment)	c /= a same as c = c / a
%= (modulus assignment)	c %= a same as c = c % a
&= (AND assignment)	c &= a same as c = c & a
= (OR assignment)	c  = a same as c = c   a
^= (XOR assignment)	c ^= a same as c = c ^ a
<<= (Left Shift assignment)	c <<= a same as c = c << a
>>= (Right Shift assignment)	c >>= a same as c = c >> a

Source Code :

```
class Sample{
public static void main(String args[]){
int a=20, b=12;
b = a + b;
    System.out.println("value of b is " + b);
b += a;
    System.out.println("value of b is " + b);
b -= a;
    System.out.println("value of b is " + b);
b *= a;
    System.out.println("value of b is " + b);
b /= a;
    System.out.println("value of b is " + b);
b %= a;
    System.out.println("value of b is " + b);
b &= 2;
    System.out.println("value of b is " + b);
b |= 2;
    System.out.println("value of b is " + b);
b ^= 2;
    System.out.println("value of b is " + b);
b <<= 2;
    System.out.println("value of b is " + b);
b >>= 2;
    System.out.println("value of b is " + b);
}
}
```

Output :

```
value of b is 32
value of b is 52
value of b is 32
value of b is 640
value of b is 32
value of b is 12
value of b is 0
value of b is 2
value of b is 0
value of b is 0
value of b is 0
```

## Increment and Decrement Operators

**Increment Operator (++)** ये variable की value 1 से बढ़ा देता है ।

**Decrement Operator (--)** ये variable की value 1 से घटा देता है ।

Operators
++a (Increment Prefix)
--a (Decrement Prefix)
a++ (Increment Postfix)
a-- (Decrement Postfix)

Source Code :

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

int a=20;
    System.out.println("Print Value with prefix : " + ++a); // increase value with increment
prefix
    System.out.println("Value of a : " + a);
    System.out.println("Print Value with prefix : " + --a); // decrease value with
decrement prefix
    System.out.println("Value of a : " + a);
    System.out.println("Print Value with postfix : " + a++); // increase value with
increment postfix
    System.out.println("Value of a : " + a);
    System.out.println("Print Value with postfix : " + a--); // decrease value with
decrement postfix
    System.out.println("Value of a : " + a);
}
}
```

Output :

```
Print Value with prefix : 21
Value of a : 21
Print Value with prefix : 20
Value of a : 20
Print Value with postfix : 20
Value of a : 21
Print Value with postfix : 21
Value of a : 20
```

## Conditional Operators

Conditional Operators में तीन Expressions होते हैं।

Conditional Operators को Ternary Operator भी कहते हैं।

Conditional Operators में अगर पहला expression true होता है, तो वो दूसरा expression output में print करता है।

अगर Conditional Operators में पहला expression false होता है, तो वो तीसरा expression output में print करता है।

Syntax:

```
expression1 ? expression 2 : expression 3
```

Source Code:

```
class Sample{
public static void main(String args[]){
int a = 100, b ;
b = ( a == 100 ? 2 : 0 ) ;
    System.out.println("Value of a is " + a);
    System.out.println("Value of b is " + b);
}
}
```

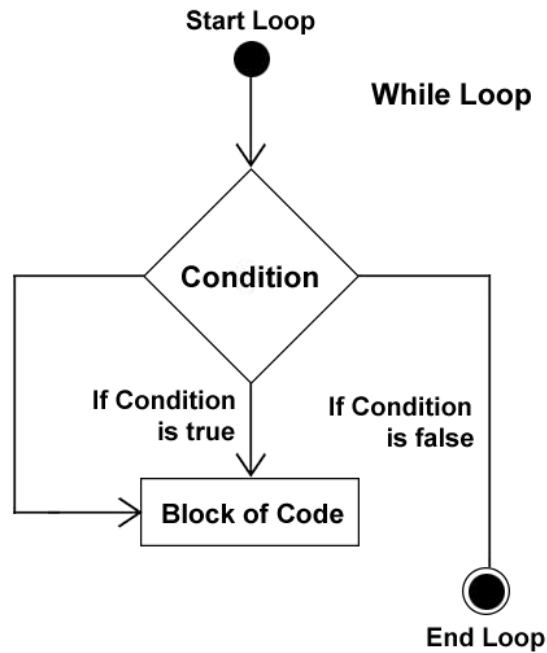
Output:

```
Value of a is 100
Value of b is 2
```

# Loops

## While Loop

while Loop जब तक condition true होती है तब तक loop repeat होता रहता है | अगर condition false होता है, तब Loop exit हो जाता है |



Syntax:

```
variable_initialization;
while( condition ){
    //statement(s);
    increment/decrement;
}
```

Source Code:

```
//WhileLoop.java
public class WhileLoop{
    public static void main(String args[]) {
        int i = 0;
        while(i < 10){
            System.out.println("Value of i is " + i);
            i++;
        }
    }
}
```

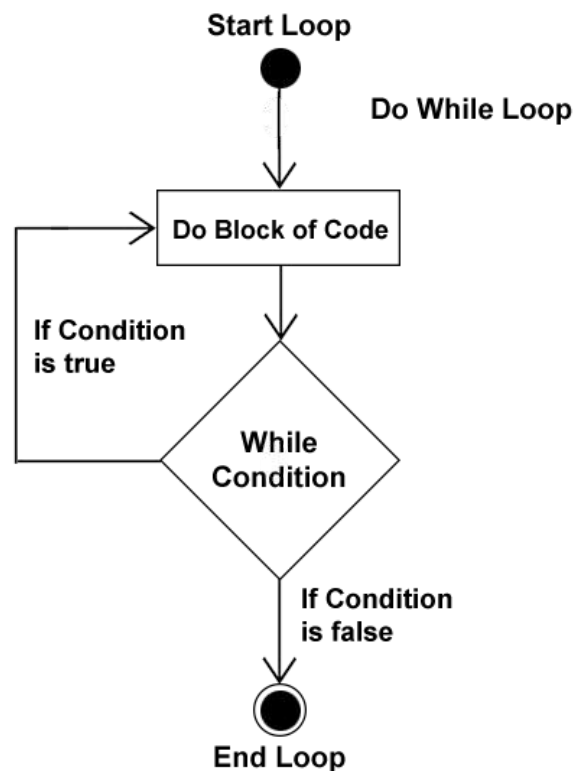


Output:

```
Value of i is 0  
Value of i is 1  
Value of i is 2  
Value of i is 3  
Value of i is 4  
Value of i is 5  
Value of i is 6  
Value of i is 7  
Value of i is 8  
Value of i is 9
```

## Do While Loop

do\_while ये Loop; while Loop के जैसा ही है, लेकिन ये loop अगर condition true होती है तब do\_while loop चलता है और condition false होती है तब सिर्फ do का statement execute हो जाता है ।



Syntax:

```
variable_initialization;  
do{  
  //statement(s);  
  increment/decrement;  
}  
while( condition );
```

Source Code:

```
//DoWhileLoop.java
public class DoWhileLoop{
    public static void main(String args[]){
        int i = 0;
        do{
            System.out.println("Value of i is " + i);
            i++;
        }
        while(i < 10);
    }
}
```

Output:

```
Value of i is 0
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
Value of i is 5
Value of i is 6
Value of i is 7
Value of i is 8
Value of i is 9
```

## For Loop

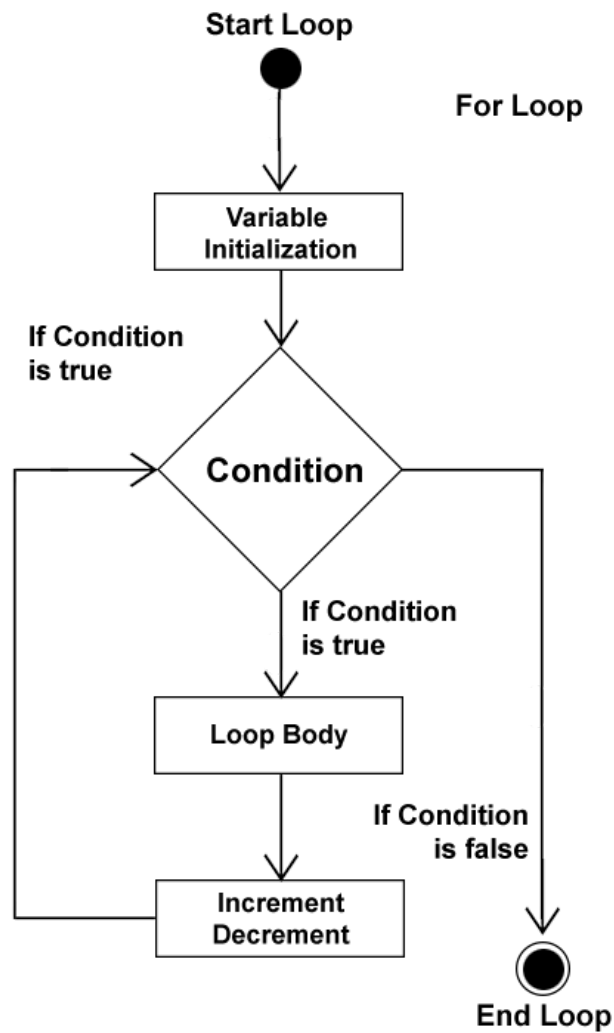


Core Java में for Loop के लिए दो प्रकार हैं ।

- Normal for Loop
- Enhanced or foreach Loop

## Normal for Loop

for loop में दिए हुए condition तक statement repeat होता रहता है ।



Syntax:

```
for( initialization; condition; increment/decrement ){  
    //statements;  
}
```

Source Code:

```
public class ForLoop{  
    public static void main(String args[]){  
        for( int i=0; i < 10; i++ ){  
            System.out.println("Value of i is " + i);  
        }  
    }  
}
```

## Enhanced or Foreach Loop

array के elements की value को print करने के लिए foreach का इस्तेमाल किया जाता है, लेकिन ये array के मामले में for loop से बेहतर रहता है ।

Syntax:

```
for(variable_name : array_name){  
    //statement(s);  
}
```

Source Code:

```
//ForandForeachLoop.java  
public class ForandForeachLoop{  
    public static void main(String args[]){  
        int[] arr = {10, 15, 20, 25, 30};  
        System.out.println("Using for Loop");  
        for( int i=0; i<arr.length; i++ ){  
            System.out.println("Value of arr is " + arr[i]);  
        }  
        System.out.println("\nUsing foreach Loop");  
        for(int a : arr){  
            System.out.println("Value of arr is " + a);  
        }  
    }  
}
```

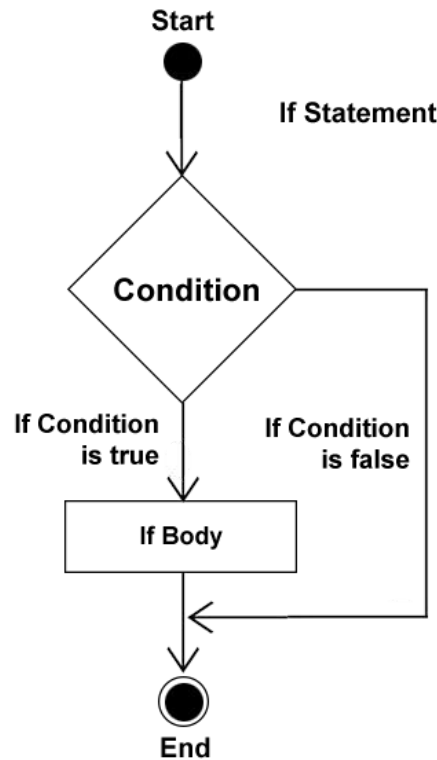
Output :

```
Using for Loop  
Value of arr is 10  
Value of arr is 15  
Value of arr is 20  
Value of arr is 25  
Value of arr is 30  
  
Using foreach Loop  
Value of arr is 10  
Value of arr is 15  
Value of arr is 20  
Value of arr is 25  
Value of arr is 30
```

# Control Statements

## If Statement

if Statement में अगर Condition true होती है तब Statement Execute होता है ।



Syntax:

```
if(condition){  
    statement(s);  
}
```

Source Code:

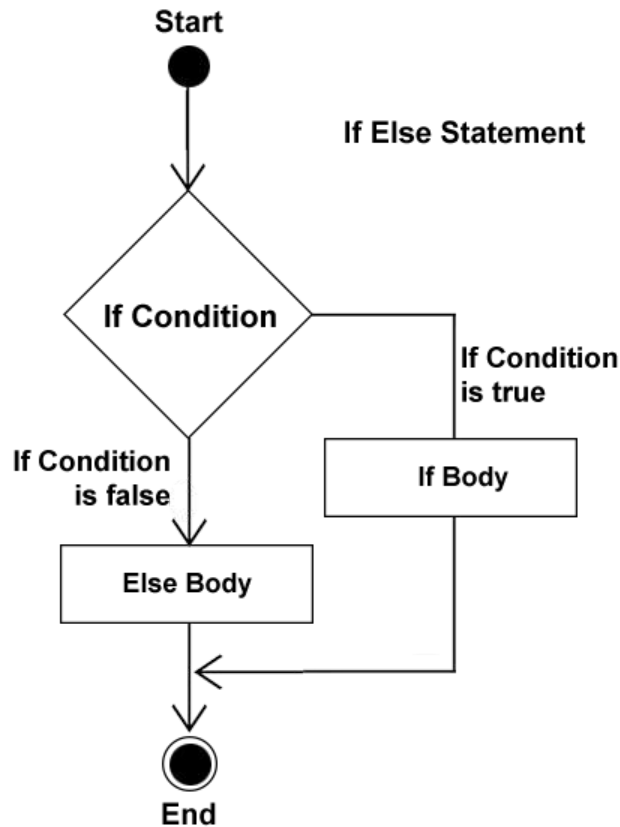
```
class Myif  
{  
    public static void main(String args[])  
    {  
        int x = 3;  
        if (x > 10){  
            System.out.println("Inner If");  
        }  
        System.out.println("Outer If");  
    }  
}
```

Output:

Outer If

# If Else Statement

if else Statement में अगर Condition true हो तो वो if का statement Execute करता है | अगर Condition false होती है तो else का Condition करता है |



Syntax:

```
if(condition){  
    statement(s);  
}else{  
    statement(s);  
}
```

Source Code:

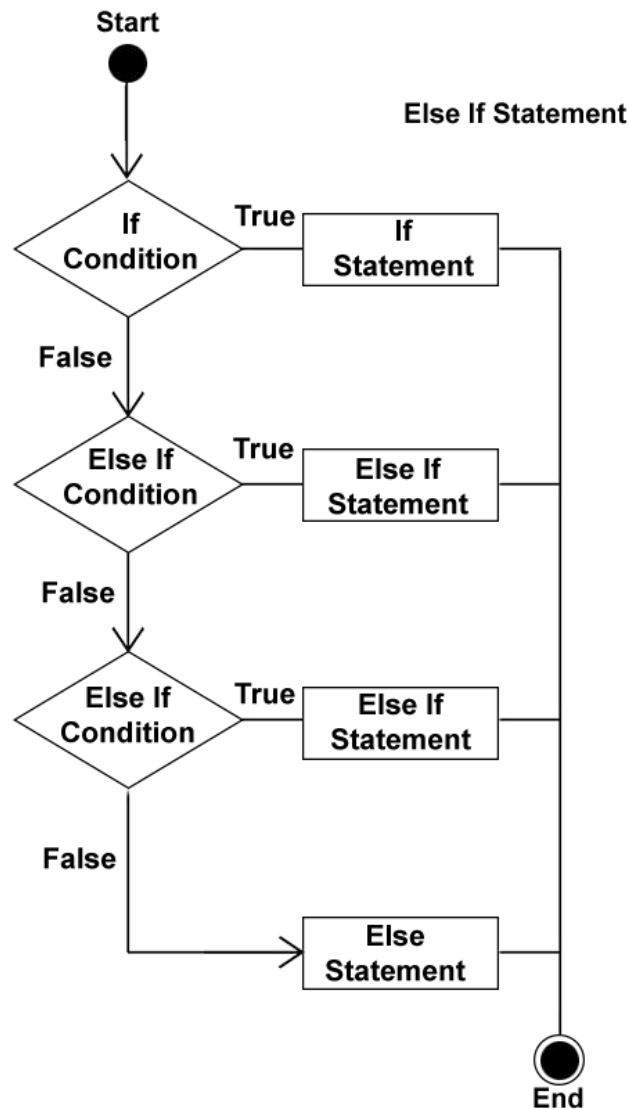
```
class MyifElse  
{  
    public static void main(String args[])  
    {  
        int x = 3;  
        if (x > 10){  
            System.out.println("Inner If");  
        }else{  
            System.out.println("Inner Else");  
        }  
        System.out.println("Outer If Else");  
    }  
}
```

Output:

Inner Else  
Outer If Else

## Else If Statement

else if Statement में अगर if की Condition true होती है तो if का statement execute होता है | अगर if का condition false होता है तो वो अगले condition पर जाकर check करता है | अगर वो condition true होता है तो वो उसका statement execute करता है | अगर कोई भी condition true नहीं होती तो वो else का statement execute करता है |



Syntax:

```
if(condition){  
    statement(s);  
}else if(condition){  
    statement(s);  
}else{  
    statement(s);  
}
```

Source Code:

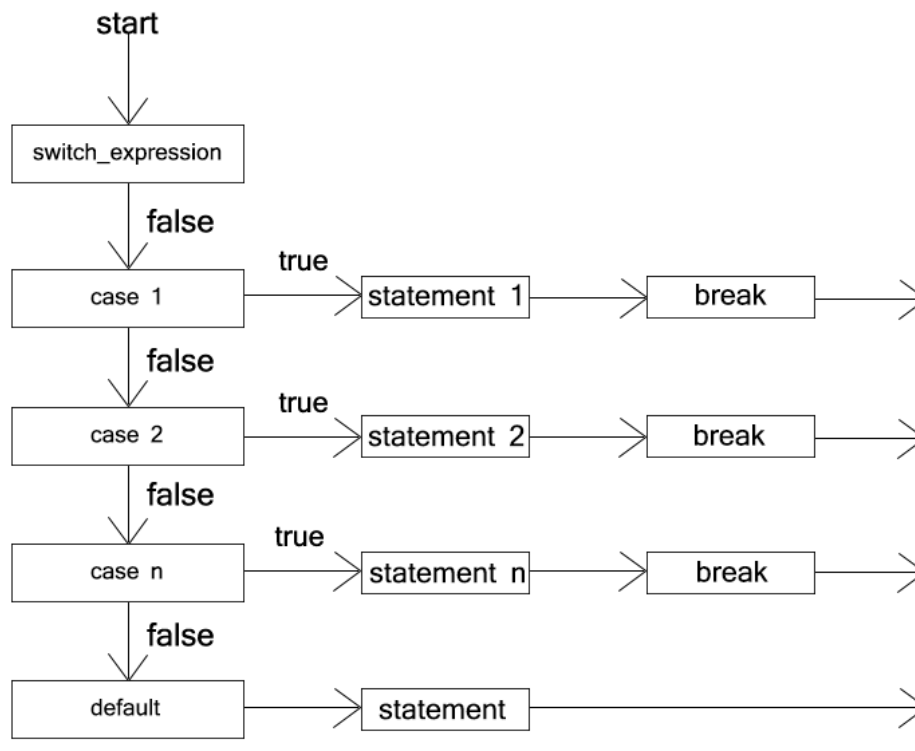
```
class MyElseIf
{
    public static void main(String args[])
    {
        int x = 3;
        if (x > 10){
            System.out.println("x is greater than 10");
        }else if (x < 10){
            System.out.println("x is less than 10");
        }else{
            System.out.println("x is equal to 10");
        }
        System.out.println("Outer Else If");
    }
}
```

Output:

```
x is less than 10
Outer Else If
```

## Switch Case Statement

Switch case statement में expression होता है और उससे related कुछ cases होते हैं। जो case उस expression या declare किये हुए variable से match होती है तब वो output में print होता है। अगर कोई भी case expression से match नहीं होती तो वो default का statement output में print करेगा। आपको हर statement के बाद break लगाना पड़ता है, इसका मतलब वो उसके पहले का statement ही print करेगा। अगर आप break नहीं लगाते तो वो पहला और दूसरा ये दोनों statement को print करेगा। default case के बाद break नहीं लगाते।





### Syntax:

```
switch (expression){  
case value1 :  
statement1 ;  
break;  
case value2 :  
statement2 ;  
break;  
case valueN :  
statementN ;  
break;  
default :  
statement3 ;  
}
```

### Source Code:

```
class MySwitch{  
public static void main(String args[]){  
int x = 1;  
switch(x){  
case 0:  
System.out.println("x is equal to 0.");  
break;  
  
case 1:  
System.out.println("x is equal to 1.");  
break;  
  
case 2:  
System.out.println("x is equal to 2.");  
break;  
default:  
System.out.println("x is not found.");  
break;  
}  
}  
}
```

### Output:

```
x is equal to 1.
```

# Break Statement

Break Statement Program के loops और switch case के execution का काम किसी condition पर बंद कर देता है |

Syntax:

```
break;
```

Source Code:

```
class MyBreak{
    public static void main(String args[]){
        int i=0;
        while ( i < 20 ){
            System.out.println("value of i is %d\n", i);

            i++;
            if ( i == 10 ){
                break;
            }
        }
    }
}
```

Output:

```
value of i is 0
value of i is 1
value of i is 2
value of i is 3
value of i is 4
value of i is 5
value of i is 6
value of i is 7
value of i is 8
value of i is 9
```

# Continue Statement

Continue Statement Program के loops के condition के हिसाब से बीचवाले statements को skip कर देता है और बादवाले statement को execute करता है ।

Syntax:

```
continue;
```

Source Code:

```
class MyContinue{
    public static void main(String args[]){
        int i;
        for(i=0; i<10; i++){
            if(i==7){

                System.out.println("Number "+ i +" is skipped.");

                continue;
            }
            System.out.println(i);
        }
    }
}
```

Output:

```
0
1
2
3
4
5
6
Number 7 is skipped.
8
9
```

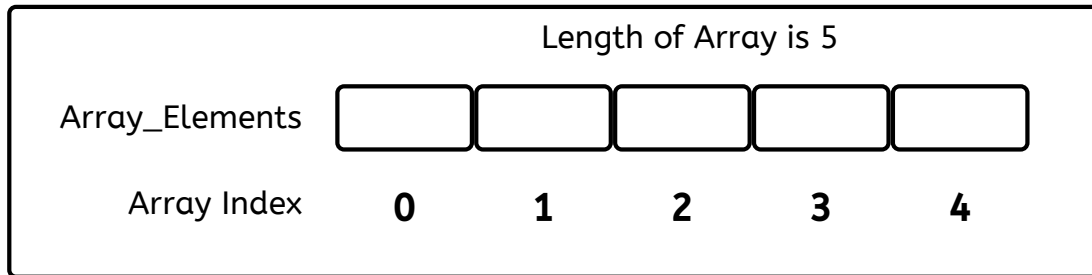
# Arrays

## Array Introduction

Java के लिए Array एक Non-Primitive Data है | Java के array में एक ही data type के अनेक elements होते हैं | लेकिन ये elements अपने पास-पास के memory locations पर होते हैं |

Java Array ये primitive data types के variables अपने अन्दर रखता है |

Array एक object होता है | जो एक ही primitive data type के variables रखता है | Array की size fixed होती है |



Array के तीन methods हैं |

- Array Declaration
- Creating Array
- Array Initialization

### Syntax for Array Declaration

Array का declaration दो प्रकार से किया जाता है |

```
data_type array_name[]; //or  
data_type[] array_name;
```

Syntax में पहला प्रकार है वो java के लिए इस्तेमाल किया जाता है |

Syntax में दूसरा प्रकार है वो C और C++ में इस्तेमाल किया जाता है | ये java में भी चलता है, लेकिन इसका इस्तेमाल नहीं किया जाता |

### Example for Array Declaration

Array को जब declare किया जाता है, तब उसके subscript([]) में उसका size नहीं दिया जाता |

```
int[] arr;  
char[] str;  
double[] arr;
```

### Syntax for Creating Array

```
data_type[] array_name = new data_type[array_size]; //declaring and creating Array
```

## Example for Creating Array

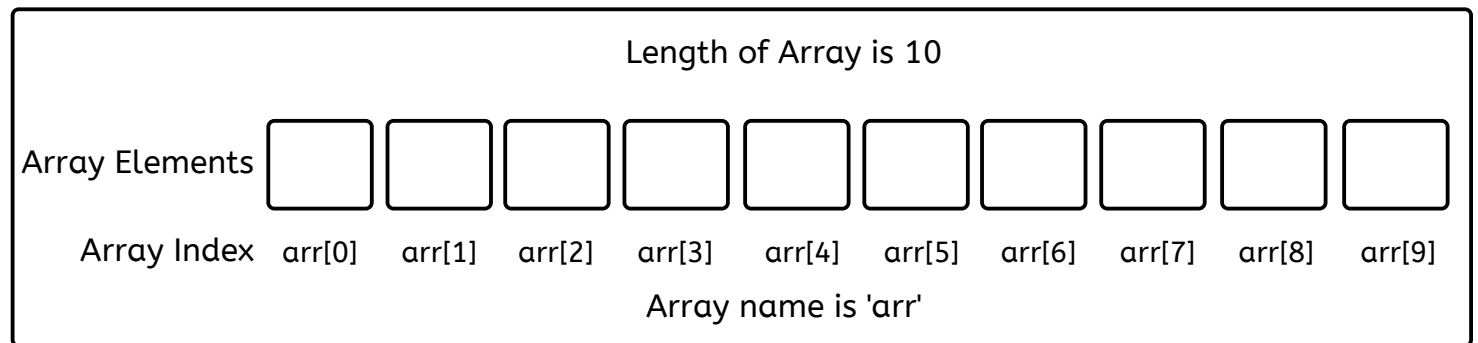
जब Array को create करना हो तो new operator का इस्तेमाल किया जाता है | यहाँ पर array का size दिया जाता है |

निचे दिए हुए example में दो प्रकार से 'arr' इस नाम के array को declare और create किया गया है |

पहले प्रकार में 'arr' इस array को declaration के साथ create भी किया गया है |

दुसरे प्रकार में 'arr' इस array को पहले declare किया है और बाद में उसको create किया गया है |

```
int[] arr = new int[10]; // declaring and creating Array
// or
int[] arr;      //declaring Array
arr = new int[10]; //Creating Array
```



## Syntax for Array Initialization

