



# JAVA BASIC CONCEPTS

EDITED BY : JAYASHANKA DESHAN

# Content

Chapter 01 : Introduction	<ul style="list-style-type: none"><li>1.1 Evolution Of Java Programming Language</li><li>1.2 Types Of Java Applications</li><li>1.3 Java Platforms / Editions</li><li>1.4 Important Features Of Java</li><li>1.5 Types Of Programming Languages</li></ul>
Chapter 02 : Install JDK Programming Environment	What is Java Development Kit
Chapter 03 : Structure of Java Program	<ul style="list-style-type: none"><li>3.1 Packages</li><li>3.2 Class</li><li>3.3 Class Name</li><li>3.4 Program Drive(main method)</li><li>3.5 Introduction of public static void main(String[] args)</li></ul>
Chapter 04 : How to Run Java Program On Your Pc	
Chapter 05 : Comments and Variables	<ul style="list-style-type: none"><li>5.1 Comments<ul style="list-style-type: none"><li>Single line comments</li><li>Multi line comments</li></ul></li><li>5.2 Variable mean</li></ul>

Chapter 06 : Java Data types	6.1 Numerical Data Types. 6.2 List Data Type (array) 6.3 Strings Data Type
Chapter 7 : Java Operators	7.1 Arithmetic Operator 7.2 Relational Operator 7.3 Logical Operator 7.4 Bit Operator
Chapter 8 : User Inputs and Outputs ( java Scanner Class)	
Chapter 9 : Control Statements in Java	
Chapter 10 : Loops	
Chapter 11 : Arrays	
Chapter 12 : Array list	
Chapter 13 : OOP (Object Oriented Concepts)	13.1 class and objects 13.2 methods 13.3 Constructors 13.4 Access Modifiers 13.5 Non Access Modifiers 13.6 Method overriding and overloading

	<b>13.7 Up casting and down casting</b> <b>13.8 Polymorphism</b> <b>13.9 Encapsulation</b>
<b>Chapter 14 : Java I/O Streams</b>	





## Chapter 01 : Introduction

Java කියන්නේ වර්තමානයේ ජනප්‍රියම object oriented programming language එකක්. ජාවා (Java) පරිගණක භාෂාව(Programming Language) පරිගණක ක්‍රමලේඛනය(Computer Programming) පිළිබඳව උනන්දුවක් දක්වන ඕනෑම කෙනෙක්ට ගොඩක් වැදගත් වෙන පරිගණක භාෂාවක් . පරිගණක මෘදුකාංග ඉන්ජිනේරු (Software Engineering) තොරතුරු හා සන්නිවේදන තාක්ෂණය (ICT)වගේ උපාධි පාඨමාලා වල ඇතුළත් වෙලා තියන ක්‍රමලේඛන භාෂාවක්.

### **Java Versions :**

JDK Alpha and Beta (1995)

JDK 1.0 (1996)

JDK1.1(1997)

J2SE 1.2(1998)

J2SE 1.3(2000)

J2SE 1.4(2002)

J2SE 5.0(2004)

Java SE 6 (2006)

Java SE 7 (2011)

Java SE 8 (2014)

Java SE 9 (2017)

Java SE 10 (2018)

### 1.1 Evolution Of Java Programming Language

ජාවා ක්‍රමලේඛන භාෂාව 1991 දී ජේම්ස් ගොස්ලින් (James Gosling) විද්‍යාඥයා ඇමරිකාවේ කොලරාඩෝ වල දී කරපු විශේෂ ව්‍යාපෘතියක ප්‍රතිපලයක් ලෙස Java නිර්මාණය වන්නේ සන් මයික්‍රොසිස්ටම්ස් සමාගමේ වන අතර ව්‍ත්මානයේ දී එය ඔරකල්(Oracle) සමාගම සතු වේ. Java කියන්නේ කෙනෙකුට පහසුවෙන් ඉගෙනගන්න පුළුවන් පරිගණක භාෂාවක්. ජාවා ලෝකේ දැනට භාවිතා කරන ජනප්‍රියම ඉහල මට්ටමේ(High level) ක්‍රමලේඛන භාෂාවක්. Oracle වෙබ් පිටුවට අනුව බිලියන 3 ක් උපකරණවල ධාවනය වෙනවා. උදාහරණ ලෙස ඩෙස්ක්ටොප් යෙදුම් වන ඇක්‍රොබැට් රිඩර් , මීඩියා ප්ලේයර් , ඇන්ටි -වයිරස ආදිය සහ [lrctc.co.in](http://lrctc.co.in), [javatpoint.com](http://javatpoint.com) වැනි වෙබ් යෙදුම් , බැංකු යෙදුම් ,ජංගම , නිහිත පද්ධති (Embedded System), ස්මාට් කාඩ් , රොබෝ , පරිගණක ක්‍රීඩා ආදිය දැක්විය හැක.