```
data_type[] array_name = {array_elements};
```

यहाँ पर array को एक ही line में initialize किया गया है | इस array के initialization में new operator की जरूरत नहीं पड़ती | जैसे-जैसे elements में value को दिया जाता है | वैसे-वैसे array की size भी बढ़ जाती है |

## Example for Array Initialization

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

यहाँ पर array को seperatly initialize किया गया है |

## Another Example for Array Initialization

```
arr[0] = 1;    // Array initialization index by index
arr[1] = 2;
arr[2] = 3;
arr[3] = 4;
arr[4] = 5;
```

## Accessing Array Elements

Array के elements को दो प्रकार से access किया जाता है ।

- 1.By Indexing
- 2.By for या foreach Loop

### 1. By Indexing : Accessing Array Elements

```
class Sample{

    public static void main(String args[]){
int[] arr = {1, 2, 3, 4, 5}; // Array initialization
System.out.println("arr[0] = " + arr[0]);
System.out.println("arr[1] = " + arr[1]);
System.out.println("arr[2] = " + arr[2]);
System.out.println("arr[3] = " + arr[3]);
System.out.println("arr[4] = " + arr[4]);
    }
}
```

Output:

```
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
```

### 2. By for या foreach Loop : Accessing Array Elements

```
class Sample{

    public static void main(String args[]){
int[] arr = {1, 2, 3, 4, 5}; // Array initialization
int i;
for(i=0; i<arr.length; i++){
System.out.println("arr[" + i + "] = " + arr[i]);
    }
}
}
```

Output:

```
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
```

### Using Foreach Loop

```
class Sample{
public static void main(String args[]){
int[] arr = {1, 2, 3, 4, 5};
for(int i : arr){
System.out.println("arr[" + (i-1) + "] = " + i);
}
}
}
```

Output:

```
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
```

## Types of Arrays

### Array के दो प्रकार होते है |

1. One-Dimensional Array
2. Two-Dimensional or Multi-Dimensional Array

## 1. One-Dimensional Array

ऊपर दिए हुए Array Declaration, Creating Array, Array Initialization और Accessing Array ये सभी 'One Dimensional Array' के है |

## 2. Two-Dimensional or Multi-Dimensional Array

### Multi-Dimensional Array Declaration

```
int arr[][]; //or  
int[][] arr; //or  
int[] arr[];
```

### Create Multi-Dimensional Array

```
int arr[][] = new int[2][2]; //or  
int[][] arr = new int[2][2]; //or  
int[] arr[] = new int[2][2];
```

### Multi-Dimensional Array Initialization

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

Multi-Dimensional Array के elements को दो प्रकार से access किया जाता है ।

1. By Indexing
2. By for Loop

#### 1. By Indexing : Accessing Array Elements

```
class Sample{  
  
    public static void main(String args[]){  
int[][] arr = {{1, 2},{3, 4}}; // Array initialization  
System.out.println("arr[0][0] = " + arr[0][0]);  
System.out.println("arr[0][1] = " + arr[0][1]);  
System.out.println("arr[1][0] = " + arr[1][0]);  
System.out.println("arr[1][1] = " + arr[1][1]);  
}  
}
```

Output:

```
arr[0][0] = 1  
arr[0][1] = 2  
arr[1][0] = 3  
arr[1][1] = 4
```



## 2. By for Loop : Accessing Array Elements

Using for Loop

```
class Sample{

    public static void main(String args[]){
int[][] arr = {{1, 2},{3, 4}}; // Array initialization
int i, j;
for(i=0; i<arr.length; i++){
for(j=0; j<arr.length; j++){
System.out.println("arr[" + i + "][" + j + "] = " + arr[i][j]);
}
}
}
}
```

Output:

```
arr[0][0] = 1
arr[0][1] = 2
arr[1][0] = 3
arr[1][1] = 4
```

## What is Strings

String ये Java के लिए एक 'Non-primitive' Data Type होता है | String; एक से अधिक characters का group होता है | String को double quotes(" ") के अन्दर लिखा जाता है | String ये class; java.lang इस package में define होता है | C और C++ में String के लिए character data type और array का इस्तेमाल किया जाता था | लेकिन Java में String के object को create किया जाता है | Java में भी C और C++ के जैसे String को create किया जाता है |

Source Code :

```
public class Sample{
    public static void main(String args[]){
        char str1[] = {'H','e','l','l','o',' ','W','o','r','l','d'};
        String s = new String(str1);
        //both are same.
        String str2 = "Hello World";

        System.out.println("String 1 : " + s);
        System.out.println("String 2 : " + str2);
    }
}
```

Output:

```
String 1 : Hello World
String 2 : Hello World
```

String Literal double quotes(" ") में close होता है |

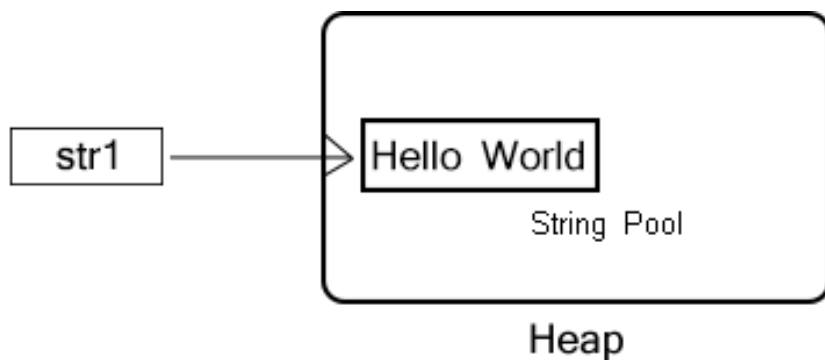
For eg.

```
String str = "Hello World";
```

## String Object; Store कैसे होता है ?

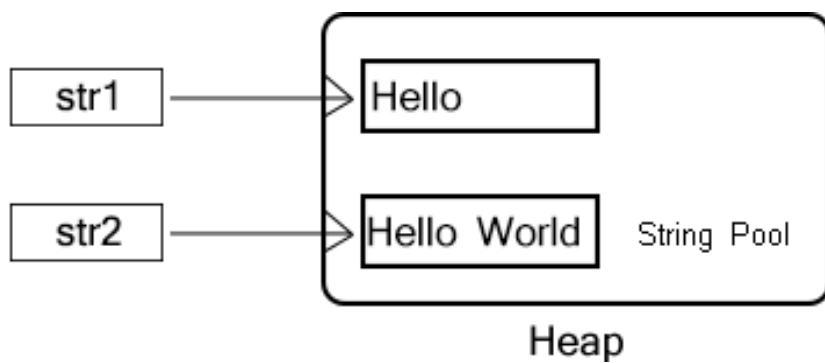
जब String object के साथ string literal इस्तेमाल होता है, तब string literal; string pool पर store होता है | String Pool ये String Object को store करने के खास memory है |

```
String str1 = "Hello World";
```



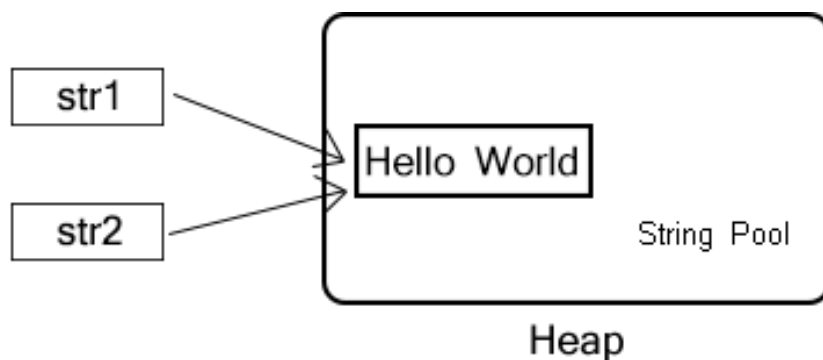
निचे दिए हुए program में string pool पर दो अलग-अलग object store किये गए हैं।

```
String str1 = "Hello";  
String str2 = "Hello World";
```



एक जैसे String के साथ जब दो object बनाये जाते हैं, तब दोनों object को एक ही string literal को string pool से return किया जाता है।

```
String str1 = "Hello World";  
String str2 = "Hello World";
```



# String Methods

Return type and String Methods	Description
boolean contentEquals(StringBuffer strbuff)	String से string buffer को compare किया जाता है और ये method boolean value को return करता है ।
boolean endsWith(String suffix)	String में end के suffix को boolean value में return किया जाता है ।
boolean equals(Object obj)	दो string को compare किया जाता है और boolean value; return होती है ।
boolean equalsIgnoreCase(String str)	दो string को compare किया जाता है और boolean value; return होती है ।
boolean matches(String regex)	String दिए हुए regular expression से match हो रहा है या नहीं हो रहा है इसके हिसाब से boolean value; return होती है ।
boolean regionMatches(boolean ignoreCase, int startIndex, String str, int str_startIndex, int len)	दो string या string के substring को compare किया जाता है ।
boolean regionMatches(int startIndex, String str, int str_StartIndex, int len)	दो string या string के substring को compare किया जाता है ।
boolean startsWith(String prefix)	string prefix को check करके boolean value; return की जाती है ।
boolean startsWith(String prefix, int startIndex)	string prefix को check करके boolean value; return की जाती है ।
byte[] getBytes()	string से byte array को return किया जाता है ।
char charAt(int index)	दिए हुए index पर जो character है उसे return करता है ।
char[] toCharArray()	string को character array में convert किया जाता है ।
CharSequence subSequence(int beginIndex, int endIndex)	character sequence को return किया जाता है ।
int compareTo(Object obj)	String को किसी दुसरे object से compare किया जाता है ।
int compareTo(String str2)	String को किसी दुसरे string से compare किया जाता है ।
int compareToIgnoreCase(String str2)	String को किसी दुसरे string से compare किया जाता है ।
int indexOf(String str)	string के character या substring का index; integer में return किया जाता है ।

int indexOf(String str, int fromIndex)	starting Index से string के character या substring का index; integer में return किया जाता है ।
int indexOf(int ch)	string के character का index; integer में return किया जाता है ।
int indexOf(int ch, int fromIndex)	starting Index से string के character या substring का index; integer में return किया जाता है ।
int lastIndexOf(String str)	string या substring का index; integer में return किया जाता है ।
int lastIndexOf(String str, int fromIndex)	starting Index से string के character या substring का index; integer में return किया जाता है ।
int lastIndexOf(int ch)	string के character का index; integer में return किया जाता है ।
int lastIndexOf(int ch, int fromIndex)	starting Index से string के character का index; integer में return किया जाता है ।
int length()	string की length को integer में return किया जाता है ।
void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin)	string; character array पर copy किया जाता है ।
static String copyValueOf(char[] data)	character array; string को return किया जाता है ।
static String copyValueOf(char[] data, int offset, int count)	character array; string को return किया जाता है ।
static String valueOf(basic data type)	character array; string को return किया जाता है ।
String concat(String str)	अलग-अलग types को string में convert किया जाता है ।
String intern()	string का canonical representation return करता है ।
String replace(char oldChar, char newChar)	string के दिए हुए character से replace किया जाता है ।
String replaceAll(String regex, String replacement)	string को regular expression से replace कर देता है ।
String replaceFirst(String regex, String replacement)	string को regular expression से replace कर देता है ।
String substring(int beginIndex)	string से substring को बनाया जाता है ।
String substring(int beginIndex, int endIndex)	string से substring को बनाया जाता है ।
String toLowerCase()	सभी uppercase letters को lowercase में convert कर देता है ।
String toUpperCase()	सभी lowercase letters को uppercase में convert कर देता है ।

String trim()	शुरुआत के और आखिर के whitespaces; remove किये जाते हैं।
String[] split(String regex)	string के regular expression से अलग-अलग substring बनाकर उस string का array; return किया जाता है।
String[] split(String regex, int limit)	string के regular expression से अलग-अलग substring बनाकर उस string का array; return किया जाता है।

## contentEquals - String Method

contentEquals में String से string buffer को compare किया जाता है और ये method boolean value को return करता है।

Syntax:

```
boolean contentEquals(StringBuffer strbuff)
```

Parameter:

**strbuff** : यहाँ पर string से compare करने के लिए StringBuffer को लिखा जाता है।

Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[]) {
        String str1 = "String1";
        String str2 = "String2";
        StringBuffer str3 = new StringBuffer( "String1");
        System.out.println("str1 and str3 is equal(true) or not(false) :" + str1.contentEquals(str3));
        System.out.println("str2 and str3 is equal(true) or not(false) : " + str2.contentEquals(str3));
    }
}
```

Output:

```
str1 and str3 is equal or not :true
str2 and str3 is equal or not : false
```

## endsWith - String Method

endsWith में दिए हुए String में end के suffix को boolean value में return किया जाता है ।

Syntax:

```
boolean endsWith(String suffix)
```

Parameter:

**suffix** : यहाँ पर string का suffix आता है ।

अगर दिए हुए string से suffix; match होता है, तो true; return होता है और अगर match नहीं होता तो false; return होता है ।

Source Code :

```
//Sample.java
public class Sample{
public static void main(String args[]) {
String str1 = "Hello World";
String str2 = "World";
String str3 = "Hello";
System.out.println("str1 end with World :"+str1.endsWith(str2));
System.out.println("str1 end with Hello : "+str1.endsWith(str3));
}
}
```

Output:

```
str1 end with World : true
str1 end with Hello : false
```

## equals - String Method

equals में दो string को compare किया जाता है और boolean value; return होती है ।

Syntax:

```
boolean equals(Object obj)
```

Parameter:

**obj** : यहाँ पर string से compare करने के लिए object लिया है ।

अगर string और object जब equal होता है , तब true value; return होती है , अगर equal नहीं होता तो false; return होता है ।

Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[]){

        String str1= "Hello World";
        String str2= "Hello World";
        String str3= "Hello Friends";

        System.out.println("str1 is equals to str2 : "+str1.equals(str2));
        System.out.println("str1 is equals to str3 : "+str1.equals(str3));
        System.out.println("str2 is equals to Hello World : "+str1.equals("Hello World"));
        System.out.println("str1 is equals to hello world : "+str1.equals("hello world"));    //equals()
        string method is case-sensitive
    }
}
```

Output:

```
str1 is equals to str2 : true
str1 is equals to str3 : false
str2 is equals to Hello World : true
str1 is equals to hello world : false
```

## equalsIgnoreCase - String Method

equalsIgnoreCase में दो string को compare किया जाता है और boolean value; return होती है ।

Syntax:

```
boolean equalsIgnoreCase(String str)
```

Parameter:

**obj** : यहाँ पर string से compare करने के लिए object लिया है ।

अगर string और object जब equal होता है , तब true value; return होती है , अगर equal नहीं होता तो false; return होता है ।

यहाँ पर ये string method case-sensitive नहीं है । ये uppercase और lowercase में कोई फर्क नहीं रखता ।



Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[]){

        String str1= "Hello World";
        String str2= "Hello World";
        String str3= "Hello Friends";

        System.out.println("str1 is equals to str2 : "+str1.equals(str2));
        System.out.println("str1 is equals to str3 : "+str1.equals(str3));
        System.out.println("str2 is equals to Hello World : "+str1.equals("Hello World"));
        System.out.println("str1 is equals to hello world : "+str1.equals("hello world"));
        //equalsIgnoreCase() string method is incase-sensitive
    }
}
```

Output:

```
str1 is equals to str2 : true
str1 is equals to str3 : false
str2 is equals to Hello World : true
str1 is equals to hello world : true
```

## matches - String Method

matches() ये string method; String दिए हुए regular expression से match हो रहा है या नहीं हो रहा है इसके हिसाब से boolean value; return होती है ।

Syntax:

```
boolean matches(String regex)
```

Parameter:

**regex** : यहाँ पर ये regular expression है ।

अगर regular expression से string match हो रहा है तो true; return होता है , अगर match नहीं होता है तो false; return होता है ।

Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[]){
        String str = new String("Hello World");

        System.out.println(str.matches("(.)Hello(.)"));
        System.out.println(str.matches("Hello(.)"));
        System.out.println(str.matches("(.)World"));
        System.out.println(str.matches("(.)World(.)"));
        System.out.println(str.matches("World(.)"));
    }
}
```

Output:

```
true
true
true
false
false
```

## regionMatches (ignorecase)- String Method

regionMatches() इस String Method से दो string या string के substring को compare किया जाता है ।

Syntax:

```
boolean regionMatches(boolean ignoreCase, int startIndex, String other, int
other_startIndex, int len)
```

Parameter:

**ignoreCase** : जब true होता है, तब regionMatches method in-case-sensitive होता है, अगर false या ignorecase नहीं होता तो regionMatches method case-sensitive हो जाता है ।

**startIndex** : यहाँ पर string object की starting index की position होती है ।

**other** : यहाँ पर ये region match करने के लिए दूसरा string object है ।

**other\_startIndex** : ये दूसरे string object की starting index है ।

**len** : यहाँ पर ये substring की length है ।

## Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[])
    {
        String str1 = "Hello World";
        String str2 = "My World";
        String str3 = "My world";

        System.out.println("str1 and str2 have 'World' word : " + str1.regionMatches(6, str2, 3, 5));
        System.out.println("for Case-sensitive, str1 and str3 have 'World' word : " +
            str1.regionMatches(false, 6, str3, 3, 5));
        System.out.println("for Case-sensitive, str1 and str3 have 'World' word : " +
            str1.regionMatches(6, str3, 3, 5));
        System.out.println("str1 and str3 have 'World' word : " + str1.regionMatches(true, 6, str3, 3,
            5));
        // regionMatches is in-case-sensitive if ignorecase is true
        // regionMatches is case-sensitive if ignorecase is false or none
        System.out.println("str1 and str3 have 'World' word " + str1.regionMatches(true, 6, str3, 3,
            5));
    }
}
```

## Output:

```
str1 and str2 have 'World' word : true
for Case-sensitive, str1 and str3 have 'World' word : false
for Case-sensitive, str1 and str3 have 'World' word : false
str1 and str3 have 'World' word : true
str1 and str3 have 'World' word true
```

अगर str1 के Hello World के 'World' इस अक्षर से str2 के 'World' से match करना हो तो,

```
str1.regionMatches(6, str2, 3, 5))
```

str1 H e l l o W o r l d

str1.regionMatches(6, str2, 3, 5)

str2 M y W o r l d  
Substring Length

return true

## regionMatches (without ignorecase)- String Method

regionMatches() इस String Method से दो string या string के substring को compare किया जाता है ।  
ये regionMatches() case-sensitive है ।

Syntax:

```
boolean regionMatches(int startIndex, String other, int other_startIndex, int len)
```

Parameter:

**startIndex** : यहाँ पर string object की starting index की position होती है ।

**other** : यहाँ पर ये region match करने के लिए दूसरा string object है ।

**other\_startIndex** : ये दूसरे string object की starting index है ।

**len** : यहाँ पर ये substring की length है ।

Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[])
    {
        String str1 = "Hello World";
        String str2 = "My World";
        String str3 = "My world";

        System.out.println("str1 and str2 have 'World' word : " + str1.regionMatches(6, str2, 3, 5));
        System.out.println("for Case-sensitive, str1 and str3 have 'World' word : " +
            str1.regionMatches(false, 6, str3, 3, 5));
        System.out.println("for Case-sensitive, str1 and str3 have 'World' word : " +
            str1.regionMatches(6, str3, 3, 5));
        System.out.println("str1 and str3 have 'World' word : " + str1.regionMatches(true, 6, str3, 3,
            5));
        // regionMatches is in-case-sensitive if ignorecase is true
        // regionMatches is case-sensitive if ignorecase is false or none
        System.out.println("str1 and str3 have 'World' word " + str1.regionMatches(true, 6, str3, 3,
            5));
    }
}
```

Output:

```
str1 and str2 have 'World' word : true  
for Case-sensitive, str1 and str3 have 'World' word : false  
for Case-sensitive, str1 and str3 have 'World' word : false  
str1 and str3 have 'World' word : true  
str1 and str3 have 'World' word true
```

अगर str1 के Hello World के 'World' इस अक्षर से str2 के 'World' से match करना हो तो,

```
str1.regionMatches(6, str2, 3, 5))
```

## startsWith (without startIndex)- String Method

startsWith() इस String Method से string prefix को check करके boolean value; return की जाती है ।

Syntax:

```
boolean startsWith(String prefix)
```

Parameter:

**prefix** : यहाँ पर ये prefix है | जो string से match करेगा |

अगर prefix; String के starts से match करता है, तो true value; return करता है , अगर match नहीं होता तो false; return होता है ।

Source Code :

```
//Sample.java  
public class Sample{  
    public static void main(String args[]) {  
String str = new String("Hello World");  
        System.out.println("String is starts with Hello : " + str.startsWith("Hello") );  
        System.out.println("String is starts with World : " + str.startsWith("World") );  
    }  
}
```

Output:

```
String is starts with Hello : true  
String is starts with World : false
```

## startsWith (with startIndex)- String Method

startsWith() इस String Method से string prefix को check करके boolean value; return की जाती है ।

Syntax:

```
boolean startsWith(String prefix, int startIndex)
```

Parameter:

**prefix** : यहाँ पर ये prefix है । जो string से match करेगा ।

**startIndex** : ये string के शुरुआत से दिए हुए index पर देखता है ।

अगर prefix; String के starts से और दिए हुए index से match करता है, तो true value; return करता है , अगर match नहीं होता तो false; return होता है ।

Source Code :

```
//Sample.java
public class Sample{
    public static void main(String args[]) {
        String str = new String("Hello World");
        System.out.println(str.startsWith("World", 6));
        System.out.println(str.startsWith("Hello", 0));
    }
}
```

Output:

```
String is starts with Hello : true
String is starts with World : false
```

## getBytes- String Method

getBytes इस String Method से string से byte array को return किया जाता है ।

getBytes के लिए दो variants है ।

Syntax:

```
byte[] getBytes()
byte[] getBytes(String charsetName)
```

Parameter:

**charsetName** : ये दूसरे variant में charset का नाम होता है । for eg. UTF-8, UTF-16

यहाँ पर byets के sequence पर string को encode किया जाता है ।

**UnsupportedEncodingException** : जब दिए हुए charset names; support नहीं करते तब getBytes में Exception Handling के लिए इस्तेमाल किया जाता है ।

### Example for 1st Variant

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str1 = new String("Hello World");
        String str2 = new String( str1.getBytes());
        System.out.println( str2 );
    }
}
```

Output:

```
Hello World
```

### Example for 2nd Variant

```
//Sample.java
import java.io.*;    //using for class UnsupportedEncodingException
public class Sample {
    public static void main(String args[]) {
        String str1 = new String("Hello World");
        try{
            String str2 = new String( str1.getBytes("US-ASCII"));
            System.out.println("US-ASCII : " + str2 );
            String str3 = new String( str1.getBytes("UTF-8"));
            System.out.println("UTF-8 : " + str3 );
        }
        catch(UnsupportedEncodingException e){
            System.out.println("Unsupported character-set");
        }
    }
}
```

Output:

```
US-ASCII : Hello World
UTF-8 : Hello World
```

## charAt- String Method

charAt इस String Method से दिए हुए index पर जो character है उसे return करता है ।

Syntax:

```
char charAt(int index)
```

Parameter:

**index** : जो character ढूँढना है उसका index यहाँ पर दिया जाता है ।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str = new String("Hello World");
        System.out.println("0th index character in string : " + str.charAt(0) );
        System.out.println("6th index character in string : " + str.charAt(6) );
    }
}
```

Output:

```
0th index character in string : H
6th index character in string : W
```

	0	1	2	3	4	5	6	7	8	9	10
str	H	e	l	l	o		W	o	r	l	d
	↑						↑				
	charAt(0)						charAt(6)				

## toCharArray- String Method

charAt इस String Method से string को character array में convert किया जाता है ।

Syntax:

```
char[] toCharArray()
```



## Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str = new String("Hello World!");
        System.out.println(str.toCharArray() );
    }
}
```

## Output:

```
Hello World!
```

## subSequence - String Method

subSequence इस String Method से character sequence को return किया जाता है ।

## Syntax:

```
CharSequence subSequence(int beginIndex, int endIndex)
```

## Parameter:

**beginIndex** : String से जहाँ से string का character sequence चाहिए उसका index यहाँ पर आता है ।

**endIndex** : String से यहाँ तक string का index दिया जाता है । लेकिन जो index यहाँ पर दिया जाता है उससे एक character कम print होता है ।

## Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str = new String("Hello World");
        System.out.println(str.subSequence(0, 9) );
        System.out.println(str.subSequence(5, 11) );
    }
}
```

## Output:

```
Hello Wor
World
```

## compareTo (for Object) - String Method

compareTo इस String Method से String को किसी दुसरे object से compare किया जाता है ।

Syntax:

```
int compareTo(Object obj)
```

Parameter:

**obj** : ये किसी दुसरे String का object है ।

compareTo इस string method में जब '0' value return होती तब String और दुसरे string का object; lexicographically; equal होता है । अगर String और दुसरे string का object equal नहीं होता तो non-zero, positive और negative; value return होती है ।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str1 = "Hello World";
        String str2 = new String("Hello World");
        String str3 = new String("Hello Friends");
        System.out.println(str1.compareTo(str2) );
        System.out.println(str1.compareTo(str3) );
    }
}
```

Output:

```
0
17
```

## compareTo (for String) - String Method

compareTo इस String Method से String को किसी दुसरे string से compare किया जाता है ।

Syntax:

```
int compareTo(String str2)
```

Parameter:

**str2** : ये किसी दुसरा string है ।

compareTo इस string method में जब '0' value return होती तब String और दुसरा string ; lexicographically; equal होता है । अगर String और दुसरा string equal नहीं होता तो non-zero, positive और negative; value return होती है ।

## Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str1 = "Hello World";
        String str2 = "Hello World";
        String str3 = "Hello Friends";
        System.out.println(str1.compareTo(str2) );
        System.out.println(str1.compareTo(str3) );
    }
}
```

Output:

```
0
17
```

## compareToIgnoreCase - String Method

compareToIgnoreCase इस String Method से String को किसी दुसरे string से compare किया जाता है ।

Syntax:

```
int compareToIgnoreCase(String str2)
```

Parameter:

**str2** : ये किसी दुसरा string है ।

compareToIgnoreCase ये string method; compareTo(String str2) जैसा होता है । सिर्फ ignorecase में uppercase और lowercase में कोई फर्क नहीं रहता । ये दोनों case को ignore करता है ।

## Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str1 = "Hello World";
        String str2 = "hello world";
        String str3 = "hello friends";
        System.out.println(str1.compareToIgnoreCase(str2) );
        System.out.println(str1.compareToIgnoreCase(str3) );
    }
}
```