## 1.2 Types Of Java Applications

**(01)Standalone Applications :-** මෙහෙයුම් පද්ධතියේ සේවාවන්ගෙන් ස්වාධීනව ක්‍රියාත්මක විය හැකි යෙදුම් වේ. මේවා පරිගණකයේ ස්ථාපනය කිරීමට අවශ්‍ය නොවේ. මතක උපාංගය(Disc or Flash Drive) පමණක් භාවිතා කරමින් ක්‍රියාත්මක විය හැක. උදාහරණ ලෙස Anti Virus, Operating System Installers දැක්විය හැක.

**(02)වෙබ් යෙදුම්(web Applications):-** වෙබ් පිටුවක සේවාදායකය හා පරිශීලකයා අතර සම්බන්ධතාවය ඇති කරන යෙදුම් වේ. පරිශීලකයා විසින් ලබාදෙන හෝ සේවාදායකය පරිශීලකයාගෙන් ලබාගන්නා ආදාන හා දත්ත අනුව වෙබ් පිටුවේ/අඩවියේ වෙනස්කම් සිදු වේ. උදාහරණ ලෙස jsp,servlet ජාවා හි වෙබ් යෙදුම් වේ.

**(03)Enterprise Applications :-** යම් ව්‍යාපාරයක්, ආයතනයක් හෝ රජයක් පරිපාලනයේ පහසුව සඳහා නිර්මාණය කරන විවිධ කායි පද්ධතීන් රාශියක් එකිනෙකට සම්බන්ධ කරන ඉතා සංකීර්ණ මෘදුකාංග පරිසරයක් සරලව මෙලෙස හඳුන්වයි. මෙහිදී ඉහළ ආරක්‍ෂාව ,බර සමතුලිතතාව (load balancing) , පොකුරු කිරීම (clustering) වැනි කරුණුවලට වැඩි අවධානයක් යොමු කරයි. (Enterprise Java Beans – EJB භාවිතා කර මෙවැනි පද්ධති නිර්මාණය කරයි )

**.(04)ජංගම යෙදුම් :-**ජංගම උපාංග සඳහා යෙදුම් නිර්මාණයේදී යොදාගනී .

දැනට Android සහා java ME සඳහා භාවිත වේ.

## 1.3 Java Platforms / Editions

**1) ජාවා එස් ඊ- JAVA SE(සම්මත සංස්කරණ ) :-**ජාවා ක්‍රමලේඛණ වේදිකාවකි.එයට ජාවා ක්‍රමලේඛණ ඇතුලත් වේ.

**Example :** Java.lang,java.io, java.net, java.util, java.sql වේ.

ගණිතය සඳහා OOPs, String, Regex, Exception, inner classes, Multi threading, I/O Stream ආදිය දැක්විය හැක .

**2) JAVA EE :-** වෙබ් සංවර්ධනය සඳහා භාවිත කරයි. උදාහරණ ලෙස JSP, Web Services, EJB, JPAආදිය දැක්විය හැක.

**3) JAVA ME( ජාවා මයික්‍රො සංස්කරණ ) :-**ජංගම උපාංග සඳහා යෙදුම් සංවර්ධනයට යොදාගනී.

**4) JAVA FX :-** පරිගණක යෙදුම් හා උසස් මට්ටමේ අන්තර්ජාල යෙදුම් සංවර්ධනයට යොදාගනී. Ex: API

ජාවා යෙදුම් වර්ග(Type Of Java Applicaton) හා ජාවා වේදිකා හෝ සංස්කරණ(Java Platforms / Editions) එසේ දැක්විය හැක.



## 1.4 Important Features Of Java

Java භාෂාව භාවිතයෙන් කේතයක් එක සැරයක් ලිවීම ඒක ඕනෑම පරිගණකයක ක්‍රියාත්මක කරන්න පුළුවන් වීම WORA (Write Once Run Anywhere) මෙහි විශේෂ ලක්ෂණයක් වෙනවා. ඒ වගේම java වල තියෙන සුච්ඡේදී ලක්ෂණයක් තමයි java වලින් ලියන ලද කේත machine code එකට හැරවීම සඳහා compilation සහ interpretation යන ක්‍රම දෙකම භාවිත වීමයි මෙතනදී source code එක .class extension එක සහිත byte code එකක් බවට පත් කරනවා. මෙම byte code එක interpreter එක මගින් machine code එක බවට හරවනවා.

## 1.5 Types Of Programming Languages

මෙතෙක් නිර්මාණය වූ පරිගණක භාෂා මෙලෙස වර්ග දෙකකට බෙදා දැක්විය හැක.

### 1. low level computer languages

Machine language

Assembly language

### 2. High level computer languages

උදාහරණ:- Fortran, algol, cobol, visual basic, basic pascal, C, C++, java, .net, C#, perl, prolog, python, lisp, php...

## Evolution Of Programming Languages :

1945 පමණ ජපාන් වොන් නියුමාන්ගේ අදහසකට අනුව පරිගණකයේ භෞතික තත්ව වෙනස් නොකොට ක්‍රමලේඛ මගින් විවිධ කාර්යයන් කරගනීම ඇරඹිණි. එතැන් සිට මෙතෙක් දක්වා නිර්මාණය වූ සියළු පරිගණක භාෂා ප්‍රධාන ආකාර 4කට බෙදා දැක්විය.

### First generation Computer languages

- ✚ මේ වර්ගයට අයත් වන්නේ යන්ත්‍ර භාෂාවයි. එහි පහත ලක්ෂණ පවතී,
- ✚ ක්‍රමලේඛ නිර්මාණය සඳහා 0 සහ 1 යන ද්වීමය සංඛ්‍යා පමණක් යොදා ගැනීම
- ✚ ක්‍රමලේඛ නිර්මාණය ඉතා දුෂ්කර කටයුත්තක් වීම
- ✚ ඉතා දියුණු මට්ටමේ ක්‍රමලේඛ නිර්මාණය කළ නොහැකි වීම
- ✚ කිසියම් පරිගණකයක තාක්ෂණයක් මූලික කර ගනිමින් නිර්මාණය කර ඇති නිසා වෙනත් තාක්ෂණයන් සහිත පරිගණක වල භාවිත් කළ නොහැකි වීම
- ✚ දෘඩාන්ත මත යැපෙන භාෂාවක් නිසා දෘඩාන්ත ගැන මනා අවබෝදයක් තිබිය යුතු වීම
- ✚ පරිවර්තනය කළ යුතු නොවේ. පරිගණකයට කෙලින්ම තෙරුම් ගත හැක. එම නිසා පරිවර්තක මෘදුකාන්ත අවශ්‍ය නොවේ
- ✚ එම නිසා ක්‍රමලේඛ වල වේගය වැඩිය

## Second generation Computer Languages

- + මේ යටතට ගැනෙන්නේ **assembly** පරිගණක භාෂාවයි
- + යන්ත්‍ර භාෂාවට වඩා තරමක් දියුණුය
- + ඉන්ග්‍රීසි අකුරු සහ ඉලක්කම්වලින් සකස්කළ, **add, sum** වැනි යෙදුම්ද භාවිතා කරයි
- + භාවිතය තරමක් පහසුය
- + දෘඩාන්ත මත තරමක් දුරට යැපෙන භාෂාවකි
- + ක්‍රමලේඛ ක්‍රියාත්මක කිරීම සඳහා **assembler** නම් පරිවර්තක භාවිතා කරයි
- + ක්‍රමලේඛ වල වෙගය 1වන පරම්පරාවට වඩා අඩුය