Output:

```
0
17
```

## indexOf - for string (Without fromIndex) - String Method

indexOf इस String Method से string के character या substring का index; integer में return किया जाता है।

Syntax:

```
int indexOf(String str)
```

Parameter:

**str2** : ये किसी दूसरा substring है।

अगर substring या किसी string का character ढूँढा नहीं जाता तो '-1' integer value return होती है।

ये string का first occurrence को ढूँढकर उसका index return करता है।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World");
        String str2 = new String("World");


        System.out.println("Index found");
        System.out.println(str1.indexOf(str2));
    }
}
```

Output:

```
Index found
6
```

index	0	1	2	3	4	5	6	7	8	9	10
str	H	e	l	l	o		W	o	r	l	d

indexOf("World")    Output : 6



## indexOf - for string (With fromIndex) - String Method

indexOf इस String Method से starting Index से string के character या substring का index; integer में return किया जाता है।

Syntax:

```
int indexOf(String str, int fromIndex)
```

Parameter:

**str2** : ये किसी दुसरा substring है।

**fromIndex** : ये string के कौनसे index से substring या character को ढूँढना है, वो index यहाँ पर दिया जाता है।

अगर substring या किसी string का character ढूँढा नहीं जाता तो '-1' integer value return होती है।

ये string का first occurrence को ढूँढकर उसका index return करता है।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World");
        String str2 = new String("World");

        System.out.println("Index found");
        System.out.println(str1.indexOf(str2, 6));
    }
}
```

Output:

```
Index found
6
```

index	0	1	2	3	4	5	6	7	8	9	10
str	H	e	l	l	o		W	o	r	l	d

←————— 6 —————→

indexOf("World", 6)      Output : 6

## indexOf - for character (Without fromIndex) - String Method

indexOf इस String Method से string के character का index; integer में return किया जाता है ।

Syntax:

```
int indexOf(int ch)
```

Parameter:

**str2** : ये किसी दुसरा substring है ।

अगर किसी string का character ढूँढा नहीं जाता तो '-1' integer value return होती है ।

ये character का first occurrence को ढूँढकर उसका index return करता है ।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World");
        char ch = 'o';
        System.out.println("Index found");
        System.out.println(str1.indexOf( ch ));
    }
}
```

Output:

```
Index found
4
```

index	0	1	2	3	4	5	6	7	8	9	10
str	H	e	l	l	o		W	o	r	l	d

indexOf( ch ) first occurrence of o in string

ch is 'o' return 4

## indexOf - for character (With fromIndex) - String Method

indexOf इस String Method से starting Index से string के character या substring का index; integer में return किया जाता है ।

Syntax:

```
int indexOf(int ch, int fromIndex)
```

Parameter:

**str2** : ये किसी दुसरा substring है ।

**fromIndex** : ये string के कौनसे index से character को ढूँढना है, वो index यहाँ पर दिया जाता है ।

अगर किसी string का character ढूँढा नहीं जाता तो '-1' integer value return होती है ।

ये string का first occurrence को ढूँढकर उसका index return करता है ।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World");
        char ch = 'o';

        System.out.println("Index found");
        System.out.println(str1.indexOf(ch, 6));
    }
}
```

Output:

```
Index found
7
```

index	0	1	2	3	4	5	6	7	8	9	10
str	H	e	l	l	o	W	o	r	l	d	

indexOf( ch, 6 )      first occurrence of o in string  
   after 6th index  
   return 7

ch is 'o'

## lastIndexOf - for string (Without fromIndex) - String Method

lastIndexOf इस String Method से string या substring का index; integer में return किया जाता है ।

Syntax:

```
int lastIndexOf(String str)
```

Parameter:

**str2** : ये किसी दूसरा substring है ।

अगर substring या किसी string को ढूँढा नहीं जाता तो '-1' integer value return होती है |ये string का last occurrence को ढूँढकर उसका index return करता है ।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World World");
        String str2 = new String("World");

        System.out.println("Index found");
        System.out.println(str1.lastIndexOf( str2 ));
    }
}
```

Output:

```
Index found
12
```

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
str	H	e	l	l	o		W	o	r	l	d		W	o	r	l	d

lastIndexOf( str2 )

str2 is World

return 12

last occurrence of World starts on 12nd index



## lastIndexOf - for string (With fromIndex) - String Method

indexOf इस String Method से starting Index से string के character या substring का index; integer में return किया जाता है ।

Syntax:

```
int lastIndexOf(String str, int fromIndex)
```

Parameter:

**str2** : ये किसी दुसरा substring है ।

**fromIndex** : यहाँ पर जो index दी जाती उस index के पीछे से substring ढूँढा जाता है ।

अगर substring ढूँढा नहीं जाता तो '-1' integer value return होती है ।

अगर एक string में एक ही प्रकार के दो substring होते हैं तब उसका last occurrence को ढूँढकर उसका index return करता है ।

जब fromindex दिया जाता है, तब दिए हुए index से पीछे से substring का last occurrence ढूँढा जाता है ।

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World World");
        String str2 = new String("World");

        System.out.println("Index found");
        System.out.println(str1.lastIndexOf( str2, 3 ));
        System.out.println(str1.lastIndexOf( str2, 9 ));
    }
}
```

Output:

```
Index found
-1
6
```

index 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
str H e l l o W o r l d W o r l d

lastIndexOf( str2, 3 )      last occurrence of World  
str2 is World      lastIndexOf( str2, 9 )

Output : return -1(World is not found) | return 6(World is found on 6th index)

lastIndexOf इस String Method से string के character का index; integer में return किया जाता है।

```
int lastIndexOf(int ch)
```

**ch** : ये किसी दुसरा substring है ।

**fromIndex** : यहाँ पर जो index दी जाती उस index के पीछे से substring ढूँढा जाता है।

अगर किसी string का character ढूंढा नहीं जाता तो '-1' integer value return होती है।

ये character का last occurrence को ढूँढकर उसका index return करता है।

```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World");
        char ch = 'o';

        System.out.println("Index found");
        System.out.println(str1.lastIndexOf( ch ));
    }
}
```

Index found  
7

82

## lastIndexOf - for character (With fromIndex) - String Method

lastIndexOf इस String Method से starting Index से string के character का index; integer में return किया जाता है ।

Syntax:

```
int lastIndexOf(int ch, int fromIndex)
```

Parameter:

**ch** : ये एक character है ।

**fromIndex** : ये string के कौनसे index से character को ढूँढना है, वो index यहाँ पर दिया जाता है ।

अगर character ढूँढा नहीं जाता तो '-1' integer value return होती है ।

अगर एक string में एक ही प्रकार के दो character होते हैं तब उसका last occurrence को ढूँढकर उसका index return करता है ।

जब fromindex दिया जाता है, तब दिए हुए index से पीछे से character का last occurrence ढूँढा जाता है ।

Source Code

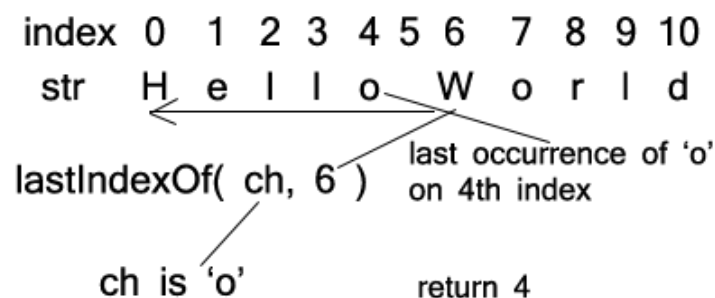
```
//Sample.java
public class Sample {
    public static void main(String args[]) {

        String str1 = new String("Hello World");
        char ch = 'o';

        System.out.println("Index found");
        System.out.println(str1.lastIndexOf(ch, 6));
    }
}
```

Output:

```
Index found
4
```



## length - String Method

length इस String Method से string की length को integer में return किया जाता है ।

Syntax:

```
int length()
```

Source Code

```
//Sample.java
public class Sample {
    public static void main(String args[]) {
        String str = new String("Hello World");
        System.out.print("Length of str : ");
        System.out.println(str.length());
    }
}
```

Output:

```
Length of str : 11
```

## getChars - String Method

getChars इस String Method से string; character array पर copy किया जाता है ।

Syntax:

```
void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin)
```

Parameter:

**srcBegin** : ये वो index होता है जो copy करने के लिए पहला character होता है ।

**srcEnd** : ये वो index होता है जो copy करने के लिए आखिरी character होता है ।

**dest** : किस character array पर string; copy करके रखने है वो character array ।

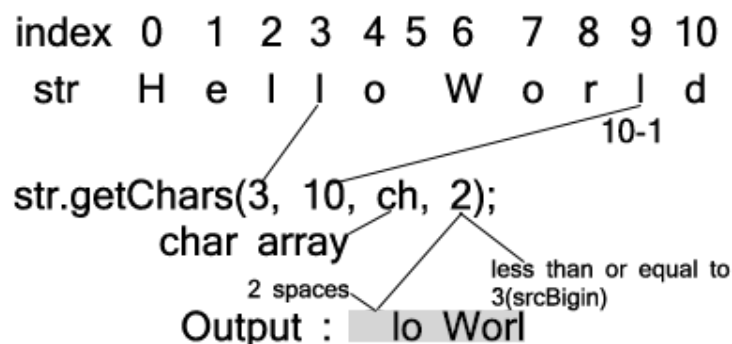
**destBegin** : जितना srcBegin है उतनी ही value और उससे कम value इसकी होती है ।

Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str = new String("Hello World");
char[] ch = new char[10];
    str.getChars(3, 10, ch, 2);
    System.out.println(ch);
}
}
```

Output:

```
lo Worl
```



## copyValueOf (Without offset, count) - String Method

copyValueOf इस String Method से character array; string को return किया जाता है ।

Syntax:

```
static String copyValueOf(char[] data)
```

Parameter:

**data** : ये character array है ।

Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
char[] str1 = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'};
String str2 = "";
System.out.println("String : " + str2.copyValueOf(str1));
}
}
```

Output:

```
String : Hello World
```

## copyValueOf (With offset, count) - String Method

copyValueOf इस String Method से character array; string को return किया जाता है ।

Syntax:

```
static String copyValueOf(char[] data, int offset, int count)
```

Parameter:

**data** : ये character array है ।

**offset** : ये subarray की शुरुआत है ।

**count** : इतने characters offset से count किये जाते है ।

Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
char[] str1 = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'};
String str2 = "";
System.out.println("String : " + str2.copyValueOf(str1, 3, 7));
}
}
```

Output:

```
String : lo Worl
```

index	0	1	2	3	4	5	6	7	8	9	10
str1	H	e	l	l	o		W	o	r	l	d
Count characters		1	2	3	4	5	6	7			
copyValueOf( str2, 3, 7)											
char array stored in str2      Output : lo Worl											

## valueOf - String Method

valueOf इस String Method से अलग-अलग types को string में convert किया जाता है ।

Syntax:

```
static String valueOf(basic data type)
```

ये हर एक data type को string में return करता है ।

Source Code

```
//Sample.java
class Sample{
    public static void main(String args[]) {
        byte b = 5;
        int i = 10;
        float f = 0.8f;
        double d = 5.45455d;
        char[] str = {'H', 'e', 'l', 'l', 'o'};
        boolean bool = true;
        System.out.println("Byte Value : " + String.valueOf(b) );
        System.out.println("Integar Value : " + String.valueOf(i) );
        System.out.println("Float Value : " + String.valueOf(f) );
        System.out.println("Double Value : " + String.valueOf(d) );
        System.out.println("Character Value : " + String.valueOf(str) );
        System.out.println("boolean Value : " + String.valueOf(bool) );
    }
}
```

Output:

```
Byte Value : 5
Integar Value : 10
Float Value : 0.8
Double Value : 5.45455
Character Value : Hello
boolean Value : true
```



## concat - String Method

मुख्य string के end पर दूसरे string को जोड़ा है ।

Syntax:

```
String concat(String str)
```

Parameter:

**str** : इस string को मुख्य string के end पर जोड़ा जाता है |:

Source Code

```
//Sample.java
class Sample{
    public static void main(String args[]) {

        String str = "Hello ";

        System.out.println(str.concat("World"));
    }
}
```

Output:

```
Hello World
```

## intern - String Method

intern ये String method; string का canonical representation return करता है ।

Syntax:

```
String intern()
```

अगर intern() इस method के साथ string object का इस्तेमाल किया जाता है तो string pool से string object को string में return किया जाता है । ये case sensitive होता है ।

## Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str1=new String("Hello World");
String str2="Hello World";
String str3="hello world";
System.out.println(str1.intern()==str2); //both are same with intern
System.out.println(str1==str2); //both are different without intern
System.out.println(str1.intern()==str3); //intern is case sensitive
}
}
```

## Output:

```
true
false
false
```

## replace - String Method

replace ये String method string के दिए हुए character से replace किया जाता है ।

## Syntax:

```
String replace(char oldChar, char newChar)
```

## Parameter:

**oldChar** : ये string में से लिया हुआ character होता है ।

**newChar** : जिस character से replace करना होता है वो character यहाँ पर आता है ।

अगर कोई character उसे found नहीं होता तो original string; return हो जाती है ।

## Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str=new String("Hello World");
System.out.println("Replaced o to j from String : " + str.replace('o', 'j'));
System.out.println("Replaced l to o from String : " + str.replace('l', 'o'));
}
}
```

Output:

```
Replaced o to j from String : Hellj Wjrlld  
Replaced l to o from String : Heooo Worod
```

## replaceAll - String Method

replaceAll ये String method; string को regular expression से replace कर देता है ।

Syntax:

```
String replaceAll(String regex, String replacement)
```

Parameter:

**regex** : ये string में से लिया हुआ character होता है ।

**replacement** : जिस regular expression से replace करना हो वो regular expression ।

PatternSyntaxException, अगर regular expression और syntax गलत होता है ।

Source Code

```
//Sample.java  
class Sample{  
    public static void main(String args[]){  
        String str1=new String("Hello World");  
        String str2=new String("Hi# Hello# How are you ?");  
        System.out.println(str1.replaceAll("World", "Friends"));  
        System.out.println(str2.replaceAll("#", "!"));  
    }  
}
```

Output:

```
Hello Friends  
Hi! Hello! How are you ?
```

## replaceFirst - String Method

replaceFirst ये String method; string को regular expression से replace कर देता है ।

Syntax:

```
String replaceFirst(String regex, String replacement)
```

Parameter:

**regex** : ये string में से लिया हुआ character होता है ।

**replacement** : जिस regular expression से replace करना हो वो regular expression ।

यहाँ पर string के पहले ही regular expression को replace किया जाता है ।

PatternSyntaxException, अगर regular expression और syntax गलत होता है ।

Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str1=new String("Hello World World");
String str2=new String("Hi# Hello# How are you ?");
System.out.println(str1.replaceFirst("World", "Friends"));
System.out.println(str2.replaceFirst("#", "!"));
}
}
```

Output:

```
Hello Friends World
Hi! Hello# How are you ?
```

## substring (Without endIndex) - String Method

substring ये String method; string से substring को बनाया जाता है ।

Syntax:

```
String substring(int beginIndex)
```

Parameter:

**beginIndex** : ये शुरुआत की index जिससे substring को बनाया जाता है ।

IndexOutOfBoundsException, अगर index; string की length से ज्यादा हो और negative हो ।

## Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str1=new String("This is my Website");
System.out.println(str1.substring(0));
System.out.println(str1.substring(5));
}
}
```

## Output:

```
This is my Website
is my Website
```

## substring (With endIndex) - String Method

substring ये String method; string से substring को बनाया जाता है ।

## Syntax:

```
String substring(int beginIndex, int endIndex)
```

## Parameter:

**beginIndex** : ये शुरुआत की index जिससे substring को बनाया जाता है ।

**endIndex** : जिस index तक string को बनाया जाए वो index यहाँ आता है ।

IndexOutOfBoundsException, अगर index; string की length से ज्यादा हो और negative हो ।

## Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str1=new String("This is my Website");
System.out.println(str1.substring(0, 6));
System.out.println(str1.substring(5, 13));
}
}
```

## Output:

```
This i
is my We
```

## toLowerCase - String Method

toLowerCase ये String method; सभी uppercase letters को lowercase में convert कर देता है।

Syntax:

```
String toLowerCase()
```

Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str = new String("This is my Website");
System.out.println(str.toLowerCase());
}
}
```

Output:

```
this is my website
```

## toUpperCase - String Method

toUpperCase ये String method; सभी lowercase letters को uppercase में convert कर देता है।

Syntax:

```
String toUpperCase()
```

Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str = new String("This is my Website");
System.out.println(str.toUpperCase());
}
}
```

Output:

```
THIS IS MY WEBSITE
```

## trim - String Method

trim इस String method; से शुरुआत के और आखिर के whitespaces को remove किये जाते है ।

Syntax:

```
String trim()
```

Source Code

```
//Sample.java
class Sample{

    public static void main(String args[]){

        String str = new String("  Hello World  ");
        System.out.println("String Without trim : " + str);
        System.out.println("String with trim : " + str.trim());
    }
}
```

Output:

```
String Without trim :   Hello World
String with trim : Hello World
```

## split (Without limit) - String Method

split इस String method; से string के regular expression से अलग-अलग substring बनाकर उस string का array; return किया जाता है ।

Syntax:

```
String[] split(String regex)
```

Parameter:

**regex** : यहाँ पर regular expression आता है ।

PatternSyntaxException, अगर regular expression गलत हो ।

## Source Code

```
//Sample.java
class Sample{

    public static void main(String args[]){

        String str = new String("Hello/my/Friends");
        System.out.println("String : " + str);
        String[] arr = str.split("/");

        for( String strarr : arr ){
            System.out.println(strarr);
        }
    }
}
```

## Output:

```
String : Hello/my/Friends
Hello
my
Friends
```

## split (With limit) - String Method

split इस String method; से string के regular expression से अलग-अलग substring बनाकर उस string का array; return किया जाता है ।

### Syntax:

```
String[] split(String regex, int limit)
```

### Parameter:

**regex** : यहाँ पर regular expression आता है ।

**limit** : शुरुआत से कितनी बार regular expression applied करना है उसका limit यहाँ पर दिया जाता है ।

अगर regular expression से ज्यादा limit(0 और negative value) हो तो string के जितने substring बनते हैं उसे बनाया जाता है ।

PatternSyntaxException, अगर regular expression गलत हो ।



## Source Code

```
//Sample.java
class Sample{
public static void main(String args[]){
String str = new String("Hello/my/Friends");
System.out.println("String : " + str);
String[] arr1 = str.split("/");

for( String strarr : arr1 ){
System.out.println(strarr);
}
System.out.println("Split with 1 limit");
System.out.println("String : " + str);
String[] arr2 = str.split("/", 1);
for( String strarr : arr2 ){
System.out.println(strarr);
}
System.out.println("Split with 2 limit");
System.out.println("String : " + str);
String[] arr3 = str.split("/", 2);
for( String strarr : arr3 ){
System.out.println(strarr);
}
System.out.println("Split with 0 limit");
System.out.println("String : " + str);
String[] arr4 = str.split("/", 0);
for( String strarr : arr4 ){
System.out.println(strarr);
}
System.out.println("Split with -2 limit");
System.out.println("String : " + str);
String[] arr5 = str.split("/", -2);
for( String strarr : arr5 ){
System.out.println(strarr);
}
}
}
```

Output:

String : Hello/my/Friends

Hello

my

Friends

Split with 1 limit

String : Hello/my/Friends

Hello/my/Friends

Split with 2 limit

String : Hello/my/Friends

Hello

my/Friends

Split with 0 limit

String : Hello/my/Friends

Hello

my

Friends

Split with -2 limit

String : Hello/my/Friends

Hello

my

Friends

## Methods Introduction

हर Java के Program में Main Method का इस्तेमाल किया जाता है ।

C, C++ में जो function इस्तेमाल किये जाते थे, वैसे ही Java में उसे Methods कहा जाता है ।

Java के Methods में अनेक से statements grouped करके रखे जाते है ।

Java में अपने खुदके Methods भी बनाये जाते है । Methods parameters, बिना parameters, return type और बिना return type से create किये जाते है ।

### main Method

```
class Sample{  
    public static void main(String args[]){ // main method  
        //statements;  
    }  
}
```

### Syntax for Java Method

```
visibility return_type method_name(arguments_list){  
    //method's statments;  
}
```

### Example for Java Method

दिए हुआ example बिना parameter के है ।

Java Methods चार हिस्सों में होता है ।

**Visibility** : Method की accessibility कहा तक होनी चाहिए, ये visibility modes बताते है । इसे access specifier या access modifier भी कहते है । Java में visibility के तीन प्रकार होते है ।

**private** : private methods सिर्फ अपने class के लिए ही सिमित होते है । class के बाहर या किसी दुसरे class में इन्हें access नहीं किया जाता ।

**public** : public methods को किसी भी class पर access किया जाता है ।

**protected** : protected methods; inheritance या अपने sub-classes के साथ कहा पर भी access किये जाते है ।

**return\_type** : ये method का return type है । जब method को कोई return type नहीं दिया जाता, तो 'void' इसका default type होता है ।

**method\_name** : Method का नाम statements से related कुछ भी हो सकता है । Method का नाम कोई keyword नहीं होता ।

**arguments\_list** : Methods के parameters को () parenthesis के अन्दर लिखे जाते है । ये program की requirement के हिसाब से दिए जाते है ।

Example में देखे,

**visibility** : public

**return\_type** : int

**method\_name** : add

**arguments\_list** : No Argument

```
public int add(){  
    int a, b, c;  
    a = 50; b = 60;  
    c = a + b;  
    return c;  
}
```

### Method With Arguments

```
public int add(int x, int y){ //x and y is Method Arguments  
    return x + y;  
}
```

### Method Calling

जब Method को define किया जाता है तभी उसको call किया जा सकता है। कुछ methods pre-defined होते हैं। जिनकी definition; packages में लिखी जाती है।

### Example for Pre-defined Method

यहाँ पर PrintStream class की println ये method इस्तेमाल की गयी है। ये method java.io इस package के अन्दर defined की गयी है। यहाँ पर println method; call की गयी है।

```
class Sample{  
    public static void main(String args[]){  
        Sample s = new Sample();  
        System.out.println("Hello"); // println is predefined method  
    }  
}
```

Output:

```
Hello
```

Method को return value के साथ या बिना return value के साथ call किया जाता है ।  
Method को call किसी return type के variable में भी store किया जाता है ।

```
int a = add();
```

### Full Example for Method Calling

```
class Sample{  
    public int add(int x, int y){    // Non-Static Method  
        return x + y;  
    }  
    public static void main(String args[]){  
        Sample s = new Sample();  
        int a = s.add(5, 9);  
        System.out.println(a);  
    }  
}
```

Output:

```
14
```

### Using Static Method

जब static का इस्तेमाल method में किया जाता है तो class के object के बिना method को access किया जाता है ।

```
class Sample{  
    public static int add(int x, int y){ // static method  
        return x + y;  
    }  
    public static void main(String args[]){  
        int a = add(5, 9);  
        System.out.println(a); //or System.out.println(add(5, 9));  
    }  
}
```

Output:

```
14
```

Java में एक से ज्यादा एक जैसे नाम के method को इस्तेमाल किया जाता है ।

एक java program में एक से ज्यादा एक ही नाम के method इस्तेमाल करना 'Method Overloading' कहते हैं ।

एक method दूसरी बार पहले जैसा इस्तेमाल नहीं किया जा सकता ।

### For Example,

जब ऐसा होता है तो, method is already defined in class ये error आ जाता है ।

```
public void disp(){  
public void disp(){
```

Method Overloading का इस्तेमाल करना हो तो, हर same name के method का parameter का data\_type, parameters की संख्या अलग-अलग इस्तेमाल किया जाता है ।

### For Example,

```
void disp(){ }  
void disp(int a){ }  
void disp(int a){ }  
void disp(int a, int b){ }  
void disp(int a, int b, int c){ }
```

### Example for Changing Data Type : Method Overloading

```
class Sample{  
public void disp(double x){  
System.out.println(x);  
}  
public int disp(int x){  
return x;  
}  
public static void main(String[] args){  
Sample s = new Sample();  
System.out.println("Value of x : " + s.disp(5));  
System.out.println("Value of y : " + s.disp(6));  
}  
}
```

Output:

```
Value of x : 5  
Value of y : 6
```

## Example for Changing number of Arguments : Method Overloading

```
class Sample{  
    public void disp(int x, int y){  
        System.out.println("Value of x : " + x);  
        System.out.println("Value of y : " + y);  
    }  
    public void disp(int x){  
        System.out.println("Value of x : " + x);  
    }  
    public static void main(String[] args){  
        Sample s = new Sample();  
        s.disp(5, 6);  
        s.disp(7);  
    }  
}
```

Output:

```
Value of x : 5  
Value of y : 6  
Value of x : 7
```

# Date and Time Methods

Java में Date और Time; java.util.Date इस class में होते हैं | date और time के लिए constructor और कुछ methods होते हैं |

Source Code:

```
import java.util.Date;
class Sample{
public static void main(String args[]){
Date date = new Date();
System.out.println(date);
}
}
```

Output:

```
Mon Jan 30 11:51:47 IST 2017
```

**OR**

Source Code:

```
class Sample{
public static void main(String args[]){
java.util.Date date = new java.util.Date();
System.out.println(date);
}
}
```

Output:

```
Mon Jan 30 11:51:47 IST 2017
```

Current date का timestamp को date object और getTime() इस Method के साथ access किया जाता है |



Source Code:

```
import java.util.Date;
class Sample{
    public static void main(String args[]) {
        Date date = new Date();
        System.out.println("Current Timestamp : " + date.getTime());
    }
}
```

Output:

```
Current Timestamp : 1485758540247
```

## Character Methods

Java में character(char) इस primitive data type के लिए कुछ Methods बनाये गए हैं।

Java में ये सभी methods; java.lang.Character इस class के अंतर्गत आते हैं।

इन्हीं Methods से character के बारे में जाना जाता है जैसे, लिया हुआ character; uppercase है या नहीं, space है या नहीं। ये Methods; Unicode character को support करता है।

Character Methods	Description
isDigit()	character; Digit है या नहीं इसे boolean; value में return करता है।
isLetter()	character; Letter है या नहीं इसे boolean; value में return करता है।
isLowerCase()	character; lowercase में है या नहीं इसे boolean; value में return करता है।
isUpperCase()	character; uppercase में है या नहीं इसे boolean; value में return करता है।
isWhitespace()	character; whitespace है या नहीं इसे boolean; value में return करता है।
toLowerCase()	Uppercase के character को lowercase में convert कर देता है।
toUpperCase()	Lowercase के character को uppercase में convert कर देता है।

## isDigit - Character Method

isDigit() इस Character Method से दिया हुआ character; Digit है या नहीं इसे boolean; value में return करता है।

Syntax:

```
boolean isDigit(char c)
```

Source Code:

```
class Sample{
public static void main(String args[]){
    System.out.println(Character.isDigit(3)); // is only Digit like Integer
    System.out.println(Character.isDigit('3')); // is Character Digit
    System.out.println(Character.isDigit('r')); // is Letter
}
}
```

Output:

```
false
true
false
```

## isLetter - Character Method

isLetter() इस Character Method से दिया हुआ character; Letter है या नहीं इसे boolean; value में return करता है।

Syntax:

```
boolean isLetter(char c)
```

Source Code:

```
class Sample{
public static void main(String args[]){
    System.out.println(Character.isLetter(3));
    System.out.println(Character.isLetter('3'));
    System.out.println(Character.isLetter('r'));
}
}
```

Output:

```
false
false
true
```

## isLowerCase - Character Method

isLowerCase() इस Character Method से दिया हुआ character; lowercase में है या नहीं इसे boolean; value में return करता है ।

Syntax:

```
boolean isLowerCase(char c)
```

Source Code:

```
class Sample{  
    public static void main(String args[]){  
        System.out.println(Character.isLowerCase(3));  
        System.out.println(Character.isLowerCase('R'));  
        System.out.println(Character.isLowerCase('r'));  
    }  
}
```

Output:

```
false  
false  
true
```

## isUpperCase - Character Method

isUpperCase() इस Character Method से दिया हुआ character; uppercase में है या नहीं इसे boolean; value में return करता है ।

Syntax:

```
boolean isUpperCase(char c)
```

Source Code:

```
class Sample{  
    public static void main(String args[]){  
        System.out.println(Character.isUpperCase(3));  
        System.out.println(Character.isUpperCase('R'));  
        System.out.println(Character.isUpperCase('r'));  
    }  
}
```

Output:

```
false  
true  
false
```

## isWhitespace - Character Method

isWhitespace() इस Character Method से दिया हुआ character; whitespace है या नहीं इसे boolean; value में return करता है |

Syntax:

```
boolean isWhitespace(char c)
```

Source Code:

```
class Sample{
public static void main(String args[]){
System.out.println(Character.isWhitespace(3));
System.out.println(Character.isWhitespace(' '));
System.out.println(Character.isWhitespace('r'));
}
}
```

Output:

```
false
true
false
```

## toLowerCase - Character Method

toLowerCase() इस Character Method से दिया हुआ Uppercase के character को lowercase में convert कर देता है |

Syntax:

```
char toLowerCase(char c)
```

Source Code:

```
class Sample{
public static void main(String args[]){
System.out.println(Character.toLowerCase(3));
System.out.println(Character.toLowerCase('R'));
System.out.println(Character.toLowerCase('r'));
}
}
```

Output:

```
3
r
r
```

## toUpperCase - Character Method

toUpperCase() इस Character Method से दिया हुआ Lowercase के character को uppercase में convert कर देता है ।

Syntax:

```
char toUpperCase(char c)
```

Source Code:

```
class Sample{
public static void main(String args[]){
System.out.println(Character.toUpperCase(3));
System.out.println(Character.toUpperCase('R'));
System.out.println(Character.toUpperCase('r'));
}
}
```

Output:

```
3
R
R
```

## Math Methods

Java ये कुछ Mathematical Methods प्रदान करता है ।

Java में java.lang.Math इस class के अंतर्गत ये सभी Math के Methods आते है ।

अगर जो Math पढ़ता है , उसे Logarithm, Trigonometry आदि chapter के बारे में पता होता है । वही java में Methods या functions के रूप में होता है ।

जैसे Logarithm के लिए log(), log10() ये Methods या Trigonometry के लिए sin(), cos() आदि Methods दिए गए है ।

Math Methods	Description
abs()	जब इसमें कोई parameter; pass होता है , तब absolute value; return करता है ।
acos()	Arc cosine radians में returns करता है ।
asin()	Arc sine Number को radians में return करता है ।
atan()	Arc tangent Number को radians में return करता है ।
atan2()	Arc tangent Number को radians में return करता है ।

ceil()	जब floating-point value या double type की value यहाँ पर दी जाती है तो ये उसके पास की rounded value return करता है ।
cos()	cosine का angle radians में return करता है ।
cosh()	Hyperbolic cosine का angle radians में return करता है ।
exp()	e को घातांक दिया जाता है । ये double data type की value return करता है ।
floor()	जब floating-point value या double type की value यहाँ पर दी जाती है तो ये उसके पास की rounded value return करता है ।
floorDiv()	इसे किसी integer value से divide किया जाता है और बाद में आनेवाले उत्तर को पासवाले rounded floating-point value को return करता है ।
log()	log() ये method; Number 10 base पर natural logarithm return करता है ।
log10()	log10() ये method; Number का natural logarithm return करता है ।
max()	जो parameters यहाँ पर दिए जाते हैं, उसमें से सबसे ज्यादा value; return की जाती है ।
min()	जो parameters यहाँ पर दिए जाते हैं, उसमें से सबसे कम value; return की जाती है ।
PI	ये एक double type की constant value होती है ।
pow()	pow() ये method 'a' ये value होती है और 'b' ये उस value का घातांक होता है ।
random()	ये दिए हुए number के बीच का random number; return करता है ।
round()	ये Floating-point type की value को rounded value में return करता है ।
sin()	sine का angle radians में return करता है ।
sinh()	Hyperbolic sine का angle radians में return करता है ।
sqrt()	दिए हुए parameter का square root बताता है ।
tan()	Number का tangent का angle radians में return करता है ।
toDegrees()	Number को degrees में convert किया जाता है ।
toRadians()	Number को radians में convert किया जाता है ।

## abs - Math Method

abs() इस Math Method में जब इसमें कोई parameter; pass होता है , तब absolute value; return करता है ।

Syntax:

```
int abs(int i)
float abs(float f)
double abs(double d)
```

जब abs Method में जो parameter होता है , उसका कोई भी primitive data type होता है ।

Source Code:

```
class Sample{
public static void main(String args[]){
int i = -1;
float f = -8.365f;
double d = -9898.65895d;
System.out.println("Absolute Value of " + i + " is " + Math.abs(i));
System.out.println("Absolute Value of " + f + " is " + Math.abs(f));
System.out.println("Absolute Value of " + d + " is " + Math.abs(d));
}
}
```

Output:

```
Absolute Value of -1 is 1
Absolute Value of -8.365 is 8.365
Absolute Value of -9898.65895 is 9898.65895
```

## acos - Math Method

Arc cosine radians में return करता है ।

Arc Cosine में जो Number दिया जाता है वो -1 to +1 के बीच वाला होना चाहिए ।

Syntax:

```
double acos(double Number)
```

Arc cosine ये radians को return करता है । वो radians 0 to PI(3.145) के बीच का होता है ।

अगर radians से degree में convert करना हो तो 180/Math.PI() से multiply करना पड़ता है ।

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = -0.5;
r = Math.acos(Number);
    System.out.println("Arc cosine of " + Number + " = " + r + " in radians");
r = Math.acos(Number)*180/Math.PI; //radians to degree
    System.out.println("Arc cosine of " + Number + " = " + r + " in degree");
}
}
```

Output:

```
Arc cosine of -0.5 = 2.0943951023931957 in radians
Arc cosine of -0.5 = 120.00000000000001 in degree
```

## asin - Math Method

Arc sine Number को radians में return करता है ।

Arc sine में जो Number दिया जाता है वो -1 to +1 के बीच वाला होना चाहिए ।

Syntax:

```
double asin(double Number)
```

अगर radians से degree में convert करना हो तो 180/Math.PI() से multiply करना पड़ता है ।

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 0.5;
r = Math.asin(Number);
System.out.println(r);
r = Math.asin(Number)*180/Math.PI; //radians to degree
System.out.println(r);
}
}
```

Output:

```
0.5235987755982989
30.000000000000004
```



## atan - Math Method

Arc tangent negative या positive value radians में return करता है ।

Syntax:

```
double atan(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 1;
r = Math.atan(Number);
System.out.println(r);
r = Math.atan(Number)*180/Math.PI; //radians to degree
System.out.println(r);
}
}
```

Output:

```
0.7853981633974483
45.0
```

## atan2 - Math Method

Arc tangent2 radians में return करता है ।

atan2() के लिए दो parameters होते है ।

- x-coordinate
- y-coordinate

Syntax:

```
double atan2(double y, double x)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double x, y, r;
x = -5;
y = 5;
r = Math.atan2(y, x)*180/Math.PI; //radians to degree
System.out.println("Arc tangent2 of x = "+x+" y = "+y+" is "+r+" degrees");
}
}
```

Output:

```
Arc tangent2 of x = -5.0 y = 5.0 is 135.0 degrees
```

## ceil - Math Method

ceil() में दिया हुआ Number अपने पासवाले या उससे बड़े या उसके बराबर के integer Number को return करता है ।

Syntax:

```
double ceil(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number1 = 5.7;
double Number2 = 5.1;
double Number3 = 6;
System.out.println("ceil of "+Number1+" is "+Math.ceil(Number1));
System.out.println("ceil of "+Number2+" is "+Math.ceil(Number2));
System.out.println("ceil of "+Number3+" is "+Math.ceil(Number3));
}
}
```

Output:

```
ceil of 5.7 is 6.0
ceil of 5.1 is 6.0
ceil of 6.0 is 6.0
```

## cos - Math Method

cos() function में cosine का angle radians में return करता है |

Syntax:

```
double cos(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 60;
r = Math.cos(Number);
System.out.println("cosine of "+Number+" = "+r+" in radians");
r = Math.cos(Number*Math.PI/180);
System.out.println("cosine of "+Number+" = "+r+" in degrees");
}
}
```

Output:

```
cosine of 60.0 = -0.9524129804151563 in radians
cosine of 60.0 = 0.5000000000000001 in degrees
```

## cos - Math Method

Hyperbolic cosine का angle radians में return करता है |

Syntax:

```
double cosh(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double r;
r = Math.cosh(0);
System.out.println("Hyperbolic cosine of 0 is " + r);
r = Math.cosh(0.5);
System.out.println("Hyperbolic cosine of 0.5 is " + r);
r = Math.cosh(1);
System.out.println("Hyperbolic cosine of 1 is " + r);
}
}
```

Output:

```
Hyperbolic cosine of 0 is 1.0  
Hyperbolic cosine of 0.5 is 1.1276259652063807  
Hyperbolic cosine of 1 is 1.543080634815244
```

## exp - Math Method

e को घातांक दिया जाता है | ये double data type की value return करता है |

Syntax:

```
double exp(double Number)
```

जो number यहाँ पर दिया जाता है, वो e का घातांक होता है |

**Note :** e की value '2.71828183' होती है |

Source Code:

```
class Sample{  
    public static void main(String args[]) {  
        double r;  
        r = Math.exp(10);  
        System.out.println("e^10 = " + r);  
    }  
}
```

Output:

```
e^10 = 22026.465794806718
```

## floor - Math Method

floor() ये function अपने पासवाले या दिए हुए number से छोटे या इससे बराबर के integer number को return करता है |

Syntax:

```
double floor(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double r;
r = Math.floor(-4.5);
System.out.println("floor of -4.5 is " + r);
r = Math.floor(-4.1);
System.out.println("floor of -4.1 is " + r);
r = Math.floor(4.1);
System.out.println("floor of 4.1 is " + r);
r = Math.floor(4.6);
System.out.println("floor of 4.6 is " + r);
r = Math.floor(4.5);
System.out.println("floor of 4.5 is " + r);
}
}
```

Output:

```
floor of -4.5 is -5.0
floor of -4.1 is -5.0
floor of 4.1 is 4.0
floor of 4.6 is 4.0
floor of 4.5 is 4.0
```

## floorDiv - Math Method

इसे किसी integer value से divide किया जाता है और बाद में आनेवाले उत्तर को पासवाले rounded floating-point value को return करता है।

Syntax:

```
double floorDiv(double Numerator, double Denominator)
```

Program में  $-45 / 4 = -11.25$  लेकिन floorDiv में  $-12.0$  ये rounded value मिलती है।

Source Code:

```
class Sample{
public static void main(String args[]) {
double r;
r = Math.floorDiv(-45,4);
System.out.println("floorDiv : " + r);
r = Math.floor(-45/4);
System.out.println("floor : " + r);
r = -45 / 4;
System.out.println("Division : " + r);
}
}
```

Output:

```
floorDiv : -12.0
floor : -11.0
Division : -11.0
```

## log - Math Method

log() ये method; Number का natural logarithm return करता है |

Syntax:

```
double log(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 1.5;
r = Math.log(Number);
System.out.println("Logarithm of " + Number + " is " + r);
r = Math.log(1);
System.out.println("Logarithm of " + 1 + " is " + r);
}
}
```

Output:

```
Logarithm of 1.5 is 0.4054651081081644
Logarithm of 1 is 0.0
```

## log10 - Math Method

log10() ये method; Number का base-10 logarithm return करता है ।

Syntax:

```
double log10(double Number)
```

$$\log_{10}(1.5) = 0.18$$

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 1.5;
r = Math.log10(Number);
System.out.println("Base-10 Logarithm of " + Number + " is " + r);
}
}
```

Output:

```
Base-10 Logarithm of 1.5 is 0.17609125905568124
```

## max - Math Method

max() ये method दिए हुए parameters से सबसे बड़ा number निकालता है ।

Syntax:

```
int max(int Number1, double Number2)
float max(float Number1, float Number2)
double max(double Number1, double Number2)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
int r;
r = Math.max(5, 6);
System.out.println( r );
}
}
```

Output:

```
6
```

## min - Math Method

min() ये method दिए हुए parameters से सबसे छोटा number निकालता है ।

Syntax:

```
int min(int Number1, double Number2)
float min(float Number1, float Number2)
double min(double Number1, double Number2)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
int r;
r = Math.min(5, 6);
System.out.println( r );
}
}
```

Output:

5

## pow - Math Method

pow() ये function 'a' ये value होती है और 'b' ये उस value का घातांक होता है ।

Syntax:

```
double pow(double a, double b);
```

$$a^b = 3^2 = 9$$

Source Code:

```
class Sample{
public static void main(String args[]) {
double a, b, r;
a = 3.00;
b = 2.00;
r = Math.pow(a, b);
System.out.println("pow(" + a + ", " + b + ") = " + r);
}
}
```



Output:

```
pow(3.0, 2.0) = 9.0
```

## random - Math Method

random ये method; multiply किये हुए Number के बीच का random number; return करता है।

Syntax:

```
random()
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double r;
r = Math.random() * 100;
System.out.println("Random Floating-point Number between 0 to 100 : " + r);
}
}
```

Output:

```
Random Number between 0 to 100 : 98.66123485103526
```

## round - Math Method

round() ये method; Number की round value का integer return करता है।

Syntax:

```
double round(double Number)
```

अगर 3.5 value हो तो round value 4.0 होता है। अगर value 3.1 और 3.4 के बीच में हो तो 3.0 round होता है।  
अगर value 3.5 और 3.9 के बीच में हो तो 4.0 round होता है।

Source Code:

```
class Sample{
public static void main(String args[]) {
double a, b, c, d, e, r;
a = 3.5;
b = 3.6;
c = 3.4;
d = 3.9;
e = 3.1;
r = Math.round(a);
System.out.println("round of " + a + " is " + r);
r = Math.round(b);
System.out.println("round of " + b + " is " + r);
r = Math.round(c);
System.out.println("round of " + c + " is " + r);
r = Math.round(d);
System.out.println("round of " + d + " is " + r);
r = Math.round(e);
System.out.println("round of " + e + " is " + r);
}
}
```

Output:

```
round of 3.5 is 4.0
round of 3.6 is 4.0
round of 3.4 is 3.0
round of 3.9 is 4.0
round of 3.1 is 3.0
```

## sin - Math Method

sin() ये method; sine का angle radians में return करता है ।

Syntax:

```
double sin(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 90.0;
r = Math.sin(Number);
System.out.println("sine of " + Number + " is " + r);
r = Math.sin(Number*Math.PI/180);
System.out.println("sine of " + Number + " is " + r);
}
```

Output:

```
sine of 90.0 is 0.8939966636005579
sine of 90.0 is 1.0
```

## sinh - Math Method

sinh() ये method; hyperbolic sine ये value radians में return करता है ।

Syntax:

```
double sinh(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 1.0;
r = Math.sinh(Number);
System.out.println("hyperbolic sine of " + Number + " is " + r);
}
}
```

Output:

```
hyperbolic sine of 1.0 is 1.1752011936438014
```

## sqrt - Math Method

sqrt() ये method; Number का square root return करता है |

Syntax:

```
double sqrt(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 9;
r = Math.sqrt(Number);
System.out.println("Square root of " + Number + " is " + r);
}
}
```

Output:

```
Square root of 9.0 is 3.0
```

## tan - Math Method

tan() ये method; Number का tangent का angle radians में return करता है |

Syntax:

```
double tan(double Number)
```

Source Code:

```
class Sample{
public static void main(String args[]) {
double Number, r;
Number = 6.5;
r = Math.tan(Number);
System.out.println("tangent of " + Number + " is " + r);
Number = -6.5;
r = Math.tan(Number);
System.out.println("tangent of " + Number + " is " + r);
}
}
```

Output:

```
tangent of 6.5 is 0.22027720034589682  
tangent of -6.5 is -0.22027720034589682
```

## toDegrees - Math Method

toDegrees() ये method; angle के radians को degrees में convert करता है ।

Syntax:

```
double toDegrees(double Number)
```

Source Code:

```
class Sample{  
    public static void main(String args[]) {  
        double degrees;  
        degrees = Math.toDegrees(Math.PI);  
        System.out.println("Degrees : " + degrees);  
        degrees = Math.toDegrees(Math.PI / 2);  
        System.out.println("Degrees : " + degrees);  
        degrees = Math.toDegrees(Math.PI * 2);  
        System.out.println("Degrees : " + degrees);  
    }  
}
```

Output:

```
Degrees : 180.0  
Degrees : 90.0  
Degrees : 360.0
```

## toRadians - Math Method

toRadians() ये method; angle के degrees को radians में convert करता है ।

Syntax:

```
double toDegrees(double Number)
```

#### Source Code:

```
class Sample{  
    public static void main(String args[]) {  
        double radians;  
        radians = Math.toRadians(180);  
        System.out.println("Radians : " + radians);  
        radians = Math.toRadians(90);  
        System.out.println("Radians : " + radians);  
        radians = Math.toRadians(360);  
        System.out.println("Radians : " + radians);  
    }  
}
```

#### Output:

```
Radians : 3.141592653589793  
Radians : 1.5707963267948966  
Radians : 6.283185307179586
```

# Special Keywords

## Static Keyword

Static keyword के दो हिस्सों में काम करता है ।

1. Static/Class Variable
2. Static/Class Method

Instance variables मतलब जिनका इस्तेमाल बिना static keyword के होता है ।

इनके बारे में देखे तो instance variable जब class में declare किया जाता है, तब उस class का object या instance बनाने के बाद उसकी copy; object पर होती है । अगर एक class के multiple objects हो तो हर object पर instance variable की अलग-अलग copies होती है ।

### 1. Static/Class Variable

Static Variable ये अपने class के लिए होता है, इसीलिए इसे Class Variable कहते हैं ।

Static Variable Object या instance के लिए नहीं होता ।

Static Variable के लिए object; create करने की जरूरत नहीं होती ।

Static Variable को class के नाम से access किया जाता है ।

Class के हर object पर static variable की एक-एक copy होती है ।

अगर Memory के बारे में देखे तो Static variable काफी Memory save करता है ।

Static Variable; Instance Variable जैसा इस्तेमाल हो सकता है, लेकिन Instance Variable; Static Variable जैसा इस्तेमाल नहीं हो सकता ।

```
int a; // Instance Variable
static int b; // Static/Class Variable
```

### Syntax for Accessing Static Variable

```
//Accessing for Instance Variable by Class Instance/Object
class_instance.variable_name = variable_value;
//Accessing for Static/Class Variable by class name
class_name.static_variable_name = static_variable_value;
```

### Example for Accessing Static Variable

```
//Accessing for Instance Variable by Class Instance/Object
obj.a = 5;
//Accessing for Static/Class Variable by class name
Sample.b = 10;
```

Source Code:

```
class Sample{
static int a;
Sample(){
a++;
}

public static void main(String args[]){

Sample s1 = new Sample();
Sample s2 = new Sample();
System.out.println("Value of a : " + a);
System.out.println("Value of a : " + a);
}
}
```

Output:

```
Value of a : 1
Value of a : 1
```

## 2. Static/Class Method

Static Method ये अपने class के लिए होता है, इसीलिए इसे Class Method कहते हैं।

Static Method; Object या instance के लिए नहीं होता।

Static Method के लिए object; create करने की जरूरत नहीं होती।

Static Method को class के नाम से access किया जाता है।

Class के हर object पर Static Method की एक-एक copy होती है।

Static Method; Instance Method जैसा इस्तेमाल हो सकता है, लेकिन Instance Method; Static Method जैसा इस्तेमाल नहीं हो सकता।

```
int disp(){  }; // Instance Method
static int show(){  }; // Static/Class Method
```

### Syntax for Accessing Static Method

```
//Accessing for Instance Method by Class Instance/Object
class_instance.instance_method_name();
//Accessing for Static/Class Method by class name
class_name.static_method_name();
```



## Example for Accessing Static Method

```
//Accessing for Instance Method by Class Instance/Object  
obj.disp();  
//Accessing for Static/Class Method by class name  
Sample.show();
```

Source Code:

```
class Sample{  
  
    void disp(){  
        System.out.println("Calling with object.");  
    }  
    static void show() {  
        System.out.println("Calling without object or class.");  
    }  
    public static void main(String args[]){  
        Sample s = new Sample();  
        s.disp(); //Calling with instance  
        show(); //Calling without instance  
        Sample.show(); //Calling with class  
    }  
}
```

Output:

```
Calling with object.  
Calling without object or class.  
Calling without object or class.
```

# Final Keyword

Java में final keyword से तीन काम किये जाते हैं ।

1. final Variable
2. final Method
3. final Class

## 1. final Variable

final keywords; constants जैसे होते हैं । एक बार final keyword के साथ variable initialized हो जाए तो दोबारा उसकी value; assign नहीं हो सकती ।

Source Code:

```
class Sample{
public static void main(String args[]){
final int a = 5;
a = 10;
System.out.println("Value of a : " + a);
}
}
```

Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac Sample.java
Sample.java:5: error: cannot assign a value to final variable a
a = 10;
^
1 error
```

## Blank or Uninitialized final Variable

C और C++ के बारे में देखे तो const keyword के साथ जो variable लिखा जाता है उसे एक साथ ही declare और initialize किया जाता है ।

Java में final keyword; c++ को const keyword जैसा ही होता है, लेकिन उसे पहले declared करके बाद में initialize किया तो भी चलता है ।

Source Code:

```
class Sample{
    public static void main(String args[]){
        final int a;
        a = 5;
        System.out.println("Value of a : " + a);
    }
}
```

Output:

```
Value of a : 5
```

## 2. final Method

child class के साथ final method override नहीं होता ।

Source Code:

```
class A{
    public final void disp(){
        System.out.println("In class A");
    }
}
class B extends A{
    public void disp(){
        System.out.println("In class B");
    }
}
public class Sample{
    public static void main(String args[]) {
        B b = new B();
        b.disp();
    }
}
```

Output:

```
Sample.java:8: error: disp() in B cannot override disp() in A
    public void disp(){
            ^
    overridden method is final
1 error
```

### 3. final Class

final parent class को किसी child class के साथ inherit नहीं किया जा सकता

Source Code:

```
final class A{
void show(){
System.out.println("class A");
}
}
class B extends A{
void disp(){
System.out.println("class B");
}
    public static void main(String args[]){
        B b= new B();
        b.show();
        b.disp();
    }
}
```

Output:

```
Sample.java:7: error: cannot inherit from final A
class B extends A{
    ^
1 error
```

# Super Keyword

super Keyword; Java के लिए काफी प्रभावशाली साबित हुआ है | जब super keyword का इस्तेमाल करना हो तो Inheritance का भी इस्तेमाल होता है | super Keyword ये एक reference variable है, जो parent class के object को refer करता है |

super Keyword तीन चीजों के लिए java में काम करता है |

1. For Variable
2. For Method
3. For Constructor

## 1. super Keyword For Variable

निचे दिए हुए example में parent-class में और child-class में एक ही variable को initialized किया हुआ है | Parent-class के variable को access करने के लिए कोई पर्याय ही नहीं है |

### Example Without super Keyword

```
//B.java
class A{
int a = 5;
}
class B extends A{
int a = 10;
void show(){
System.out.println("Value of a : " + a); // print for class B
}
public static void main(String args[]){
B b = new B();
b.show();
}
} // closing class B
```

Output:

```
Value of a : 10
```

## Example With super Keyword

Syntax:

```
super.variable_name
```

निचे दिए हुए program में child-class(class B) पर दिए super reference variable से parent-class(class A) के variable को access किया जाता है ।

Source Code:

```
//B.java
class A{
int a = 5; // print for class A
}
class B extends A{
int a = 10;
void show(){
System.out.println("Value of a : " + super.a);
}
public static void main(String args[]){
B b = new B();
b.show();
}
} // closing class B
```

Output:

```
Value of a : 5
```

## 2. super Keyword For Method

Syntax:

```
super.method_name();
```

अगर parent-class और child-class में same method हो तो child-class का method; invoked होता है । लेकिन super keyword के साथ method का इस्तेमाल किया जाता है, तो parent-class का method; invoked होता है ।

Source Code:

```
//B.java
class A{
void show(){
System.out.println("class A");
}
}
class B extends A{
void show(){
System.out.println("class B");
}
void disp(){
show();    //print for class B
super.show(); //print for class A using super
}
public static void main(String args[]){
B b = new B();
b.disp();
}
} // closing class B
```

Output:

```
class B
class A
```

### 3. super Keyword For Constructor

निचे Constructor के दो examples दिए हुए है एक without super() है और दूसरा with super() का है और दोनों ही examples का output एक जैसा ही है । इसका मतलब दुसरे example में दिया हुआ super() को call करना किसी काम का नहीं है । अगर program में जब constructor create होता है, तब by default super() वहा पर call हो जाता है ।