## Third Generation Computer Languages

- + මෙහි හැකියාවන් වැඩිය. ඉතා දියුණු මට්ටමේ ක්‍රමලේඛ නිර්මාණය කිරීම සඳහා යොදා ගනී
- + පරිගණක දෘඩාන්ත කොටස් මත රඳා නොපවතී
- + ක්‍රමලේඛ නිර්මාණය කිරීම හා නඩත්තු කිරීම පහසුය
- + සම්පාදක(**compiler**) සහ අර්ථවිනාසක(**interpreter**) නම් පරිවර්තක මෘදුකාන්ත භාවිතා කරයි
- + ක්‍රමලේඛ ක්‍රියාත්මක කිරීමේ වෙගය සාපේක්ශව අඩුය
- + උදාහරණ ලෙස **fortran, cobol, basic pascal, C, C++, visual basic, java** දක්විය හැක

## Fourth Generation Computer Languages

- + 3වන පරම්පරාවේ පරිගණක භාෂා සතු ලක්ෂණ බොහෝමයක් පවතී
- + සම්පාදක(**compiler**) සහ අර්ථවිනාසක(**interpreter**) නම් පරිවර්තක මෘදුකාන්ත භාවිතා කායි
- + කෘතීම බුද්ධියක් සහිත ක්‍රමලේඛ නිර්මාණය කිරීම සඳහා භාවිතා කරයි
- + උදාහරණ ලෙස **prolog, LISP, mercury, python** දැක්විය හැක

## Language Translators

1 assembler

2 compiler

3 interpreter

### assembler :

**assembly** පරිගණක භාෂාවෙන් නිර්මාණය කරන ලද ක්‍රමලේඛ පරිවර්තනය කිරීම සඳහා **assembler** භාවිතා කරයි.

## Compiler & interpreter :

3වන හෝ 4වන පරම්පරා වල පරිගණක භාෂා වලින් ලියන ලද ක්‍රමලේඛ පරිවර්තනය කිරීම සඳහා **Compiler** සහ **interpreter** භාවිතා වේ.

### interpreters

අර්ථ විනයසක මෘදුකාන්ත මගින් ක්‍රමලේඛ ක්‍රියාත්මක වන අවස්තාවේ එම ක්‍රියාත්මක වන ජේලිය පමණක් පරිවර්තනය කර දීම සිදු කරනු ලබයි.

ජේලියෙන් ජේලිය(**line by line**) පරිවර්තනය වීම නිසා ක්‍රමලේඛය ක්‍රියාත්මක වන අවස්තාවේ නැවත නැවත පරිවර්තනය වීම සිදුවේ.

එම නිසා ක්‍රමලේඛ ක්‍රියාත්මක වීමේ වේගය සම්පාදක වලට වඩා අඩුය.

ක්‍රමලේඛ වල වැරදි නිවැරදි කිරීම(**debug**) සම්පාදක වලට සාපේක්ශව පහසුය.

### Compilers

සම්පාදක මෘදුකාන්ත මගින් ක්‍රමලේඛ සම්පූර්ණයෙන්ම පරිවර්තනය කිරීම සිදු වේ.

ක්‍රමලේඛ ක්‍රියාත්මක වීමේ වේගය වැඩිය.

ක්‍රමලේඛ වල වැරදි නිවැරදි කිරීම(**debug**) අර්ථ විනයසක වලට සාපේක්ශව අපහසුය.

ක්‍රියාත්මක වීමේදී නැවත පරිවර්තනය වීමක් අවශ්‍ය නොවේ

එම නිසා පරිවර්තනයෙන් පසු ලැබෙන පරිවර්තක ක්‍රමලේඛය වෙනම ගොනුවක් සේ ගබඩා කළ හැක.