## Without using super()

Source Code:

```
class A{
A(){
System.out.println("class A"); // print class A
}
}
class B extends A{
B(){
System.out.println("class B"); // print class B
}
public static void main(String args[]){
B b = new B();
}
} // closing class B
```

Output:

```
class A
class B
```

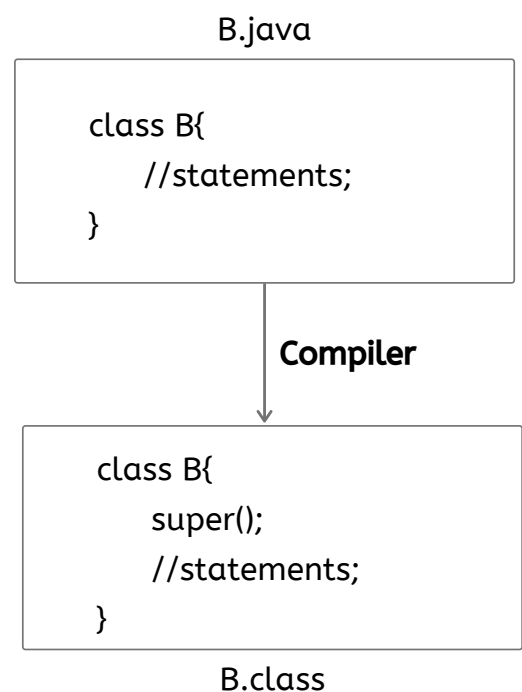
## With using super()

Source Code:

```
class A{
A(){
System.out.println("class A"); // print class A
}
}
class B extends A{
B(){
super();
System.out.println("class B"); // print class B
}
public static void main(String args[]){
B b = new B();
}
} // closing class B
```

Output:

```
class A
class B
```





# This Keyword

Java में 'this' ये एक keyword है | जो java में एक reference variable है |

this keyword से current class के instance variable को refer किया जाता है |

यहाँ पर program में Instance variable और Local variable ये दोनों same ही है | इसीलिए जब instance variable(left) की value; local variable(right) पर assign की जाती है, तब compiler को समझ नहीं आता कि, कौनसा variable; instance है और कौनसा variable local है |

Source Code:

```
class Sample{
int a; // instance variable
int b; // instance variable
Sample(int a, int b){ // a and b is local variable
a = a;
b = b;
}
void show(){
System.out.println("Value of a : " + a);
System.out.println("Value of b : " + b);
}
public static void main(String args[]){
Sample s = new Sample(5, 6);
s.show();
}
}
```

Output:

```
Value of a : 0
Value of b : 0
```

## Same Operation without using this keyword

Source Code:

```
class Sample{
    int a;
    int b;
    Sample(int x, int y){
        a = x;
        b = y;
    }
    void show(){
        System.out.println("Value of a : " + a);
        System.out.println("Value of b : " + b);
    }
    public static void main(String args[]){
        Sample s = new Sample(5, 6);
        s.show();
    }
}
```

Output:

```
Value of a : 5
Value of b : 6
```

Program में this keyword के साथ Instance variable और Local variable इन दोनों variables को अलग-अलग किया गया है ।

Source Code:

```
class Sample{
    int a;
    int b;
    Sample(int a, int b){
        this.a = a; //Instance Variable = Local Variable;
        this.b = b; //Instance Variable = Local Variable;
    }
    void show(){
        System.out.println("Value of a : " + a);
        System.out.println("Value of b : " + b);
    }
    public static void main(String args[]){
        Sample s = new Sample(5, 6);
        s.show();
    }
}
```

Output:

```
Value of a : 5  
Value of b : 6
```

## Using this() without parameter with Constructor

this() ये constructor के लिए पहला statement होता है | current class के constructor को invoked करने के लिए भी this() का इस्तेमाल किया जाता है |

Source Code:

```
class Sample{  
    Sample(){  
        System.out.println("Default Constructor.");  
    }  
    Sample(int a){  
        this(); //Default Constructor invoke first  
        System.out.println("Parameterized Constructor.");  
        System.out.println("Value of a : " + a);  
    }  
    public static void main(String args[]){  
        Sample s = new Sample(5);  
    }  
}
```

Output:

```
Default Constructor.  
Parameterized Constructor.  
Value of a : 5
```

## Using this() with parameter with Constructor

Source Code:

```
class Sample{
    Sample(){
        this(5); //parameterized constructor invoke first
        System.out.println("Default Constructor.");
    }
    Sample(int a){
        System.out.println("Parameterized Constructor.");
        System.out.println("Value of a : " + 5);
    }
    public static void main(String args[]){
        Sample s = new Sample();
    }
}
```

Output:

```
Parameterized Constructor.
Value of a : 5
Default Constructor.
```

## This keyword using with class method

current class method को invoke करने के लिए 'this' keyword का इस्तेमाल किया जाता है ।

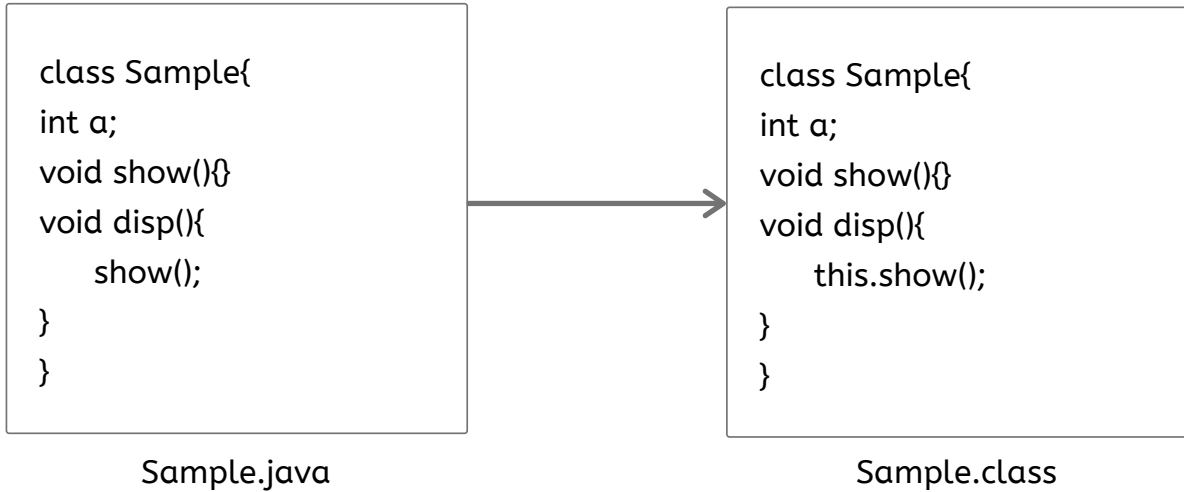
Source Code:

```
class Sample{
    int a;
    void show(int a){
        System.out.println("Value of a : " + a);
    }
    void disp(){
        this.show(5);
    }
    public static void main(String args[]){
        Sample s = new Sample();
        s.disp();
    }
}
```

Output:

Value of a : 5

ऊपर दिए हुए example में method के साथ this keyword को इस्तेमाल करने की जरूरत नहीं है | Compiler द्वारा automatically ये this keyword method को दिया जाता है |



# Exception Handling

## Introduction of Exception Handling

जब program execution के वक्त कोई problem आ जाता है, तो उसे Exception कहते हैं।

Exception ये runtime पर आता है।

Java में दो प्रकार के Errors होते हैं।

1. Compile-Time Error

2. Run-Time Error

**Compile-Time Error** : ये एक normal Errors है जो Compiler द्वारा आ जाता है। जब Program में कहीं syntax में, कहीं curly brace, semi-colon या comma नहीं दिया जाता है, तो ये Compile-Time पर Error occur होता है।

**Run-Time Error** : यहाँ Program successfully run होता है। लेकिन कुछ ऐसी internal errors आ जाती हैं, जो interpreter द्वारा दी जाती हैं। इससे Program भी बंद हो जाता है।

Exception के बारे में देखें तो, जब Program compile-time पर successfully run होता है, तो program में कुछ statements ऐसे होते हैं कि वो Compiler द्वारा execute नहीं होते हैं, उसी वक्त interpreter द्वारा उस statements के सम्बंधित वो एक object create करता है, जब object create किया जाता है, तब interpreter द्वारा Run-Time पर उसे throw किया जाता है।

इससे ये समझ आता है कि, जब भी कभी ऐसी condition program में आती है, तब Exception Handling का इस्तेमाल Program में किया जाता है।

## Exception और Error का फर्क

**Exception** : Exception Run-time पर आ जाता है, जब Program; में divided by zero पर Exception आ जाता है।

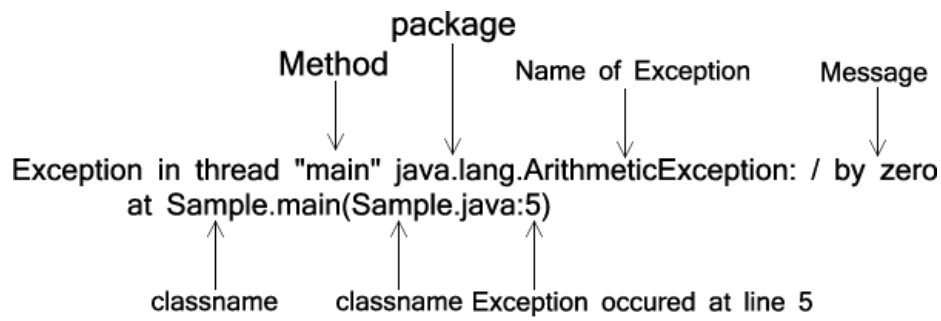
**Error** : जब Program में Syntax error, format error या program; out of memory जाता है, तो Error आ जाती है। ये errors; critical होती हैं। इन error को ठीक करना काफी कठिन होता है। ये Error Compile-time पर आ जाता है।

## Check Example for Normal Exception

```
class Sample{
    public static void main(String args[]){
        int a = 4/0;
    }
}
```

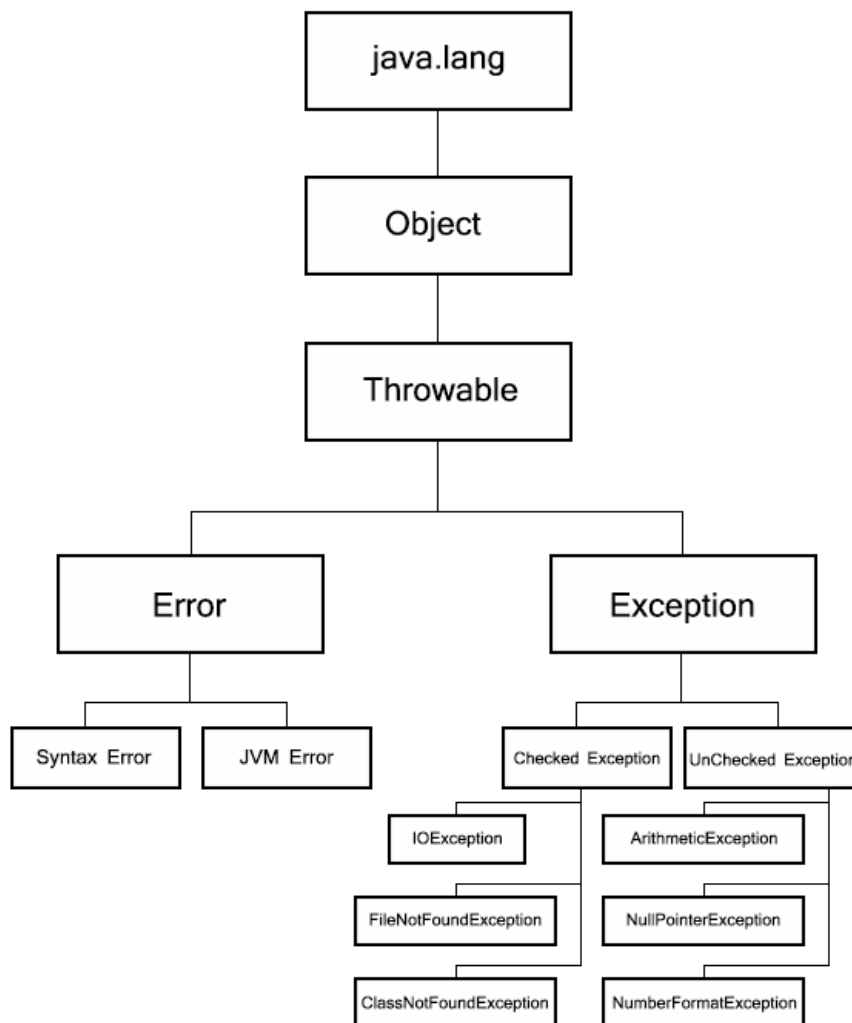
Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac Sample.java
C:\Program Files\Java\jdk1.8.0_111\bin>java Sample
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Sample.main(Sample.java:5)
```



ऊपर के program में देखे तो ये Run-Time पर ArithmeticException ये Exception occur होता है | ये एक User का खराब Programming का उदाहरण है | यहाँ पर Runtime पर User को warning दी जाती है | Program पर आया हुआ Exception काफी अच्छे से समझ आता है |

## Java Exception Classes



Program में Exception Handling में दो प्रकार के काम करता है ।

- 1.Checked Exception
- 2.Un-checked Exception

## Checked Exception

Checked Exception ये compile time पर occur होता है । इसे Compile-time Exception भी कहा जाता है । जब program में किसी प्रकार की checked exception आती है तो User को compiler द्वारा बताया जाता है कि आप उस Exception को handle करें ।

For Example, IOExceptionClassNotFoundException आदि.

Program में देखें तो FileInputStream data को read करने के लिए इस्तेमाल किया गया है । FileInputStream से FileNotFoundException; ये Exception occur होता है । ये सब compile-time पर होता है और Programmer को Exception handle करने के लिए बताया जाता है ।

### Source Code

```
import java.io.FileInputStream;
class Sample{
public static void main(String args[]){
FileInputStream file = null;
file = new FileInputStream("file.txt");
}
}
```

### Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac Sample.java
Sample.java:7: error: unreported exception FileNotFoundException; must be caught or
declared to be thrown
    file = new FileInputStream("file.txt");
    ^
1 error
```



निचे दिया हुआ program Exception Handling का है | जब कोई file found नहीं होती तो एक message display होता है | Exception Handling के लिए try, catch का इस्तेमाल किया गया है |

#### Source Code

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
class Sample{
public static void main(String args[]){
FileInputStream file = null;
try{
file = new FileInputStream("file.txt");
}
catch(FileNotFoundException e){
    System.out.println("File is not Found.");
}
}
}
```

#### Output:

```
File is not Found.
```

## Un-checked Exception

Un-checked Exception ये run time पर occur होता है | इसे Run-time Exception भी कहा जाता है | इसे compile-time पर check नहीं किया जाता |जब इसे programmer द्वारा इसे handle नहीं किया जाता तो JVM इसे handle कर देता है |

For Example,ArithmeticException, NullPointerException, NumberFormatException आदि.

#### Source Code

```
class Sample{
public static void main(String args[]){
int a = 4/0;
}
}
```

#### Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac Sample.java
C:\Program Files\Java\jdk1.8.0_111\bin>java Sample
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Sample.(Sample.java:3)
    at Sample.main(Sample.java:6)
```

# try and catch

try और catch ये दो keywords; Exception Handling में इस्तेमाल किये जाते हैं। जब Program पर Run-time पर exception आ रहा है, तो उसे control करने के लिए try और catch का इस्तेमाल किया जाता है।

## try क्या करता है ?

try ये एक block होता है जिसमें statements होते हैं। जो statements जब Exception occur करता है, तब उसे try block पर लिखा जाता है।

Syntax:

```
try{  
    //statements;  
}
```

## catch क्या करता है ?

ये catch block; try block से related होता है। जब try block पर कोई Exception occur होता है, तब उसका flow; catch block पर चला जाता है और catch block का statement print किया जाता है।

try पर जिसका Exception occur कर हो रहा है, उसके Exception के साथ उसका object catch के parenthesis() में लिखा जाता है।

Syntax:

```
try{  
    //statements;  
}catch(Exception object){  
    //statements;  
}
```

## Example for try and catch Block

निचे दिए program पर try block में divide by zero और a की value को display करने के लिए statement दिया हुआ है। अगर जब try block पर Exception occur होता है, तो उसका statement भी print नहीं होता। Exception occur होने की वजह से exception; catch पर throw होता है और program successfully execute होता है।

Source Code:

```
class Sample{
public static void main(String args[]){
try{
int a = 4/0;
System.out.println("Value of a : " + a);
}catch(ArithmeticException e){
System.out.println("Divided by Zero.");
}
}
}
```

Output:

```
Divided by Zero.
```

### Multiple catch Blocks

यहाँ पर एक से अधिक catch blocks होते हैं। जिसके लिए एक से अधिक Exceptions होते हैं। यहाँ पर एक try block के साथ कितने भी catch blocks दिए जा सकते हैं। Program में try statement में जो Exception occur होता है, तो वो पहले catch पर जाता है। अगर पहले catch पर exception नहीं मिलता तो वो दूसरे catch पर चला जाता है। अगर दूसरे catch पर भी Exception नहीं मिलता तो Run-Time पर interpreter द्वारा Object create होता है।

Source Code:

```
class Sample{
public static void main(String args[]){
try{
//statement1
int i = 4/0; // divided by zero
int[] a = new int[2]; //statement2
a[0] = 1;
a[1] = 2;
a[2] = 3; // extra initialization for array
System.out.println("This statement for try Block");
}
catch(ArithmeticException e){
System.out.println("Divided by Zero error");
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("Array Out of Bound error");
}
}
}
```

Output:

Divided by Zero error

### ऊपर दिए हुए Example से थोडासा अलग Program

यहाँ पर try Block में दो statements है और दोनों ही Exception occur कर रही है | उपरवाले program के मुकाबले सिर्फ उनकी position change की गयी है |

Source Code:

```
class Sample{
public static void main(String args[]){
try{
int[] a = new int[2];      //statement1
a[0] = 1;
a[1] = 2;
a[2] = 3; // extra initialization for array
int i = 4/0;  // divided by zero
//statement2
System.out.println("This statement for try Block");
}
catch(ArithmeticException e){
System.out.println("Divided by Zero error");
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("Array Out of Bound error");
}
}
}
```

Output:

Array Out of Bound error

### Nested try catch Statements

Nested try catch में try Block के अन्दर और भी कई try catch Block हो सकते है | कई बार Program में exception के बारे में ऐसी स्थिति आती है, कि Nested try catch Blocks का इस्तेमाल करना पड़े | इसीलिए nested try catch सिखना उतना ही जरूरी है, जितना try catch है |

## Syntax for try..catch Statement

```
try{           //outer try
//statement1;
//statement2;
try{ //inner try
//statement1;
//statement2;
}
catch(Exception object1){ //inner catch
//statements;
}
}
catch(Exception object2){ //outer catch
//statements;
}
```

## Example for try...catch statement

```
class Sample{
public static void main(String args[]){
try{
int a = 4;
int b = 0;
try{
int c = a / b;           //inner try statement
}
catch(ArithmeticException e1){
System.out.println("divided by zero"); //inner catch statement
}
int[] arr = new int[3];    //outer try statement
arr[3] = 2;
} //close outer try
catch(ArrayIndexOutOfBoundsException e2){
System.out.println("array out of bound"); //outer catch statement
} //close outer catch
} //close main method
} //close class
```

Output:

```
divided by zero
array out of bound
```

# finally, throw and throws

finally, throw और throws ये तीन keywords; Exception Handling में इस्तेमाल किये जाते हैं।

## Introduction for finally Block

finally Block को सिर्फ try Block के साथ या try...catch Block के साथ इस्तेमाल किया जाता है।

finally हर वक्त execute होता है चाहे try के साथ exception occur होता है या नहीं होता है।

Syntax:

```
try{
//statements;
}
catch(Exception object){
//statements;
}
finally{
//statements;
}
```

## Example1 for finally Block

निचे दिए हुए program में exception occur कर रहा है और उसे handle भी किया गया है।

```
public class Sample{
public static void main(String args[]){
try{
int a = 4 / 0;

System.out.println("Value of a : " + a);
}
catch(ArithmeticException e){
System.out.println("ArithmeticException");
}
finally{
System.out.println("finally statement is always print.");
}
}
}
```

Output:

```
ArithmeticException
finally statement is always print.
```

### Example2 for finally Block

निचे दिए हुए program में exception occur कर रहा है और उसे handle नहीं किया गया है ।

```
public class Sample{
    public static void main(String args[]){
        try{
            int a = 4 / 0;

            System.out.println("Value of a : " + a);
        }
        finally{
            System.out.println("finally statement is always print");
        }
    }
}
```

Output:

```
finally statement is always print
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Sample.main(Sample.java:6)
```

### Example3 for finally Block

निचे दिए हुए program में exception occur नहीं कर रहा है लेकिन उसे handle किया गया है ।

```
public class Sample{
    public static void main(String args[]){
        try{
            int a = 4 / 2;
            System.out.println("Value of a : " + a);
        }
        catch(ArithmeticException e){
            System.out.println("ArithmeticException");
        }
        finally{
            System.out.println("finally statement is always print");
        }
    }
}
```

Output:

```
Value of a : 2
finally statement is always print
```

## Introduction for throw and throws

Java में throw और throws ये काफी फायदेमंद keywords हैं, जिनका इस्तेमाल Java में Exception Handling के लिए किया जाता है।

throw keyword के साथ Exception के object को throw किया जाता है।

निचे दिया हुआ code; उस Exception को handle नहीं कर रहा, सिर्फ throw कर रहा है।

```
void data(){  
    throw new ArithmeticException();  
}
```

जब इसे handle करने की बारी आती तब throws का इस्तेमाल किया जाता है। लेकिन जहाँ पर भी उस method को call किया जाता है वहाँ पर उसे handle करना पड़ता है। अगर programmer द्वारा handle नहीं किया जाता तो automatically JVM उसे handle करता है।

```
void data() throws ArithmeticException{  
    throw new ArithmeticException();  
}
```

या इसे try..catch Block से handle किया जाए।

```
void data(){  
    try{  
        throw new ArithmeticException();  
    }catch(ArithmeticException e){  
        System.out.println(e);  
    }  
}
```

निचे दिए हुए code पर ArithmeticException ये Un-checked Exception होने के कारण JVM उसे खुद handle करेगा।

```
void data(){  
    throw new ArithmeticException(); //check by JVM  
}
```

**Note :** ऊपर दिए हुए examples Run-Time(Un-checked) Exception के हैं ये JVM द्वारा handle किया जा सकता है, लेकिन अगर checked exception का इस्तेमाल किया जाता है तो Programmer को खुद उसे handle करना पड़ता है।



## Introduction of File Handling

Java में File Handling के लिए java.io इस package का इस्तेमाल किया जाता है | java.io package पर input.output के लिए सभी classes मौजूद होते हैं |

Program में File Handling के लिए streams का इस्तेमाल किया जाता है |

Java के java.io package ऐसे classes हैं जिनको दो streams में विभाजित किया गया है |

1. Byte Streams
2. Character Streams

## Byte Streams

Byte Streams 8-bit bytes से input और output करने के लिए इस्तेमाल किया जाता है |

यहाँ पर Byte Stream के भी दो प्रकार होते हैं |

1. **InputStream** : Input Streams; source से data को read करने के लिए इस्तेमाल किये जाते हैं |
2. **OutputStream** : Output Streams; destination पर data को write करने के लिए इस्तेमाल किये जाते हैं |

Java I/O के लिए दो महत्वपूर्ण methods बनाये गए हैं |

**public int read() throws IOException** : यहाँ stream से byte को read किया जाता है |

**public void write(int i) throws IOException** : यहाँ stream पर byte को write किया जाता है |

Byte Streams में कुछ महत्वपूर्ण classes बनाये गए हैं |

Byte Stream Classes	Byte Stream Classes
BufferedInputStream	Buffered Input Stream के लिए इस्तेमाल किया जाता है
BufferedOutputStream	Buffered Output Stream के लिए इस्तेमाल किया जाता है
DataInputStream	यहाँ पर Data Input Stream के लिए कुछ methods बनाये गए हैं
DataOutputStream	यहाँ पर Data Output Stream के लिए कुछ methods बनाये गए हैं
FileInputStream	file से bytes के रूप के data को read किया जाता है
FileOutputStream	Data को file में write करने के लिए इस्तेमाल किया जाता है
InputStream	ये सभी input streams का superclass है
OutputStream	ये सभी Output streams का superclass है
PrintStream	यहाँ पर print() और println() ये basic methods होते हैं

## Example for Reading file using BufferedInputStream

यहाँ पर stream से data read करने के लिए BufferedStream का इस्तेमाल किया जाता है ।

**file.txt**

Example for BufferedInputStream.

Source Code

```
import java.io.*;
public class Sample{
public static void main(String args[]){
try{
FileInputStream fis = new FileInputStream("file.txt");
    BufferedInputStream bis = new BufferedInputStream(fis);
    int i;
while((i=bis.read())!=-1){
System.out.print((char)i);
}
    fis.close();
bis.close();
}catch(FileNotFoundException e1){
System.out.println("File is not Found");
}
catch(Exception e2){
System.out.println("Reading File Error");
}
}
}
```

Output :

Example for BufferedInputStream.

## Example for Writing file using BufferedOutputStream

BufferedOutputStream को data store करने के लिए internal buffer का इस्तेमाल किया जाता है ।

### Source Code

```
import java.io.*;

public class Sample{

    public static void main(String args[]){
        try{
            FileOutputStream fos = new FileOutputStream("file1.txt");
            BufferedOutputStream bos = new BufferedOutputStream(fos);
            String str = "Example for BufferedOutputStream.";
            byte by[] = str.getBytes();
            bos.write(by);
            bos.close();
            fos.close();
            System.out.println("File is created successfully");
        }
        catch(Exception e){
            System.out.println("File is not created");
        }
    }
}
```

### Output :

File is created successfully

### file1.txt

Example for BufferedOutputStream.

## Writing on File Using DataOutputStream

DataOutputStream से standard data type से data को write किया जाता है ।

### Source Code

```
import java.io.*;

public class Sample {
    public static void main(String args[]){
        int id = 1;           //this data written to file
        String emp_name = "Rakesh Sharma";
        float salary = 50000.52f;
        try {
            FileOutputStream fos = new FileOutputStream("file.txt");
            DataOutputStream dos = new DataOutputStream(fos);
            dos.writeInt(id);
            dos.writeUTF(emp_name);
            dos.writeFloat(salary);
            dos.close();
            System.out.println("Data written successfully.");
        }catch(IOException e){
            System.out.println(e);
        }
    }
}
```

### Output :

Data written successfully.