### **Note :**

සම්පාදක හෝ අර්ථ විනයසක භාවිතයෙන් පරිවර්තනය කරන ලද ක්‍රමලේඛ කෙලින්ම යන්ත්‍ර භාෂාවට හැරේ.

**ප්‍රභව ක්‍රමලේඛය (source code) ----> විෂය ක්‍රමලේඛය (object code)**

නමුත් මෙසේ ප්‍රභව ක්‍රමලේඛය කෙලින්ම විෂය ක්‍රමලේඛයට පරිවර්තනය නොකරණ පරිගණක භාෂාද ඇත.ඒ සඳහා හොඳම උදාහරණය ජාවා (**java**) පරිගණක භාෂාවයි.එවැනි භාෂා මගින් ක්‍රමලේඛ පරිවර්තනය කරන්නේ මෙසේය.

**source code --> byte code --> object code**

මෙහි ඇති විශේෂත්වය වන්නේ යම් ක්‍රමලේඛයක **source code** එක හා **byte code** එක ඕනෑම පරිගණකයකදී සමාන වන නමුත් **object code** එක ඒ අවස්තාවේ ක්‍රමලේඛය ක්‍රියාත්මක වන **platform** එක(මෙහෙයුම් පද්ධතිය) අනුව වෙනස් වීමයි.එහිදී **source code** එක **byte code** බවට පත් කිරීමට **compiler** වර්ගයේ පරිවර්තකයක්ද, **byte code** එක **object code** බවට පත් කිරීමට **interpreter** ද භාවිතා කරනු ලබයි

## Chapter 02 : Install JDK Programming Environment

### **What is JDK (JAVA Development Kit)**

ජාවා වැඩසටහන්(java Programs) ලිවීම සඳහා ජාවා වේදිකා(java Platform) සම්මත සංස්කරණය හෝ “ජාවා එස්ඊ (Java SE)” ලෙස නම් කර ඇති ජාවා සංවර්ධන කට්ටලය (JDK) ඕන වෙනවා. ජාවා සංවර්ධන කට්ටලය(JDK) ඔරකල් (Oracal) වෙතින් කොටසක් නොමිලේ ලබා ගත හැකිය. JDK (Java SE) (සංස්කරණ ගැන වැඩි විස්තර පසු ගිය ලිපියෙන් සඳහන් කලා )එය මෙම වේබ් අඩවිය <http://www.oracle.com/> වේ.

දැන් එතකොට “JDK”(Java Development Kit) හා “JRE”(Java Runtime)කියන්නේ මොකද්ද??

ජාවා වැඩසටහන් ක්‍රියාත්මක කිරීම සඳහා JRE (Java Runtime) ඕන වෙනවා. JDK (Java Development Kit), JRE ජාවා වැඩසටහන් ලිවීමට මෙන්ම ක්‍රියාත්මක(Run) කිරීමට ඕන වෙනවා. තවත් සරලව කිවහොත්, JRE යනු JDK හි උප කුලකයකි. ඔබ ජාවා වැඩසටහන් ලියනවා, ඔබ විසින් තමයි JDK ඇතුළත් කරන්නේ, එකට JRE ඇතුළත් වෙනවා.

✚ Please refer that link for install java on your windows os : <https://youtu.be/IJ-PJbvJBGs>

✚ JDK and JAVA documentation download : <https://www.oracle.com/java/technologies/javase-downloads.html>

## Chapter 03 : Structure of Java Program

### Structure Mean ?

ක්‍රමලේඛණ ආකෘතියක් කියන්නේ භාෂා සංවර්ධකයා විසින් ක්‍රමලේඛකයා(Programmer) වෙත නිකුත් කරන සම්මත ආකෘතියක් .

දැන් , බලමු මොකද්ද Java ක්‍රමලේඛණ ආකෘතිය(Structure of Java Program) කියන්නේ කියලා.