### file.txt

Rakesh SharmaGCP...

## Reading File Using DataInputStream

DataInputStream से standard data type से data को read किया जाता है ।

### Source Code

```
import java.io.*;
class Sample1{
public static void main(String[] args){
try{
FileInputStream fis = new FileInputStream("file.txt");
DataInputStream dis = new DataInputStream(fis);
System.out.println("id : " + dis.readInt());
System.out.println("Name : " + dis.readUTF());
System.out.println("Salary : " + dis.readFloat());
} catch(IOException e) {
System.out.println(e);
}
}
```

### file.txt

Rakesh SharmaGCP...

### Output :

```
id : 1
Name : Rakesh Sharma
Salary : 50000.52
```

## Writing on File Using FileOutputStream

### Source Code

```
import java.io.*;

public class Sample {
    public static void main(String args[]){
        try{
            FileOutputStream fos=new FileOutputStream("file.txt");
            String str="Hello World!";

            byte b[]=str.getBytes();

            fos.write(b);
            fos.close();
            System.out.println("File written successfully");

        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

### Output :

Data written successfully.

### file.txt

Hello World!

## Reading File Using FileInputStream

**file.txt**

Hello World!

Source Code

```
import java.io.*;
public class Sample {
    public static void main(String args[]){
        try{
            FileInputStream fis=new FileInputStream("file.txt");
            int i=0;
            while((i=fis.read())!= - 1){
                System.out.print((char)i);
            }
            fis.close();

        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Output :

Hello World!

## Writing on File Using PrintStream

### Source Code

```
import java.io.*;

public class Sample{
    public static void main(String args[]){
        try{
            FileOutputStream fos = new FileOutputStream("file.txt");
            PrintStream ps = new PrintStream(fos);
            ps.println("Hello World!");

            ps.close();
            fos.close();
            System.out.println("Data Written Successfully");
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

### Output :

File written successfully.

### file.txt

Hello World!



# Character Streams

Stream पर Character data को read और write करने के लिए character streams का इस्तेमाल किया जाता है। Character Stream के लिए दो Abstract class बनाये गए हैं।

- Reader
- Writer

यहाँ पर character streams के लिए कुछ महत्वपूर्ण classes बनाये गए हैं।

Byte Stream Classes	Byte Stream Classes
BufferedReader	Buffer से character read करने के लिए कुछ Methods बनाये गए हैं।
BufferedWriter	Buffer से character write करने के लिए कुछ Methods बनाये गए हैं।
FileReader	ये एक input stream है जो file को read करता है।
FileWriter	ये एक output stream है जो file पर write करता है।
InputStreamReader	byte से character में convert करने के लिए यहाँ पर Methods दिए गए हैं।
OutputStreamReader	character से byte में convert करने के लिए यहाँ पर Methods दिए गए हैं।
PrintWriter	ये एक output stream है जिसमें print() और println() ये methods होते हैं।
Reader	ये एक Abstract class है जिसमें character input streams होते हैं।
Writer	ये एक Abstract class है जिसमें character output streams होते हैं।

## Reading File Using BufferedReader

file.txt

Hello World!

Source Code

```
import java.io.*;

class Sample {
    public static void main(String args[]){
    try{
    FileReader fr = new FileReader("file.txt");
    BufferedReader br = new BufferedReader(fr);

    int i;
    while((i=br.read())!= - 1){
    System.out.print((char)i);
    }
    br.close();
    fr.close();
    }
    catch(Exception e){
    System.out.println(e);
    }
    }
}
```

Output :

Hello World!

## Writing on File Using BufferedWriter

### Source Code

```
import java.io.*;

class Sample {
    public static void main(String args[]){
    try{
    FileWriter fw = new FileWriter("file.txt");
    BufferedWriter bw = new BufferedWriter(fw);

    bw.write("Hello World!");
    bw.close();
    fw.close();
    System.out.println("Data written successfully.");
    }
    catch(Exception e){
    System.out.println(e);
    }
    }
}
```

### Output :

Data written successfully.

### file.txt

Hello World!

## Writing on File Using PrintWriter

### Source Code

```
import java.io.*;

class Sample {
    public static void main(String args[]){
    try{
    FileWriter fw = new FileWriter("file.txt");
    PrintWriter pw = new PrintWriter(fw);

    pw.write("Hello Friends!");
    pw.close();
    System.out.println("Data written successfully.");
    }
    catch(Exception e){
    System.out.println(e);
    }
    }
}
```

### Output :

Data written successfully.

### file.txt

Hello Friends!

## Writing on File Using FileWriter

### Source Code

```
import java.io.*;

class Sample {
    public static void main(String args[]){
    try{
    FileWriter fw = new FileWriter("file.txt");
    fw.write("Hello World!");
    fw.close();
    System.out.println("Data written successfully.");
    }
    catch(Exception e){
    System.out.println(e);
    }
    }
}
```

### Output :

Data written successfully.

### file.txt

Hello World!

## Reading File Using FileReader

file.txt

Hello World!

### Source Code

```
import java.io.*;

class Sample {
    public static void main(String args[]){
    try{
    FileReader fr = new FileReader("file.txt");

    int i;
    while((i=fr.read())!= - 1){
    System.out.print((char)i);
    }
    fr.close();
    }
    catch(Exception e){
    System.out.println(e);
    }
    }
}
```

### Output :

Hello World!

# Classes and Objects

Classes और Objects यह Object Oriented Programming का basic concept है | संपूर्ण Java यह Classes और Objects के साथ जुड़ा हुआ है जिसमे उसके attributes(states) और methods(behaviors) भी शामिल है | उदाहरण में , जैसे कि **मनुष्य** यह एक Object है | उसके attributes उसका नाम, उम्र और शिक्षा है और methods जैसे कि हाथों से खाना, आँखों से देखना और पैरों से चलना है |

## Class क्या होता है ?

Programmer द्वारा बनाया गया एक Class यह user-defined blueprint या template है जिसमे Object को बनाया जाता है और यह Object को program के उपयोग में लाया जाता है |

Source Code

```
public class Man{  
    //attributes  
    String name;  
    int age;  
    String qualification;  
  
    //methods  
    public void walking(){  
    }  
    public void see(){  
    }  
    public void eating(){  
    }  
}
```

उपर्युक्त उदाहरण में Man यह class name है | हम इस class के माध्यम से किसी भी नामक मनुष्य का वर्णन कर सकते है | यहाँ पर name, age और qualification को man के states को दर्शाता है और walking, see और eating या behavior को दर्शाता है |

## Constructor

जब Java में class बनाया जाता है तब Compiler द्वारा by default उसमे empty constructor बनता है | अगर आप कोई भी constructor नहीं बनाते है तो उपर्युक्त class का निचे constructor दिया हुआ है |

```
Man(){}
```

जब Object बनाया जाता है तब कम से कम एक एब बार constructor चलता है | Constructor यह class के नाम का होता है | आप एक class के लिए एक से ज्यादा constructor भी define कर सकते है |

For Example,

```
public class Man{
    public Man(){
    }
    public Man(String name, int age){
        // Constructor with parameters.
    }
}
```

## Object क्या होता है ?

Class में हमने देखा कि class यह object के लिए blueprint बनाता है | अब यह स्पष्ट हो जाता है कि class से object को create किया जाता है |

Java में new keyword के साथ object को create किया जाता है |

For Example

```
public class Man {
    String name = "Rakesh";
    int age = 26;
    public static void main(String[] args) {

    Man m = new Man();
        System.out.println("Name is " + m.name);
        System.out.println("Age is " + m.age);
    }
}
```

Output :

```
Name is Rakesh
Age is 26
```



## Create Multiple Objects

आप एक class के लिए एक से ज्यादा objects भी create कर सकते है ।

For Example,

```
public class Man {  
    String name = "Rakesh";  
    int age = 26;  
    public static void main(String[] args) {  
  
    Man m1 = new Man();  
    Man m2 = new Man();  
        System.out.println("Name is " + m1.name);  
        System.out.println("Age is " + m1.age);  
        System.out.println("Name is " + m2.name);  
        System.out.println("Age is " + m2.age);  
    }  
}
```

Output :

```
Name is Rakesh  
Age is 26  
Name is Rakesh  
Age is 26
```

## Accessing Instance Variables and Methods

Object के माध्यम से instance variables और methods को access किया जाता है ।

For Example,

```
public class Man {  
    String name;  
  
    public void setName( String manName ) {  
        name = manName;  
    }  
    public String getName() {  
        System.out.println("Name is :" + name );  
        return name;  
    }  
    public static void main(String[] args) {  
  
        Man m = new Man();  
        //accessing methods by object  
        m.setName("Rakesh");  
        m.getName();  
        //accessing instance variable by object  
        System.out.println("name variable value :" + m.name );  
    }  
}
```

Output :

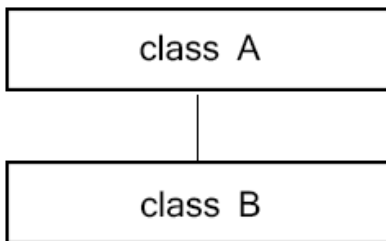
```
Name is :Rakesh  
name variable value :Rakesh
```

# Inheritance

Java में Inheritance ये Object Oriented Programming का एक प्रकार है | जिसमे एक class की properties और methods किसी दुसरे class में inherit की जाती है |

Inheritance में मुख्यतः Parent class और child class का इस्तेमाल किया जाता है | इसमे Parent class को Base class या super class भी कहा जाता है और Child class को Derived class या sub class भी कहा जाता है | C++ ये Inheritance के प्रकार को support करता है, लेकिन Java; Multiple Inheritance को support नहीं करता मतलब Java में Parent class को कई child classes हो सकते है, लेकिन child classes को सिर्फ एक ही Parent class होता है |

## Basic Inheritance



Java में Inheritance के लिए extends keyword का इस्तेमाल किया जाता है |

## Syntax for Inheritance

```
class parent_class{
//statements;
}
class child_class extends parent_class{
//statements;
}
```

## Example for Inheritance

यहाँ example में A ये एक parent class है और B ये child class है | B class; A class की सभी properties; inherit कर सकता है |

```
class A{
//statements;
}
class B extends A{
//statements;
}
```

## Full Example for Inheritance

```
//B.java
class A{
void disp(){
System.out.println("Parent Class");
}
}
class B extends A{
void show(){
System.out.println("Child Class");
}
public static void main(String args[]){
B obj = new B();
obj.disp();
obj.show();
}
}
```

Output :

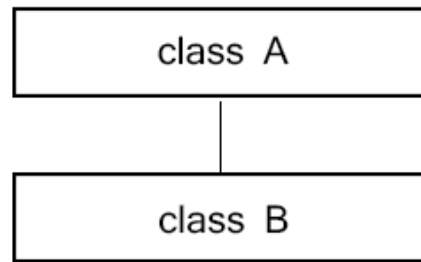
```
Parent Class
Child Class
```

## Types of Inheritance

Java में तीन Inheritance होते हैं ।

1. Single Inheritance
2. MultiLevel Inheritance
3. Hierarchical Inheritance

## 1. Single Inheritance



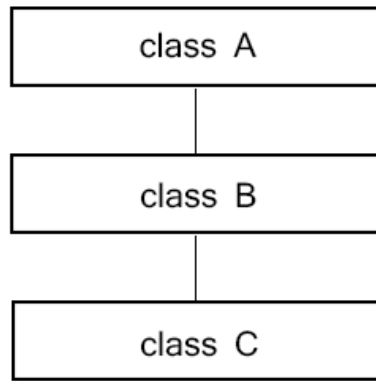
Source Code:

```
//B.java
class A{
void disp(){
System.out.println("Parent Class");
}
}
class B extends A{
void show(){
System.out.println("Child Class");
}
public static void main(String args[]){
B obj = new B();
obj.disp();
obj.show();
}
}
```

Output :

```
Parent Class
Child Class
```

## 2. Multilevel Inheritance



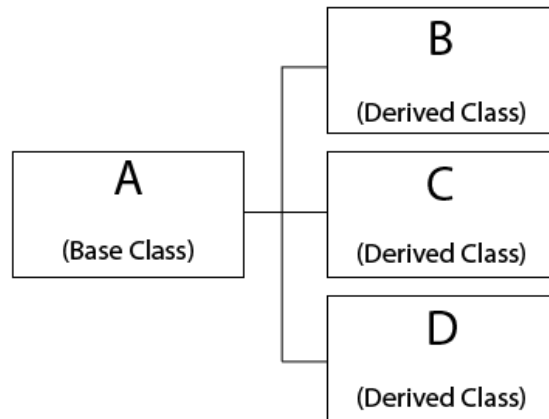
Source Code:

```
//C.java
class A{
void disp(){
System.out.println("Class A");
}
}
class B extends A{
void show(){
System.out.println("Class B");
}
}
class C extends B{
void getdata(){
System.out.println("Class C");
}
public static void main(String args[]){
C obj = new C();
obj.disp();
obj.show();
obj.getdata();
}
}
```

Output :

```
Class A
Class B
Class C
```

### 3. Hierarchical Inheritance



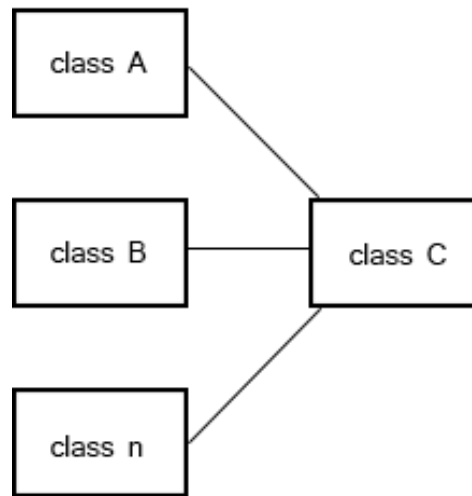
Source Code:

```
//C.java
class A{
void disp(){
System.out.println("Class A");
}
}
class B extends A{
void show(){
System.out.println("Class B");
}
}
class C extends A{
void getdata(){
System.out.println("Class C");
}
}
public static void main(String args[]){
C obj = new C();
obj.disp();
obj.getdata();
B b = new B();
b.show();
}
}
```

Output :

```
Class A
Class C
Class B
```

## Multiple Inheritance



Multiple Inheritance के लिए Java में एक से ज्यादा Base classes होते हैं और सभी base classes को एक ही derived class inherit करता है।

Java में Multiple Inheritance; support नहीं करता है, लेकिन interface के माध्यम से multiple Inheritance Java में support करता है।

### Multiple Inheritance; Java में support क्यों नहीं करता ?

Java में जब Multiple Inheritance आता है, तब Ambiguity problem आ जाता है। Java में extends के साथ कहा पर भी Multiple Inheritance का उल्लेख नहीं किया गया है। लेकिन interface में इसे इस्तेमाल किया जा सकता है। Java में एक से ज्यादा parent class नहीं हो सकते। Program में तीन class हैं। C class; A और B इन दोनों class को inherit करता है।



Source Code:

```
//C.java
class A{
void disp(){
System.out.println("Class A");
}
}
class B{
void show(){
System.out.println("Class B");
}
}
class C extends A, B{
void getdata(){
System.out.println("Class C");
}
}
public static void main(String args[]){
C obj = new C();
obj.disp();
obj.show();
obj.getdata();
}
}
```

Output :

Error.

# Constructor

Constructor ये एक class का special method है जिस class पर उसे बनाया जाता है उसी class के object को initialize करता है।

जिस class का constructor बना हो, अगर उसी class का जब object जब बनता है तब वो automatically call होता है।

Constructor अपने class के नाम जैसा होता है, लेकिन Constructor का कोई return type नहीं होता।

Syntax:

```
class class_name{  
    -----  
    class_name(){      // Constructor  
    //statements;  
    }  
    -----  
}
```

Example for Constructor

```
class A{  
    -----  
    A(){      // Constructor  
    //statements;  
    }  
    -----  
}
```

## Constructor के तीन प्रकार

1. Default Constructor
2. Parameterized Constructor
3. Constructor Overloading

## 1 .Default Constructor

Default Constructor कोई parameter या argument नहीं लेता | ऊपर दिया हुआ program Default Constructor का है |

Program में 'A' नाम का class है और उसका constructor बनाया गया है | जब A class का object बनेगा तब Constructor automatically call होगा |जितनी बार A class का object बनेगा उतनी बार constructor call होता है |

Source Code:

```
class Sample{
int a = 2;
int b = 4;
Sample(){ //Default Constructor
System.out.println("Value of a : " + a);
System.out.println("Value of b : " + b);
}
public static void main(String args[]){
Sample obj = new Sample();
}
}
```

Output:

```
Value of a : 2
Value of b : 4
```

## 2. Parameterized Constructor

Parameterized Constructor में Constructor को parameters pass किये जाते हैं |

Parameterized Constructor में constructor को अलग-अलग arguments दिए जाते हैं | इसमें arguments की कोई मर्यादा नहीं होती |

Parameterized Constructor में class के object में parameters की values देनी पड़ती है |

निचे दिए हुए program Addition के लिए दो values initialize करके उनका addition किया गया है |

Source Code:

```
class A{
    int a, b, c;
    A(int x, int y){    //Parameterized Constructor
        a = x;
        b = y;
        c = a + b;
    }
    void display(){
        System.out.println("Addition of " + a + " and " + b + " is " + c);
    }
}

public static void main(String args[]){
    A a(5, 6);
    a.display();
}
```

Output:

```
Addition of 5 and 8 is 13
```

### 3. Constructor Overloading

Constructor Overloading में class में multiple Constructor overloading की जा सकती है , सिर्फ उनकी parameters की संख्या और उनके type अलग-अलग होते हैं ।

Constructor Overloading; Function Overloading के तरह ही होता है ।

Source Code:

```
class Sample{
int a;
    int b;

Sample(int x){
    a = x;
}
    Sample(int x, int y){
        a = x;
        b = y;
    }
    void show(){
System.out.println("Value of a : " + a);
System.out.println("Value of b : " + b);
}

    public static void main(String args[]){

Sample s1 = new Sample(5);
    Sample s2 = new Sample(5, 9);
    s1.show();
    s2.show();
    }
}
```

Output:

```
Value of a : 5
Value of b : 0
Value of a : 5
Value of b : 9
```

## Copy Constructor

Java में copy constructor का कोई concept नहीं है | लेकिन constructor में एक object को दूसरे object पर copy किया जा सकता है |

Source Code:

```
class Sample{
int a;
int b;
Sample(int x, int y){
a = x;
b = y;
}
Sample(Sample s){    //copy constructor

System.out.println("copy constructor invoked");
a = s.a;
b = s.b;
}
int add(){
int c = a + b;
return c;
}
public static void main(String[] args){
Sample obj1 = new Sample(2, 10);
Sample obj2= new Sample(obj1);
System.out.println("Object1 : "+ obj1.add());
System.out.println("Object2 : "+ obj2.add());
}
}
```

Output:

```
copy constructor invoked
Object1 : 12
Object2 : 12
```

## Constructor और Method में फर्क क्या है ?

Constructor का नाम class के name जैसा ही होता है | Method का नाम कोई भी हो सकता है |

Constructor; object बनते ही call होता है | Method को call करना पड़ता है |

Constructor कोई value return नहीं करता | Method सभी value return करता है |

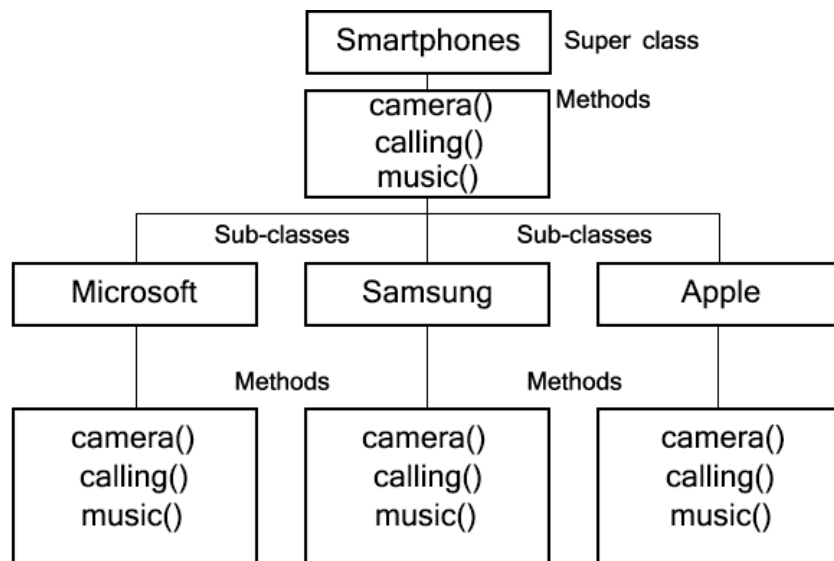
# Polymorphism

## Introduction

Polymorphism ये Object Oriented Programming का बहुत ही अच्छा feature है ।  
एक ही रूप में अनेक रूप होना polymorphism होता है ।  
Polymorphism ये शब्द 'poly' और 'morph' इन दो शब्दों को मिलकर बनाया गया है ।

## Real Life Example for Polymorphism

Real life में smartphones की कई कंपनियां है । जब smartphone इस्तेमाल किया जाता है, तब उस mobile में कई सारे features होते है जैसे Camera, Calling, Music Player, Video Player.  
यहाँ पर mobile ये नाम तो एक ही है लेकिन इसमें कई सारे forms है ।



## Example for Run-Time Polymorphism

Source Code:

```
class Smartphones{
int camera(){
return 0;
}
}
class Microsoft extends Smartphones{
int camera(){
return 8;
}
}
class Samsung extends Smartphones{
int camera(){
return 12;
}
}
class Apple extends Smartphones{
int camera(){
return 12;
}
}
class Sample{
public static void main(String args[]){
Smartphones s = new Microsoft();
System.out.println("Microsoft Camera : " + s.camera() + "MP");
s = new Samsung();
System.out.println("Samsung Camera : " + s.camera() + "MP");
s = new Apple();
System.out.println("Apple Camera : " + s.camera() + "MP");
}
}
```

Output:

```
Microsoft Camera : 8MP
Samsung Camera : 12MP
Apple Camera : 12MP
```



# Upcasting

दिए हुए program में A और B ये दो classes लिए हुए है | B class; A class को inherit कर रहा है और दोनों ही class में disp() नाम का same method है | आखिर में parent class के reference variable से child class के object को refer किया गया है |super class के reference variable से disp() ये method call किया गया है | यहाँ पर compile-time पर उन दो classes के same methods में से कौनसा method call करना ये समझ नहीं आता | इसीलिए JVM द्वारा इसे Run-time पर उस method को call किया जाता है |

Source Code:

```
//B.java
class A{
void disp(){
System.out.println("class A");
}
}
class B extends A{
void disp(){
System.out.println("class B");
}
public static void main(String args[]){
A a = new B(); //upcasting
a.disp();
}
}
```

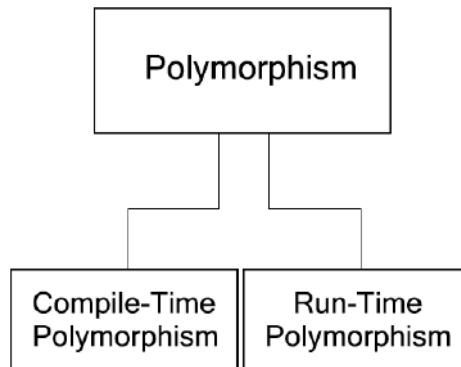
Output:

```
class B
```

# Compile and Run Time Polymorphism

Java में Polymorphism के लिए दो प्रकार हैं ।

1. Compile-Time Polymorphism
2. Run-Time Polymorphism



## 1. Compile-Time Polymorphism

Compile Time Polymorphism को Static Polymorphism भी कहा जाता है । यहाँ पर Method Overloading होता है । जिसमें एक ही प्रकार के methods के नाम और उनके parameters और उन parameters के types अलग-अलग होता है ।

### Method Overloading

Source Code:

```
class Sample{
    public void disp(int x, int y){
        System.out.println("Value of x : " + x);
        System.out.println("Value of y : " + y);
    }
    public void disp(int x){
        System.out.println("Value of x : " + x);
    }
    public static void main(String[] args){
        Sample s = new Sample();
        s.disp(5, 6);
        s.disp(7);
    }
}
```

Output:

```
Value of x : 5
Value of y : 6
Value of x : 7
```

## 2. Run-Time Polymorphism

Run-Time Polymorphism में JVM द्वारा method को run time पर call किया जाता है | Method Overriding ये Run-Time Polymorphism का एक खास उदाहरण है |

Source Code:

```
class A
{
    void disp(){
        System.out.println ("class A");
    }
}
class B extends A
{
    void disp(){
        System.out.println ("Class B");
    }
}
class C{
    public static void main (String args []) {
        A obj1 = new A();
        obj1.disp();

        A obj2 = new B();
        obj2.disp();
    }
}
```

Output:

```
Class A
Class B
```

## Introduction

Program में method के call से method की definition को जोड़ना Binding होता है ।

### Class का Reference Variable

यहाँ पर s ये Sample class का reference variable है ।

```
class Sample{  
    public static void main(String args[]){  
        Sample s;  
        // s is a reference variable of Sample  
    }  
}
```

### Class का Object

यहाँ पर B class का object बनाया गया है । लेकिन A class inherit होने की वजह से ये उसका भी Object है ।

```
class A{  
    -----  
}  
class B extends A{  
    public static void main(String args[]){  
        B b;    // b is a reference variable of class B  
        b = new Sample(); //instance/object of class B also  
                      //instance/object of class A  
    }  
}
```

# Static Binding

Static Binding को Early Binding भी कहा जाता है | जब compile-time पर compiler द्वारा object निश्चित होता है , इसे Static Binding कहते हैं |

जिस class में private, static या final method होता है वहा पर Static Binding होती है |

Compile-Time पर method को override नहीं किया जा सकता | ये methods अपने object से access होते हैं |

Source Code:

```
class A{
    static void disp(){
        System.out.println("class A");
    }
}

class B extends A{
    static void disp(){
        System.out.println("class B");
    }
}

public static void main(String args[]){

    A a = new A();
    B b = new B();
    a.disp();
    b.disp();
}
```

Output:

```
Class A
Class B
```

# Dynamic Binding

Dynamic Binding को Late Binding भी कहा जाता है | ये Binding Compiler द्वारा compile-time पर नहीं होता | यहाँ पर classes के पास same methods होते हैं | यहाँ पर Method Override होता है | यहाँ पर दोनों classes के अन्दर एक ही signature का methods है |

Source Code:

```
class A{
void disp(){
System.out.println("class A");
}
}
class B extends A{
void disp(){
System.out.println("class B");
}
public static void main(String args[]){

A a = new B();
a.disp();
}
}
```

Output:

```
class B
```

# Abstract Class

## Abstract Class and Method

C++ में जब Abstract Class होता था तब कोई abstract keyword का इस्तेमाल नहीं किया जाता था। लेकिन जिस Class के अन्दर pure virtual function होता था तब वो class Abstract Class हो जाती थी।

Java में देखे तो virtual नाम का कोई keyword नहीं होता, लेकिन Abstract Class का इस्तेमाल Java में भी abstract keyword के साथ किया जाता है।

Abstract class को abstract keyword के साथ लिखा जाता है।

### Syntax for Abstract Class

```
abstract class class_name(); //abstract class
```

### Syntax for Abstract Method

```
abstract return_type method_name(); //abstract method
```

Abstract Class का object; create नहीं किया जा सकता।

Source Code:

```
abstract class A{           //Abstract Class
void disp(){
System.out.println("class A");
}
public static void main(String args[]){
A a = new A(); //Object for Abstract Class
a.disp();
}
}
```

Output:

```
A.java:8: error: A is abstract; cannot be instantiated
```

abstract class का कोई object नहीं होता, लेकिन जो sub-class इसे inherit करता है, उससे abstract class के methods को access किया जा सकता है।

जब abstract class; create होता है तब उसे किसी abstract method की जरूरत नहीं होती है।

Source Code:

```
abstract class A{          //Abstract Class
void disp1(){
System.out.println("class A");
}
}
class B extends A{
void disp2(){
System.out.println("class B");
}
public static void main(String args[]){
B b = new B();
b.disp1();
b.disp2();
}
}
```

Output:

```
class A
class B
```

Abstract Class का reference variable बनाया जा सकता है लेकिन उसका object नहीं बनाया जा सकता।

Source Code:

```
abstract class A{          //Abstract Class
void disp1(){
System.out.println("class A");
}
}
class B extends A{
void disp2(){
System.out.println("class B");
}
public static void main(String args[]){
A a = new B();
a.disp1();
}
}
```



Output:

```
class A
```

Abstract method के बारे में देखे तो हर abstract method का class; Abstract class ही होता है | वहा पर उस abstract method को सिर्फ declare करना पड़ता है |

```
abstract class A{  
    abstract void disp();  
}
```

Source Code:

```
abstract class A{           //Abstract Class  
    abstract void disp1(); //Abstract Method  
}  
class B extends A{  
    void disp1(){  
        System.out.println("class B");  
    }  
    public static void main(String args[]){  
        A a = new B();  
        a.disp1();  
    }  
}
```

Output:

```
class B
```

## Abstract Classes using Multilevel Inheritance

जब abstract super class के method की definition दी जाती तो super class का sub class भी abstract class हो जाता है |

**For Example,**

Program पर दिखाया गया है, कि class A ये एक abstract super class है और वहा पर disp1() नाम का abstract method भी लिया है और उसके class B(sub class) कोई disp1() नाम का implementation नहीं किया है , इसीलिए वो भी abstract class हो जाता है और बाद में class C पर disp1() इस abstract method की definition दी गयी है और बाद में class C का object बनाकर disp1() और disp2() को access किया गया है |

#### Source Code:

```
abstract class A{          //Abstract Class
abstract void disp1(); //Abstract Method
}
abstract class B extends A{ //Abstract Class
void disp2(){
System.out.println("class B");
}
}
class C extends B{
void disp1(){
System.out.println("class A");
}
public static void main(String args[]){
C c = new C();
c.disp1();
c.disp2();
}
}
```

#### Output:

```
class A
class B
```

Sub-class में भी abstract method हो सकती है ।

```
abstract class A{          //Abstract Class
void disp1(){
System.out.println("class A");
}
}
abstract class B extends A{ //Abstract Class
abstract void disp2(); //Abstract Method
}
class C extends B{
void disp2(){
System.out.println("class B");
}
public static void main(String args[]){
C c = new C();
c.disp1();
c.disp2();
}
}
```

Output:

```
class A
class B
```

## Some Example of Abstract classes using Multilevel

एक abstract class में कितने भी abstract methods दिए जा सकते हैं। ये जरूरी नहीं है कि, हर abstract class के अन्दर abstract method हो।

Program पर class A इस abstract parent class में दो abstract methods दिए गए हैं। लेकिन उन दोनों abstract methods की definition उसे inherit किये गए sub-class पर की जाती है। उन दोनों abstract class का implementation करना जरूरी है।

Program पर Source Code ये है कि class A पर दो abstract methods दिए गए हैं और class B(sub-class) पर उसके सिर्फ एक ही abstract method का implementation किया गया है। इससे वो भी class abstract class हो जाती है।

Source Code:

```
abstract class A{          //Abstract Class
abstract void disp1();
abstract void disp2();
}
abstract class B extends A{  //Abstract Class
void disp2(){
System.out.println("class B");
}
}
abstract class C extends B{  //Abstract Class
void disp3(){
System.out.println("class C"); //
}
}
class D extends C{
void disp1(){
System.out.println("class D"); //
}
public static void main(String args[]){
D d = new D();
d.disp1();
d.disp2();
d.disp3();
}
}
```

Output:

```
class D  
class B  
class C
```

# Method Overloading and Overriding

## Method Overloading

Method Overloading में एक class पर एक ही नाम के multiple method हो सकते हैं। लेकिन उन methods की arguments की संख्या और उनका type अलग-अलग होता है।

Program पर Constructor Overloading भी किया जाता है।

Method Overloading का return-type से कोई लेना-देना नहीं होता।

### For example,

अगर program में एक class के अन्दर same signature के दो methods होते हैं तो Ambiguity Error आ जाता है, चाहे उनका return-type अलग हो या ना हो। इसका मतलब ये है कि, return-type से Method Overloading का कोई सम्बन्ध नहीं है।

### Source Code:

```
class Sample{
int disp(int x){
return x;
}
double disp(int y){
return y;
}
public static void main(String args[]){
Sample s = new Sample();
System.out.println("Value of x : " + s.disp(5));
System.out.println("Value of y : " + s.disp(6.5));
}
}
```

### Output:

```
Sample.java:6: error: method disp(int) is already defined in class Sample
```

**Method को दो प्रकार से overload किया जाता है ।**

- 1.Changing types of Arguments/Parameters
- 2.Changing Number of Arguments/Parameters

## **1. Changing types of Arguments/Parameters**

Program में तीन same name के methods लिए है । लेकिन उनके argument का type अलग-अलग लिया है ।  
return-type इसीलिए change नहीं किया चूँकि lossy conversion का error ना आ जाए ।

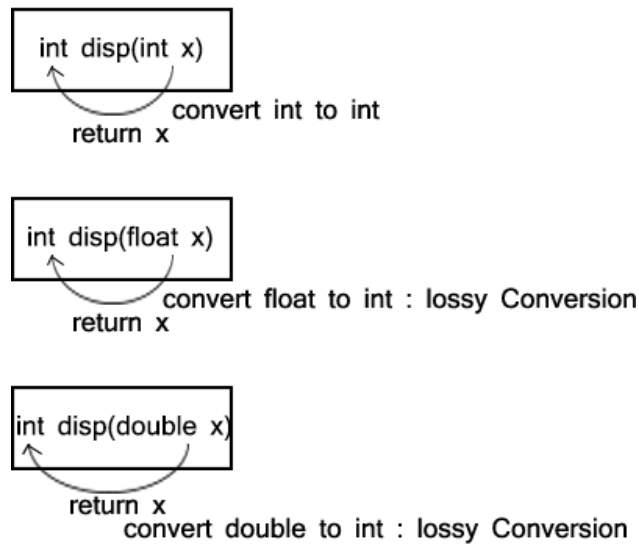
Source Code:

```
class Sample{
int disp(int x){
return x;
}
float disp(float x){
return x;
}
double disp(double x){
return x;
}
public static void main(String args[]){
Sample s = new Sample();
System.out.println("Value of x : " + s.disp(5));
System.out.println("Value of x : " + s.disp(6.5));
System.out.println("Value of x : " + s.disp(7.54));
}
}
```

Output:

```
Value of x : 5
Value of x : 6.5
Value of x : 7.54
```

## Lossy Conversion Occur using Same return-type



Program में तीन same methods लिए है लेकिन उनके return-type एक जैसे है | अगर method पर int type का argument pass किया गया है और उसको int type में ही return किया गया है, इससे कोई lossy conversion की error नहीं आती चूँकि int to int conversion हो रहा है |

Method पर float type का argument लिया जाता है और उसे int type में return किया जाता है, तो यहाँ पर lossy conversion की संभावना होती है |

Method पर double type को int type में convert नहीं किया जा सकता |

Source Code:

```
class Sample{
int disp(int x){
return x;
}
int disp(float x){
return x;
}
int disp(double x){
return x;
}
public static void main(String args[]){
Sample s = new Sample();
System.out.println("Value of x : " + s.disp(5));
System.out.println("Value of x : " + s.disp(6.5));
System.out.println("Value of x : " + s.disp(7.54));
}
}
```

Output:

```
Sample.java:7: error: incompatible types: possible lossy conversion from float to int
```

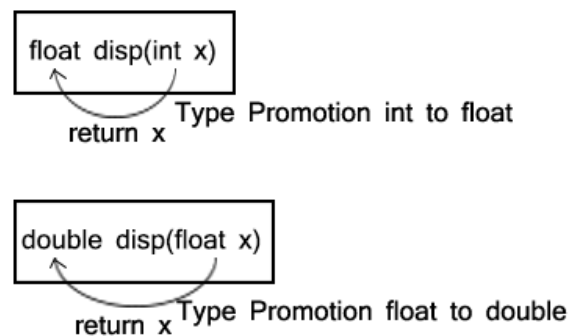
```
    return x;
```

```
    ^
```

```
Sample.java:10: error: incompatible types: possible lossy conversion from double to int
```

```
    return x;
```

## Type Conversion



Program पर data type का conversion किया गया है ।

पहले disp() method पर int to float में convert किया गया है और दूसरे disp() पर float to double convert किया गया है ।

Source Code:

```
class Sample{
float disp(int x){
return x;
}
double disp(float x){
return x;
}
public static void main(String args[]){
Sample s = new Sample();
System.out.println("Value of x : " + s.disp(5));
System.out.println("Value of x : " + s.disp(6));
}
}
```

Output:

```
Value of x : 5.0
```

```
Value of x : 6.0
```



## 2. Changing Number of Arguments/Parameters

Arguments/Parameters की संख्या बदलकर भी एक class के अन्दर same methods का इस्तेमाल किया जा सकता है।

Source Code:

```
class Sample{
    int disp(int x){
        return x;
    }
    int disp(int x, int y){
        return x + y;
    }
    int disp(int x, int y, int z){
        return x + y + z;
    }
    public static void main(String args[]){
        Sample s = new Sample();
        System.out.println("Value of x: " + s.disp(5));
        System.out.println("Addition of x and y : " + s.disp(5, 9));
        System.out.println("Addition of x, y and z : " + s.disp(5, 9, 5));
    }
}
```

Output:

```
Value of x: 5
Addition of x and y : 14
Addition of x, y and z : 19
```

## Example for Valid Sequence of Arguments types

```
disp(char, int)
disp(int, char)
```

Source Code:

```
class Sample1
{
    void disp(char c, int x)
    {
        System.out.println("Value of c : " + c);
        System.out.println("Value of x : " + x);
    }
    void disp(int x, char c)
    {
        System.out.println("Value of x : " + x);
        System.out.println("Value of c : " + c);
    }
    public static void main(String args[])
    {
        Sample1 s = new Sample1();
        s.disp('c', 12);
        s.disp(13, 'h');
    }
}
```

Output:

```
Value of c : c
Value of x : 12
Value of x : 13
Value of c : h
```

## Example for invalid Sequence of Arguments types

```
disp(float, int)
disp(int, float)
```

Source Code:

```
class Sample
{
    void disp(float x, int y)
    {
        System.out.println("Value of x : " + x);
        System.out.println("Value of y : " + y);
    }
    void disp(int x, float y)
    {
        System.out.println("Value of x : " + x);
        System.out.println("Value of y : " + y);
    }
    public static void main(String args[])
    {
        Sample s = new Sample();
        s.disp(10, 14);
        s.disp(2, 3);
    }
}
```

Output:

both method disp(float,int) in Sample and method disp(int,float) in Sample match

Sample.java:18: error: reference to disp is ambiguous

```
    obj.disp(2, 3);
```

^

both method disp(float,int) in Sample and method disp(int,float) in Sample match

# Method Overriding

Method Overriding में एक program पर same name के methods होते हैं। इन methods की signature एक जैसी ही होती है। लेकिन ये अलग-अलग class में स्थित होते हैं।

**For example,**

अगर parent class का method और उसके sub-class का method का नाम और signature एक जैसा हो तो उसे Method Overriding कहते हैं। Method Overriding ये RunTime Polymorphism का एक अच्छा उदाहरण है।

## Example for Method Overriding

Program में देखें तो दो class लिए हुए हैं। एक class A जो Parent class है और दूसरा class B जो उसका sub-class है और दोनों class में same name के और same signature के methods लिए हुए हैं। class B की method; class A की method को override कर रही है।

Source Code:

```
class A
{
    void disp(){
        System.out.println("class A");
    }
}

class B extends A{
    void disp(){
        System.out.println("class B");
    }

    public static void main(String args[]){

        B b = new B();
        b.disp();
    }
}
```

Output:

```
class B
```

# Overloading और Overriding में फर्क

Method Overloading	Method Overriding
<div>class A</div> <div>void disp()</div> <div>void disp(int x)</div> <div>void disp(int x, int y)</div>	<div>class A</div> <div>void disp()</div> <div>class B extend A</div> <div>void disp()</div>

Method Overloading	Method Overriding
ये Compile-Time Polymorphism का एक अच्छा उदाहरण है ।	ये Run-Time Polymorphism का एक अच्छा उदाहरण है ।
Overloading में parameter की संख्या और type अलग-अलग होते है ।	Overriding में methods के नाम और signature एक जैसे होते है ।
Overloading एक ही class पर होता है ।	Overriding अलग-अलग class पर होता है ।
Overloading में return-type अलग-अलग हो सकता है ।	Overriding में return-type; same होता है ।
Overloading में static method और non-static method दोनों होता है ।	Overriding में non-static methods होते है ।

# Access Modifiers

## What is Access Modifier and its Types

Access Modifiers; Classes, Data members, Methods और Constructor के accessibility के लिए इस्तेमाल किया जाता है ।

Access Modifiers चार प्रकार के होते हैं ।

- 1.default Access Modifier
- 2.private Access Modifier
- 3.public Access Modifier
- 4.protected Access Modifier

### 1. default Access Modifier

अगर कोई Modifier नहीं दिया जाता तो, by default default दिया जाता है ।

default Modifier सिर्फ package के अन्दर ही accessible होता है ।

Program में देखे तो दो packages बनाये गए हैं । class A को किसी package से access किया जाता है । लेकिन default Modifier होने के कारण ये access नहीं होता ।

Source Code:

```
//A.java
package pack1;
class A{
void disp(){
System.out.println("class A");
}
}
```

Source Code:

```
//B.java
package pack2;
import pack1.*;
class B{
public static void main(String args[]){
A a = new A();
a.disp();
}
}
```

Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . A.java
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . B.java
B.java:6: error: A is not public in pack1; cannot be accessed from outside package
    A a = new A();
    ^
B.java:6: error: A is not public in pack1; cannot be accessed from outside package
    A a = new A();
    ^
2 errors
```

## 2. private Access Modifier

private Access Modifier ये सिर्फ class के अन्दर accessible होते है |

Program में देखे तो parent class में a और b नाम के दो instance variables लिए है | लेकिन a variable; private है | ये variable सिर्फ अपने class के अन्दर ही accessible होते है | Program में b variable दुसरे Child class से access हो रहा है, लेकिन a variable access नहीं हो रहा |

Source Code:

```
class Parent{
    private int a = 5;
    int b = 10;
}
class Child extends Parent{
    void disp(){
        System.out.println("Value of a : " + a);
        System.out.println("Value of b : " + b);
    }
    public static void main(String args[]){
        Child c = new Child();
        c.disp();
    }
}
```

Output:

```
Child.java:9: error: a has private access in Parent
    System.out.println("Value of a : " + a);
```

## 3. public Access Modifier

public Access Modifier से कहा पर भी access किया जा सकता है |

Source Code:

```
//A.java
package pack1;
public class A{
    public void disp(){
        System.out.println("class A");
    }
}
```

Source Code:

```
//B.java
package pack2;
import pack1.*;
class B{
    public static void main(String args[]){
        A a = new A();
        a.disp();
    }
}
```

Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . A.java
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . B.java
C:\Program Files\Java\jdk1.8.0_111\bin>java pack2.B
class A
```

## 4. protected Access Modifier

ये सिर्फ अपने package के अन्दर और अपने sub-class पर accessible होते हैं ।

Source Code:

```
class Parent{
    int a = 5;
    int b = 10;
    protected void disp(){
        System.out.println("Value of a : " + a);
        System.out.println("Value of b : " + b);
    }
}

class Child extends Parent{
    public static void main(String args[]){
        Child c = new Child();
        c.disp();
    }
}
```

Output:

```
Value of a : 5
Value of b : 10
```



## Accessing protected method Outside Package

public Access Modifier से कहा पर भी access किया जा सकता है ।

Source Code:

```
//A.java
package pack1;

public class A{
protected void disp(){
System.out.println("class A");
}
}
```

Source Code:

```
//B.java
package pack2;
import pack1.*;
class B extends A{
public static void main(String args[]){
B b = new B();
b.disp();
}
}
```

Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . A.java
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . B.java
C:\Program Files\Java\jdk1.8.0_111\bin>java pack2.B
class A
```

# Data Encapsulation

Data Encapsulation में data members और methods एक ही unit पर या class पर wrap करके रखा जाता है ।

Data Encapsulation class के attributes और methods को combine करके रखता है ।

Data Encapsulation को Data Hiding भी कहा जाता है, चूँकि इसमें जो instance variables होते हैं, वो private होते हैं । इसका मतलब ये है , अगर इसे outside से access किये जाए तो ये accessible नहीं होते ।

Normally Encapsulation में जो methods इस्तेमाल की जाती हैं उनका उल्लेख getter और setter होता है ।

Encapsulation में data members और methods की accessibility(Access Modifiers) भी काफी प्रभावित करती है ।

Program पर देखे तो ,

तीन private data members किये गए हैं । ये members सिर्फ अपने class के लिए ही accessible होंगे, ये outside से या किसी दूसरी class से access नहीं किये जा सकते । बाद में तीन public getter और तीन setter लिए गए हैं । ये public होने के कारण outside से भी इन methods को access किया जा सकता है । जो private data members वहाँ पर दिए गए हैं, वो public methods के माध्यम से ही accessible रहेंगे ।

Source Code:

```
//Employee.java
class Employee
{
    private int id;
    private String name;
    private double salary;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
public static void main(String args[])
{
    Employee e = new Employee();

    e.setId(99);
    e.setName("Rakesh");
    e.setSalary(45000.58);
    System.out.println("Employee Id : "+e.getId());
    System.out.println("Employee Name : "+e.getName());
    System.out.println("Employee Salary : "+e.getSalary());
}
}
```

Output:

```
Employee Id : 99
Employee Name : Rakesh
Employee Salary : 45000.58
```

## Data Encapsulation के फायदे

Encapsulation का code flexible होता है चूँकि Requirement के हिसाब से code को change किया जा सकता है ।

Encapsulation में data की accessibility को control किया जाता है ।

Encapsulation में code की maintainability और reusability बढ़ाई जाती है ।

# Interfaces

## Introduction

interface ये class के जैसे होते हैं। interface जो methods होते हैं, वो abstract होते हैं और उनका scope public होता है। interface का कोई object create नहीं किया जा सकता, लेकिन reference variable create किया जा सकता है। interface में जो methods होते हैं, उनका सिर्फ declaration किया जाता है।

interface में जब instance variable को initialize किया जाता है, तब उसके साथ by default compiler द्वारा public static final ये keywords आ जाते हैं।

interface को implement करने के लिए class के जरिये implements keyword का इस्तेमाल किया जाता है।

interface में constructor नहीं होते।

Syntax:

```
interface interface_name{  
    //some code;  
}
```

Example

```
interface Sample{  
    int a = 5;  
    void disp();  
}
```

Sample.java

```
interface Sample{  
    int a = 5;void disp();  
}
```

Compiler

Sample.class

```
interface Sample{  
    public static final int a = 5;  
    public abstract void disp();  
}
```

## Full Example for interface

जब program में interface का इस्तेमाल किया जाता है, तब abstract method का नियम लागू होता है ।

### For example,

Program में देखे तो interface A में disp() नाम का method आया है । ये by default public और abstract keywords के साथ होता है । जब interface पर कोई method declare की जाती है, तब उसके sub-class पर उसकी body देना जरूरी होती है ।

```
//B.java
interface A{
void disp();
}
class B implements A{
public void disp(){
System.out.println("interface A");
}
public static void main(String args[]){
B b = new B();
b.disp();
}
}
```

Output:

```
interface A
```

## Reference Variable for interface

interface का reference variable create किया जा सकता है , लेकिन object; create नहीं किया जा सकता ।

```
//B.java
interface A
{
    void disp();
}
class B implements A
{
    public void disp()
    {
        System.out.println("interface A");
    }
    public static void main(String args[])
    {
        A obj = new B(); //obj is reference variable of interface A
        obj.disp();
    }
}
```

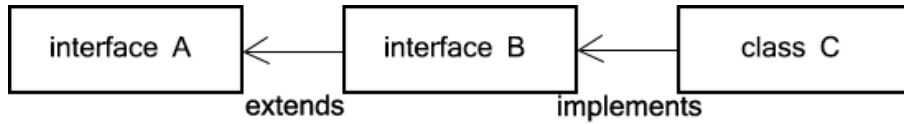
Output:

```
interface A
```

# Relation between Class and Interface

interface और interface में **extends** keyword का इस्तेमाल किया जाता है ।

interface और class में **implements** keyword का इस्तेमाल किया जाता है ।



Source Code:

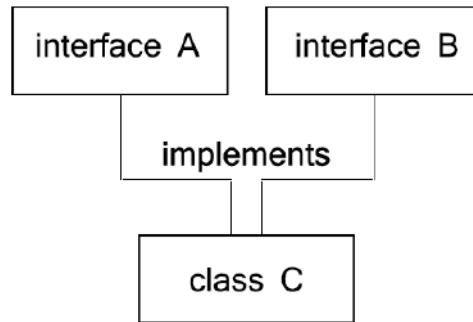
```
//C.java
interface A
{
    void disp1();
}
interface B extends A{
    void disp2();
}
class C implements B{
    public void disp1(){
        System.out.println("interface A");
    }
    public void disp2(){
        System.out.println("interface B");
    }
    public static void main(String args[])
    {
        C obj = new C();
        obj.disp1();
        obj.disp2();
    }
}
```

Output:

```
interface A
interface B
```

## Multiple Inheritance using interface and class

यहाँ पर interface और class का multiple inheritance लिया है ।



Source Code:

```
//C.java
interface A{
    void disp1();
}
interface B{
    void disp2();
}
class C implements B, A{
    public void disp1(){
        System.out.println("interface A");
    }
    public void disp2(){
        System.out.println("interface B");
    }
    public static void main(String args[])
    {
        C obj = new C();
        obj.disp1();
        obj.disp2();
    }
}
```

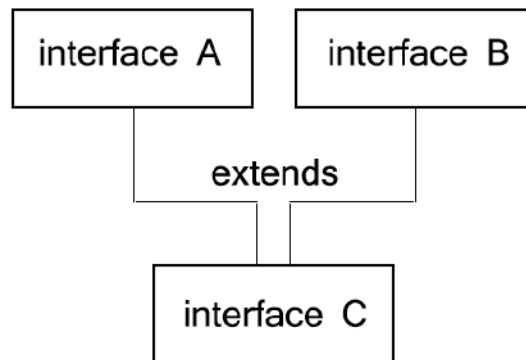
Output:

```
interface A
interface B
```



## Multiple Inheritance using Interface

यहाँ पर interface और interface का multiple inheritance लिया है ।



Source Code:

```
//D.java
interface A{
    void disp1();
}
interface B{
    void disp2();
}
interface C extends B, A{
    void disp3();
}
class D{
    public void disp1(){
        System.out.println("interface A");
    }
    public void disp2(){
        System.out.println("interface B");
    }
    public void disp3(){
        System.out.println("interface C");
    }
    public static void main(String args[])
    {
        D obj = new D();
        obj.disp1();
        obj.disp2();
        obj.disp3();
    }
}
```

Output:

```
interface A  
interface B  
interface C
```

## Rules of Interfaces

- Interface का object create नहीं किया जा सकता, लेकिन reference variable create किया जा सकता है ।
- Interface का constructor नहीं होता ।
- Interface और Interface को extend किया जाता है ।
- Interface और class को implement किया जाता है ।
- Interface का हर reference variable public, static और final keyword के साथ होता है ।
- Interface में हर instance variable; final keyword के साथ होने के कारण constant होता है ।
- Interface के methods public और abstract होते हैं ।

# Enumeration

Enumeration के लिए 'enum' keyword का इस्तेमाल किया जाता है | Enumeration ये constants की list होती है | Enumeration में instance variable, methods और constructor होते हैं | Enumeration; class और interface के जैसा होता है | हर enumeration के constant के साथ static और final keyword होता है |

## Defining enum

यहाँ पर Planet ये enumeration का type है और EARTH, MOON, JUPITER और SATURN उसके constants हैं | Constants को comma operators से separate किया जाता है |

**Enum Type** : Planet.

**Planet Enum Constants** : EARTH, MOON, JUPITER, SATURN.

Syntax:

```
public enum Planet{  
    EARTH,  
    MOON,  
    JUPITER,  
    SATURN;  
}
```

## Syntax for Assigning Value

```
enum_type enum_variable = enum_type.Constant;
```

## Example for Basic Enum

```
//Planet.java  
public enum Planet{  
    EARTH,  
    MOON,  
    JUPITER,  
    SATURN;  
}
```

```
//Sample.java
public class Sample{
public static void main(String args[]){
System.out.println(Planet.EARTH);
}
}
```

Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac Planet.java
C:\Program Files\Java\jdk1.8.0_111\bin>javac Sample.java
C:\Program Files\Java\jdk1.8.0_111\bin>java Sample
EARTH
```

### Example for Enum Iteration

यहाँ पर Enum Iteration के लिए foreach loop का इस्तेमाल किया है और values() method का इस्तेमाल किया गया है | values() ये method; enum constants का array; return करता है |

```
//Planet.java
public enum Planet{
    EARTH,
    MOON,
    JUPITER,
    SATURN;
}
```

```
//Sample.java
public class Sample{
public static void main(String args[]){
for (Planet p : Planet.values()) {
    System.out.println(p);
}
}
}
```

Output:

```
EARTH
MOON
JUPITER
SATURN
```

## Example for Enumeration with Instance Variable, Constructor and Method

यहाँ पर Enum Iteration के लिए foreach loop का इस्तेमाल किया है और values() memthod का इस्तेमाल किया गया है | values() ये method; enum constants का array; return करता है |

```
//Planet.java
public enum Planet{
    EARTH("Also known as World"),
    MOON("Moon of Earth"),
    JUPITER("Largest planet in the Solar System"),
    SATURN("second-largest in the Solar System");
    private String desc;
    Planet(String desc) {
        this.desc = desc;
    }
    public String desc() {
        return desc;
    }
}
```

```
//Sample.java
public class Sample{
    public static void main(String args[]){
        System.out.println(Planet.EARTH.desc());
    }
}
```

Output:

```
Also known as World
```

## Use Switch case with Enum

```
//Planet.java
public enum Planet{
    EARTH,
    MOON,
    JUPITER,
    SATURN;
}
```

```
//Sample.java
public class Sample{
    public static void main(String args[]){
        Planet p = Planet.MOON;
        switch( p ){
case EARTH:
    System.out.println("Earth : Also known as World");
    break;

case MOON:
    System.out.println("Moon : Moon of Earth");
    break;

case JUPITER:
    System.out.println("Jupiter : Largest planet in the Solar System");
    break;
case SATURN:
    System.out.println("Saturn : second-largest in the Solar System");
    break;
default:
    System.out.println("Other Planet Found");
    }
    }
}
```

Output:

```
Moon : Moon of Earth
```

# Multithreading

## Multithreading and Life Cycle of Thread

एक साथ अनेक threads या sub-programs को run किया जाना Multithreading होता है।

Java में Multithreading का program एक साथ एक से ज्यादा threads को run करता है। ये thread बहुत ही light-weight की process होती है।

अगर Multithreading का उदाहरण ले तो जब MS Word में कोई data लिखते वक्त spell checks होता है, ये process MS Word के background पर होता है।

### Thread क्या है ?

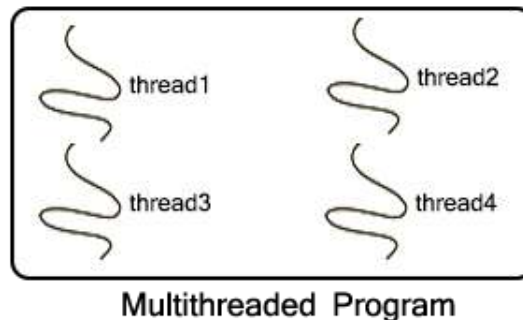
thread ये एक process का sub-process है।

thread light-weight होते हैं।

program में एक या एक से ज्यादा threads create किये जा सकते हैं।

जब process पर कोई thread create नहीं किया जाता तो main thread create होता है।

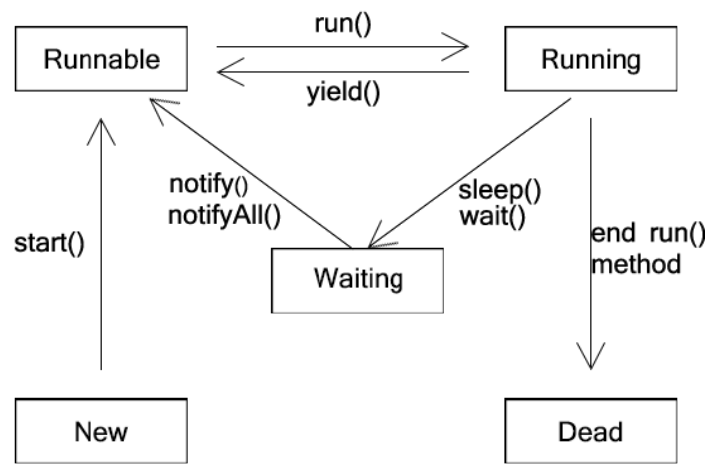
thread; process की common memory area को share करता है। thread अलग से memory को allocate नहीं करता।



### Life Cycle for Thread

threads का एक Life Cycle है। threads अलग-अलग पड़ावों से गुजरता है।

- New State
- Runnable State
- Running State
- Waiting State
- Dead State



**New State** : यहाँ से thread की शुरुआत होती है | यहाँ पर thread का object create किया जाता है |

**Ready/Runnable State** : जब `start()` method को call किया जाता है, तब thread New से Runnable State पर आ जाता है | ये execute होने के लिए Ready होता है |

**Running State** : जब thread execution होना शुरू होता है, तब thread इस state पर होता है |

**Waiting State** : दूसरे thread को perform करने के लिए कुछ thread को block या waiting state पर होते हैं, waiting state पर जो thread होता है उसे resume किया जाता है |

**Dead State** : यहाँ पर thread का काम पूरा होकर वो बंद हो जाता है |

Source Code:

```

public class CheckState extends Thread{
    public void run() {
        System.out.println("run method");
    }
    public static void main(String args[]){

        CheckState t1 = new CheckState();
        CheckState t2= new CheckState();
        System.out.println("t1 State : " + t1.getState());
        System.out.println("t2 State : " + t2.getState());
        t1.start();
        System.out.println("t1 State : " + t1.getState());
        System.out.println("t2 State : " + t2.getState());
        t2.start();
        System.out.println("t1 State : " + t1.getState());
        System.out.println("t2 State : " + t2.getState());
    }
}
  
```



Output:

```
t1 State : NEW
t2 State : NEW
t1 State : RUNNABLE
run method
t2 State : NEW
t1 State : TERMINATED
run method
t2 State : RUNNABLE
```

## Thread Priority

Java में हर एक thread के लिए JVM द्वारा default priority set की जाती है | जब thread create किया जाता है तब programmer द्वारा भी priority set की जाती है | इन priority को set करने के लिए Java में `getPriority()` और `setPriority` ये दो methods है |

Priorities 1 से लेकर 10 तक होती है |

### Constants for Thread Priorities

**public static int MIN\_PRIORITY** : ये thread के लिए minimum priority है | इसके value 1 होती है |

**public static int NORM\_PRIORITY** : ये thread की default priority है | जब thread के लिए कोई priority दी नहीं जाती तब ये priority set की जाती है | इसकी value 5 होती है |

**public static int MAX\_PRIORITY** : ये maximum priority है | इसकी value 10 होती है |

### Example for Default Thread Priority

Source Code:

```
//CheckPriority.java
class CheckPriority extends Thread{
    public void run(){
        System.out.println(Thread.currentThread().getPriority());
    }
    public static void main(String args[]){
        CheckPriority t1 = new CheckPriority();
        CheckPriority t2 = new CheckPriority();
        CheckPriority t3 = new CheckPriority();
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output:

```
5
5
5
```

## Example for set Priority

एक समय पर सभी threads execution के लिए तैयार रहते हैं , लेकिन priority जब set की जाती है तब जिस thread की priority maximum होती है वो thread execution के लिए पहले तैयार होता है ।

Source Code:

```
//CheckPriority.java
class CheckPriority extends Thread{
    public void run(){
        String threadName = Thread.currentThread().getName();
        int threadPriority = Thread.currentThread().getPriority();
        System.out.println("Thread Name is " + threadName + " and Thread priority is " +
            threadPriority );
    }
    public static void main(String args[]){
        CheckPriority t0 = new CheckPriority();
        CheckPriority t1 = new CheckPriority();
        CheckPriority t2 = new CheckPriority();
        t0.setPriority(Thread.MIN_PRIORITY);
        t1.setPriority(Thread.NORM_PRIORITY);
        t2.setPriority(Thread.MAX_PRIORITY);

        t0.start();
        t1.start();
        t2.start();
    }
}
```

Output:

```
Thread Name is Thread-2 and Thread priority is 10
Thread Name is Thread-1 and Thread priority is 5
Thread Name is Thread-0 and Thread priority is 1
```

# Class Thread and Methods

## Introduction for Thread Class

Thread ये class Java में Multithreading के लिए काम करता है | Thread Class में thread priority के लिए कुछ Constants, Constructors और Methods; Multithreading के लिए काम करते हैं |

## Thread Priorities

MIN-PRIORITY(1)

NORM-PRIORITY(5)

MAX-PRIORITY(10)

## Thread Class Constructors

Thread()

Thread(Runnable r)

Thread(String s)

Thread(Runnable r, String s)

## Thread Class के Methods

Thread Methods	Description
currentThread()	इस method से thread object का reference return किया जाता है
getName()	इस method से execute हो रहे thread का नाम return किया जाता है
getPriority()	current thread की priority को return किया जाता है
getState()	Thread का state return किया जाता है
isAlive()	Thread क्रियाशील है या नहीं ये check करने के लिए इस्तेमाल किया जाता है   अगर alive है तो true या निष्क्रिय हो तो false return करता है
isDeamon()	अगर thread daemon है तो true return करता है अन्यथा false return होता है
join()	किसी current thread को hold पर रखा जाता है   जब तक join method से specified किया हुआ thread terminate नहीं हो जाता तब तक current thread hold पर रहता है
resume() (deprecated)	Thread को Waiting से Runnable state पर लेकर आता है
run()	यहाँ thread के execution के लिए run method को override किया जाता है   जो already Thread class पर होता है

setName()	इस method से thread का नाम set किया जाता है ।
setPriority()	Thread की priority को set किया जाता है ।
sleep()	Thread specified किये हुए milliseconds तक sleep होता है ।
start()	इससे thread के execution की शुरुआत होती है और run method को call किया जाता है ।
stop() (deprecated)	यहाँ पर thread को terminate किया जाता है ।
suspend() (deprecated)	Thread को running state से waiting state पर ले आता है ।
yield()	current execute हो रहे thread को थोड़ी देर थमाकर दूसरे thread को चलाने की अनुमति देता है ।

## Creating Thread

Thread को दो प्रकार से create किया जाता है ।

1. extending Thread class
2. implementing Runnable interface

### 1. extending Thread class

**Step 1 :** Thread create करने के लिए class को create करना पड़ता है और Thread class को extend करना पड़ता है ।

**Step 2 :** उसके बाद public void run() इस method को override करना पड़ता है । इसमें thread के execution के लिए statements लिखे जाते हैं । जब object के साथ start() method को call किया जाता है, तब run method execute होता है ।

**Step 3 :** main method में class का object create किया जाता है और इसके बाद object के साथ start() method को call किया जाता है । Method call होने पर thread का execution start हो जाता है । start() method thread को New state से Runnable state पर आता है ।

Source Code:

```
//CreateThread.java
class CreateThread extends Thread{
    public void run(){
        System.out.println("Thread is run.");
    }
    public static void main(String args[]){
        CreateThread t1 = new CreateThread();
        t1.start();
    }
}
```

Output:

```
Thread is run.
```

## 2. implementing Runnable interface

**Step 1 :** Thread create करने के लिए class को create करना पड़ता है और Runnable interface को implement करना पड़ता है ।

**Step 2 :** उसके बाद public void run() इस method को override करना पड़ता है । इसमें thread के execution के लिए statements लिखे जाते हैं । जब thread के object के साथ start() method को call किया जाता है, तब run method execute होता है ।

**Step 3 :** main method में class का object create किया जाता है और इसके साथ Thread का object भी create किया जाता है । Thread object को create करने का कारण यह है कि, Runnable interface में run() ये एकमात्र method होती है और Thread class में वो सभी methods होती जो Thread को create करने में काम आती हैं ।

**Step 4 :** जब Thread का object create किया जाता है तब उस object में run method के class के object को argument के रूप में pass किया जाता है । इससे class के run method को execute किया जाता है ।

**Step 5 :** और आखिर में Thread class के object के साथ start method को call किया जाता है ।

Source Code:

```
//CreateThread.java
class CreateThread implements Runnable{
    public void run(){
        System.out.println("Thread is run.");
    }
    public static void main(String args[]){
        CreateThread c = new CreateThread();
        Thread t1 = new Thread(c);
        t1.start();
    }
}
```

Output:

```
Thread is run.
```

## Naming Thread

thread का नाम जानने के लिए और thread नाम देने के लिए `getName()` और `setName()` का इस्तेमाल किया जाता है ।

### Syntax for `getName()` and `setName()` for Thread

Thread का नाम जानने के लिए `getName()` का इस्तेमाल किया जाता है ।

```
public String getName()
```

Thread को कोई दूसरा नाम देने के लिए `setName` का इस्तेमाल किया जाता है ।

```
public void setName(String name)
```

### Example for Get Thread Name

Source Code:

```
public class CheckThreadName extends Thread{
    public void run(){
        System.out.println("Thread is run.");
    }
    public static void main(String args[]){
        CheckThreadName t1 = new CheckThreadName();
        CheckThreadName t2 = new CheckThreadName();
        CheckThreadName t3 = new CheckThreadName();
        t1.start();
        System.out.println("t1 : " + t1.getName());
        t2.start();
        System.out.println("t2 : " + t2.getName());
        t3.start();
        System.out.println("t3 : " + t3.getName());
    }
}
```

Output:

```
t1 : Thread-0  
Thread is run.  
t2 : Thread-1  
t3 : Thread-2  
Thread is run.  
Thread is run.
```

## Example for Set Thread Name

Source Code:

```
public class SetThreadName extends Thread{  
    public void run(){  
        System.out.println("Thread is run.");  
    }  
    public static void main(String args[]){  
        SetThreadName t1 = new SetThreadName();  
        SetThreadName t2 = new SetThreadName();  
        SetThreadName t3 = new SetThreadName();  
        t1.start();  
        t1.setName("Mythread1");  
        System.out.println("t1 : " + t1.getName());  
        t2.start();  
        t2.setName("Mythread2");  
        System.out.println("t2 : " + t2.getName());  
        t3.start();  
        t3.setName("Mythread3");  
        System.out.println("t3 : " + t3.getName());  
    }  
}
```

Output:

```
t1 : Mythread1  
Thread is run.  
Thread is run.  
t2 : Mythread2  
t3 : Mythread3  
Thread is run.
```

## Check currentThread Name

Current Thread का नाम जानने के लिए `currentThread()` इस method का इस्तेमाल किया जाता है ।

```
public static Thread.currentThread()
```

`currentThread()` method current thread के instance का reference return करता है ।

## Example for Get Current Thread Name

Source Code:

```
public class CurrentThreadName extends Thread{
    public void run(){
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String args[]){
        CurrentThreadName t1 = new CurrentThreadName();
        CurrentThreadName t2 = new CurrentThreadName();
        CurrentThreadName t3 = new CurrentThreadName();
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output:

```
Thread-1
Thread-0
Thread-2
```

## Joining Thread

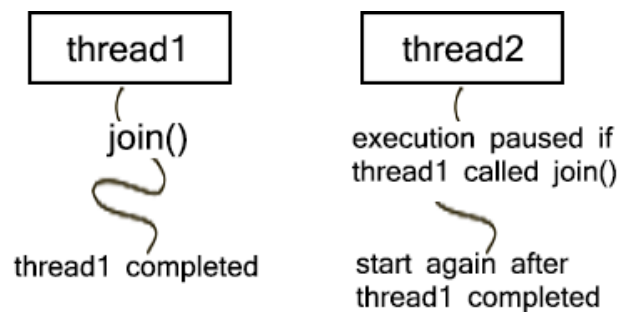
किसी current thread को hold पर रखा जाता है । जब तक join method से specified किया हुआ thread terminate नहीं हो जाता तब तक current thread hold पर रहता है ।

## Syntax for join() Method

```
final void join() throws InterruptedException
```

```
final void join(long milliseconds) throws InterruptedException
```





## Example for Joining Thread or join() Method

```
class JoinThread extends Thread{
public void run(){
int i;
for( i=0; i<5; i++ ){
try{
Thread.sleep(500);
}catch(InterruptedExcepion e){
e.printStackTrace();
}
System.out.println(i);
}
}

public static void main(String args[]){
JoinThread t1 = new JoinThread();
JoinThread t2 = new JoinThread();
JoinThread t3 = new JoinThread();
t1.start();
try{
t1.join();
}
catch(InterruptedExcepion e){
e.printStackTrace();
}
t2.start();//both threads starts after t1 thread completed.
t3.start();
}
}
```

Output:

```
0
1
2
3
4
0
0
1
1
2
2
3
3
4
4
```

## Example for Joining Thread or join(long milliseconds) Method

Example में देखे, तो join method के साथ का thread अपना task complete करने 2000(2 seconds) लगाता है । लेकिन हर loop के लिए thread 500 milliseconds(0.5 second) तक समय लेता है । कुल मिलाकर  $2000/500 = 4$  बार पहले execute होता है और thread2 और thread3 में thread1 अपना बचा हुआ task complete कर लेता है ।

Source Code:

```
class JoinThread extends Thread{
    public void run(){
        int i;
        for( i=0; i<5; i++ ){
            try{
                Thread.sleep(500);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
            System.out.println(i);
        }
    }
}
```

```
public static void main(String args[]){
    JoinThread t1 = new JoinThread();
    JoinThread t2 = new JoinThread();
    JoinThread t3 = new JoinThread();
    t1.start();
    try{
        t1.join(2000);
    }
    catch(InterruptedException e){
        e.printStackTrace();
    }
    t2.start();//both threads starts after t1 thread completed.
    t3.start();
}
}
```

Output:

```
0
1
2
3
0
4
0
1
1
2
2
3
3
4
4
```

# Sleeping Thread

sleep() method में thread को कुछ milliseconds तक pause किया जाता है ।

## Syntax for sleep() Method

```
public static void sleep(long miliseconds) throws InterruptedException
```

इस Syntax में जो nanoseconds है उसे 0 से 999999 तक limit होती है, अगर इससे ज्यादा होता है तो InterruptedException आ जाता है ।

```
public static void sleep(long miliseconds, int nanoseconds) throws InterruptedException
```

## Example for sleep() Method for Thread

Source Code:

```
class SleepThread extends Thread{
    public void run(){
        int i;
        for( i=0; i<5; i++ ){
            try{Thread.sleep(500);
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
            System.out.println(i);
        }
    }
    public static void main(String args[]){

        SleepThread t1 = new SleepThread();
        SleepThread t2 = new SleepThread();
        SleepThread t3 = new SleepThread();

        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output:

```
0
0
0
1
1
1
2
2
2
3
3
3
4
4
4
```

## Yield() Method

yield() method अपने current thread को pause करके दुसरे same priority वाले thread को चलाने की अनुमति देता है ।

yield() method thread को Running state से Runnable state पर लाता है ।

**For eg.** अगर thread1 पर yield method को call किया जाता है और thread2 की priority; thread1 से ज्यादा है, तो thread पर कोई फर्क नहीं पड़ता ।

## Syntax for yield() Method

```
public static void yield()
```

## Source Code:

```
class YieldThread1 extends Thread
{
    public void run(){
        int i;
        for ( i=0; i<5; i++ ){

            System.out.println("class YieldThread1");
            Thread.yield();
        }
    }
}

class YieldThread2 extends Thread{
    public void run(){
        int i;
        for ( i=0; i<5; i++ ){

            System.out.println("class YieldThread2");
            Thread.yield();
        }
    }

    public static void main(String[] args){
        YieldThread1 t1 = new YieldThread1();
        YieldThread2 t2 = new YieldThread2();

        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.MAX_PRIORITY);

        t1.start();
        t2.start();
    }
}
```

## Output:

```
class YieldThread2
class YieldThread2
class YieldThread2
class YieldThread2
class YieldThread2
```

```
class YieldThread1
class YieldThread1
class YieldThread1
class YieldThread1
class YieldThread1
```

## Synchronization

Java में Multithreading के लिए synchronization ये एक महत्वपूर्ण विषय है | Java में synchronization के लिए synchronized keyword का इस्तेमाल किया जाता है |

Java के synchronization में एक बार में एक ही thread को access किया जाता है |

### Why use Synchronization?

Java में जब दो threads या दो से अधिक threads का इस्तेमाल किया जाता है, तब एक साथ ये threads simultaneously चलते हैं | इससे thread से इस्तेमाल किये गए data corrupt हो सकता है | इसी समस्या जब पैदा होती है तब Java में synchronization का इस्तेमाल किया जाता है | जब thread के लिए Synchronization का इस्तेमाल किया जाता है तो एक बार में shared resource पर एक ही thread को एक बार में चलाया जाता है, इससे thread की consistency/concurrency की problem खत्म हो जाती है |

Synchronization के लिए दो प्रकार से किया जाता है |

1. Method Synchronized
2. Synchronized Block

### Example for no Synchronization

Source Code:

```
//SynchroWithout.java
class A{
void disp(int a){
for( int i=0; i<5; i++ ){
try{
Thread.sleep(500);
}
catch(InterruptedException e){
e.printStackTrace();
}
System.out.println("Thread Name : " +
Thread.currentThread().getName() + " : " + (i+a));
}
}
}
```

```

class B extends Thread{
A a;
B(A a){
this.a=a;
}
public void run(){
a.disp(1);
}
}
class C extends Thread{
A a;
C(A a){
this.a=a;
}
public void run(){
a.disp(100);
}
}
class SynchroWithout{
public static void main(String args[]){
A a = new A();
B t1=new B(a);
C t2=new C(a);
t1.start();
t2.start();
}
}

```

Output:

```

Thread Name : Thread-0 : 1
Thread Name : Thread-1 : 100
Thread Name : Thread-0 : 2
Thread Name : Thread-1 : 101
Thread Name : Thread-0 : 3
Thread Name : Thread-1 : 102
Thread Name : Thread-0 : 4
Thread Name : Thread-1 : 103
Thread Name : Thread-0 : 5
Thread Name : Thread-1 : 104

```



## Example with Synchronization

Source Code:

```
//SynchroWithout.java
class A{
    synchronized void disp(int a){
        for( int i=0; i<5; i++ ){
            try{
                Thread.sleep(500);
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
            System.out.println("Thread Name : " +
                Thread.currentThread().getName() + " : " + (i+a));
        }
    }
}

class B extends Thread{
    A a;
    B(A a){
        this.a=a;
    }
    public void run(){
        a.disp(1);
    }
}

class C extends Thread{
    A a;
    C(A a){
        this.a=a;
    }
    public void run(){
        a.disp(100);
    }
}

class SynchroWithout{
    public static void main(String args[]){
        A a = new A();
        B t1=new B(a);
        C t2=new C(a);
        t1.start();
        t2.start();
    }
}
```

## Output:

```
Thread Name : Thread-0 : 1  
Thread Name : Thread-0 : 2  
Thread Name : Thread-0 : 3  
Thread Name : Thread-0 : 4  
Thread Name : Thread-0 : 5  
Thread Name : Thread-1 : 100  
Thread Name : Thread-1 : 101  
Thread Name : Thread-1 : 102  
Thread Name : Thread-1 : 103  
Thread Name : Thread-1 : 104
```

## Introduction

Java में हर एक program में Java के package का इस्तेमाल होता है | Java के Packages में sub-packages, classes और sub-packages होते हैं | Packages का concept 'data encapsulation' की तरह होता है |

Java में Packages के लिए दो प्रकार हैं |

1. Built-in Packages
2. User-defined Packages

## In Built Packages

इन packages में कई सारे classes मौजूद होते हैं जो Java API का एक हिस्सा हैं | For Eg. java.lang, java.io, java.applet, java.awt .

**java.lang** : ये java के program में already import होता है , इसको अलग से import करने की जरूरत नहीं होती है | इसमें सभी data types, String Methods, Characters Methods, Math Methods होते हैं |

**java.io** : यहाँ पर console I/O, File I/O जैसे आदि Input/Output Operations इस package को लेकर किया जाते हैं |

**java.applet** : Applets को create करने के लिए इस package का इस्तेमाल किया जाता है |

**java.awt** : ये Windows GUI Application के लिए इस्तेमाल किया जाता है , जिसमें Buttons, Frames, menu इत्यादी होते हैं |

अगर किसी package के सभी classes को इस्तेमाल करना हो तो,

Syntax :

```
import package_name.*;
```

Example

```
import java.io.*; //import all classes from io package
```

अगर किसी package से एक ही class को इस्तेमाल करना हो तो,

Syntax :

```
import package_name.Class_name;
```

Example

```
import java.io.File; //import File classe from io package
```

# User Defined Packages

## Rules For Creating Packages

- Packages को create करने के लिए package keyword के साथ package का नाम लिखा जाता है ।
- Packages के अन्दर कोई main method नहीं होना चाहिए ।
- Packages में class name और interface के साथ public ये access modifier होना चाहिए ।
- Packages को save करना हो तो public class या public interface का उपयोग करे ।

Source Code:

```
//A.java
package pack1;
public class A{
    public void disp(){
        System.out.println("class A");
    }
}
```

```
//B.java
package pack2;
import pack1.*;

public class B{
    public static void main(String args[]){
        A a = new A();
        a.disp();
    }
}
```

Syntax for Compile Package

```
javac -d . class_name.java    //or
javac -d directory class_name.java
```

Example for Compile Package

```
javac -d . A.java    //same directory
javac -d Packages/ A.java    //User-defined directory
```

## Syntax for Run Package

```
java pack_name.class_name
```

## Example for Run Package

```
java pack2.B
```

## Compile and Run Package

ये ऊपर दिए हुए packages का output है | यहाँ पर '-d' ये folder create करके उस class की .class file उसमें create करता है | .(dot) का मतलब जहाँ पर java file को save किया गया है उसी directory पर folder create किया जाता है या वहाँ पर user द्वारा create किये directory पर भी ये folder create होता है |

### Output:

```
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . A.java //Compile class A
C:\Program Files\Java\jdk1.8.0_111\bin>javac -d . B.java //Compile class B
C:\Program Files\Java\jdk1.8.0_111\bin>java pack2.B //Run class B
class A
```