ජාවා යෙදුම් සංවර්ධනය (developing java Application)කිරීම සඳහා සන් මයික්‍රෝ සිස්ටම්(Sun Micro System ) විසින් ජාවා ක්‍රමලේඛකයින්(Java Programmers) සඳහා වියුහය නියම කර ඇත.ඒවා පහත සඳහන් කර ඇත.

```
import java.util.Scanner;
public class class
{
    public static void main(String args[])
    {
        Scanner x = new Scanner(System.in);
        System.out.print("Student Name : ");
        String name = x.nextLine();
    }
}
```

Package details

Class name

Void display( );

Block of statement

Data type

### 3.1 Package (පෙරනිම් පැකේජය)

පැකේජය පත්ති, අතුරු මුහුණත් හා උප-පැකේජ වල එකතුවක් තිබේ. උප පැකේජයක පත්ති, අතුරුමුහුණත් සහ උප උප පැකේජ ආදිය ඇතුළත් වේනවා. **Java.lang. \***; පෙරනිම් පැකේජය ආනයනය (import)කරන අතර මෙම පැකේජය පෙරනිම් පැකේජය ලෙස හැඳින්වේ.

Example:

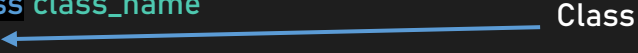
```
import Java.lang.*; //import all sub packages in lang package
import Java.io.*; //import all sub packages in io(input and output) package
import Java.util.*; //import all sub packages in util package
```

## 3.2 Class (පංතිය)

පංතිය කියන්නේ පරිශීලකට (User) ඕන කරන දත්ත සංවර්ධනය(Develop) වර්ගය (type) කිරීම සඳහා භාවිතා කරන යතුරු පදයක්(Keyword) වන අතර සෑම ජාවා වැඩසටහනක්ම (Program)පන්ති සංකල්පයකින් ආරම්භ වෙන්න ඔනේ.

Example :

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



## 3.3 Class Name (පංති නම)

“ClassName” කියන්නේ ජාවා වල වලංගු විවල්ය නාමයක් වන අතර එය පන්තියේ නමක්(ClassName) ලෙස සලකනු ලබන අතර ජාවා හි සෑම පන්ති නාමයක්ම(ClassName) පරිශීලක අර්ථ දක්වන (User-defined)දත්ත වර්ගයක්(data type) ලෙස සලකනවා.

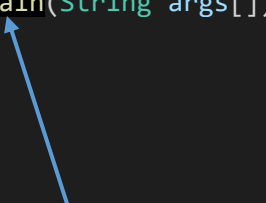
```
public class class_name
```



## 3.4 main () -Program Driver (වැඩසටහන් ධාවක)

හැම වැඩසටහනක් ක්‍රියාත්මක කිරීම ආරම්භ වෙන්නේ මෙකේන්. එ නිසා main () ක්රමය වැඩසටහන් ධාවක(Program Driver ) ලෙස හැඳින්වේ.

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



Program Driver

### 3.5 Introduce public, static, void, main, ( String args [ ] )

#### Public (පොදු පදය)

ජාවා පරිසරයේ(platform) ඕනෑම තැනක `main ()` ක්රමය ඕනෑම තැනක සිට ක්රියාත්මක කරන්න පුළුවන් `.main ()` එකට ක්රමලේඛකයා (programmer) විසින්ම ප්රවේශ විය යුතු අතර එම නිසා එය පොදු විය යුතුය.

#### Why we use void ?

ජාවා ක්රමය කිසිදු අගයක් ලබා නොදෙන(not return any Value) නිසා එහි ප්රතිලාභ වර්ගය (return type) අවලංගු විය යුතුය.

#### Why we use static ?

ජාවා ක්රමය ක්රියාත්මක වන්නේ ජාවා ක්රමලේඛය ක්රියාත්මක කිරීමේදී එක් වරක් පමණක් වන අතර එබැවින් එහි ස්වභාවය ස්ථිතික (static) විය යුතුය.

#### String args [ ]

**String type** අගයන් විධාන රේඛා තර්ක(Command) රඳවා ගැනීමට භාවිතා කරන **String Array** අරාචකි. මෙය අපට පහත ඕනෑම ආකරයකටම භාවිතා කරන්න පුළුවන්.

අපට **String** පරාමිති (Parameter) **var-arg** ගත හැකිය.

Syntax

`main ( String [ ] args ) -> main ( String ... args )`

අපට **modifier**( නවීකරණයේ ) අනුපිළිවෙල, වෙනස් කළ හැකිය.

Syntax

`public static` we can take `static public`

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

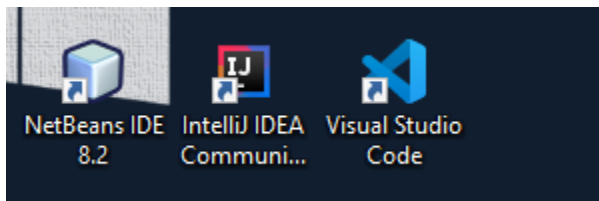
## Chapter 04 : How to Run Java Program On Your Pc

### **How To Run Java Program On Your PC**

ජාචා වැඩසටහනක් ක්‍රමලේඛනය (Java program) කර මෘදුකාංගයක්(software) බවට පත්කිරීම හා ධාවනය කරවීම අදියර (3) ක් සිදුවේ.

- 1.ජාචා ක්‍රමලේඛ (Program) ක්‍රමලේඛනය කර ගබඩා කිරීම.
2. ජාචා ක්‍රමලේඛය (Program) අතර මැදි වීඛාන ගොනුවක් කිරීම.
- 3 .ජාචා ක්‍රමලේඛය (Program) ඩෘදුකාංගයක් ලෙස ක්‍රියා කරවීම.

### What Software Use For Coding Java ?



Ex:

Visual Studio Code	}	IDE (integrated development environment)
NetBeans		
Jetbrain IntelliJ IDEA		
Eclipse		

OR

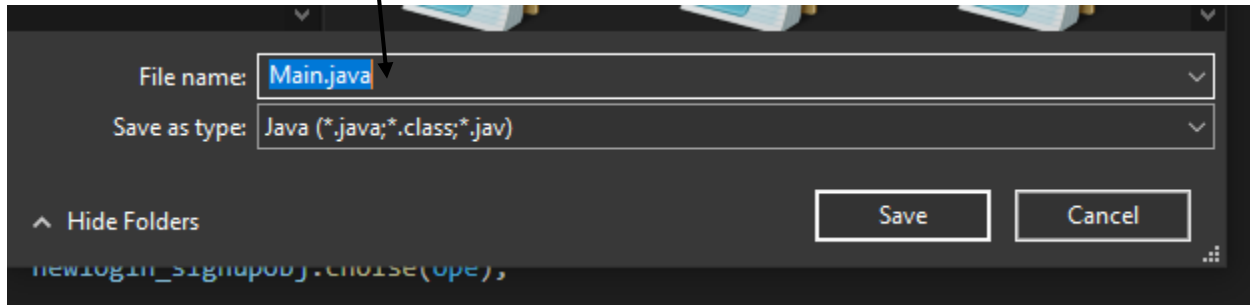
Sublime Text	}	Text Editor
Note Pad		
Note Pad++		



## How To Save Your Java File

ඡාචා ක්‍රමලේකයේ නම හා ගබඩා කරන ගොනුවේ නම (filename) සමාන වෙන්න ඔනි.

ගොනුවේ වර්ගය(extension) .java වෙන්න ඔනි.

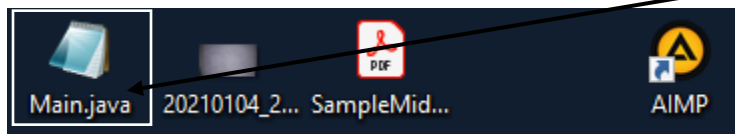


## Create Simple Java Program

```
public class Main
{
    public static void main(String[] args)
    {
        System.out.print("I AM JAYASHANKA DESHAN");
    }
}
```

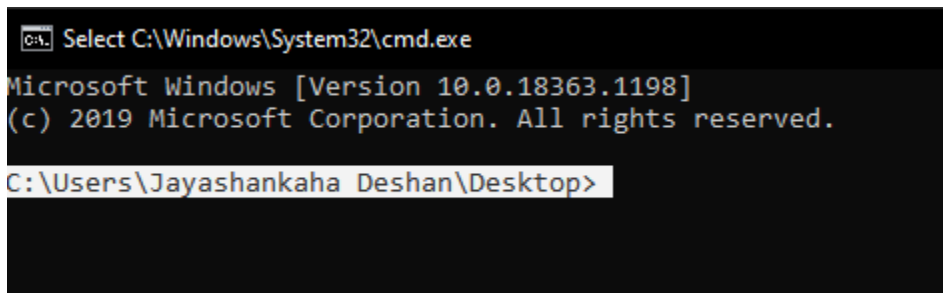
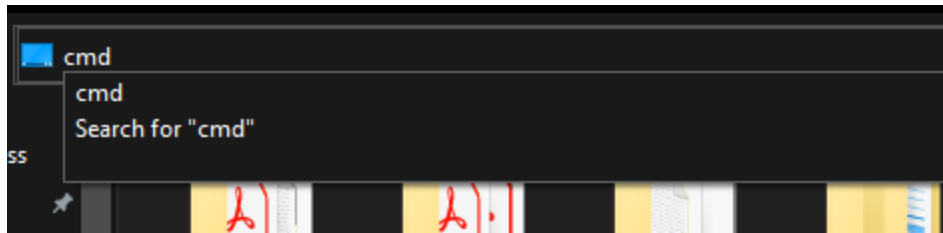
1. ලියපු ඡාචා ක්‍රමලේකනය කර ගබඩා කිරීම.

මම මේ ක්‍රමලේකනය කර ගබඩා කරන්නේ පරිගනකයේ DESKTOP එකේ "Main.java" කියලා.



2. ඡාචා ක්‍රමලේකය (Program) අතර මැදි වීඩාන ගොනුවක් කිරීම .

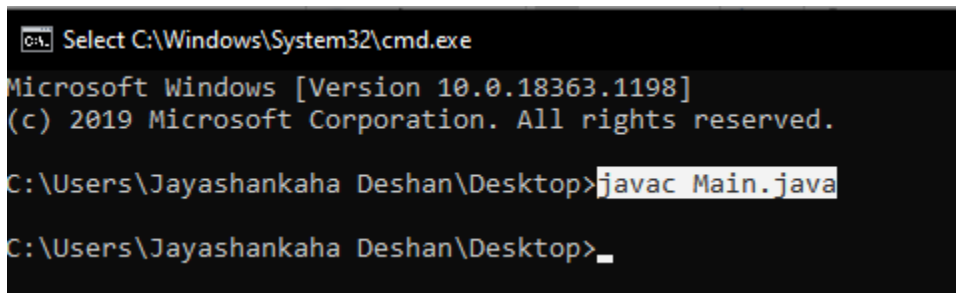
- Open CMD On Target Folder



- Enter java Compile command

Syntax:

**javac** **programe\_name.java**



When your program finish compile run it using above syntax:

**java** **programe\_name.java**

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
IDEA Community Edition 2020.2\lib\idea_rt.jar=50388:C:\Program Files\JetBrains\IntelliJ IDEA
Community Edition 2020.2\bin" -Dfile.encoding=UTF-8 -classpath
D:\Projects\IntelliJ\out\production\IntelliJ Main
I AM JAYASHANKA DESHAN
Process finished with exit code 0
```

## Chapter 05 : Comments and Variables

### 5.1 Comments ?

ලේසියෙන් පරිගණක ක්‍රමලේඛයක් කියවා වටහා ගැනීමට විවරණ(Comments)

යොදාගනී.පරිගණක ක්‍රමලේඛය යෙදී විවරණ(Comments)

යෙදීම ඉතා වැදගත් පුරුද්දකි.ඒ වගේම මේවා ක්‍රමලේඛණය(Program) සම්පාදකය (Compiler)

තුළ ධාවනය(run) නොවේ.එහි ක්‍රමලේඛණය

මොකක් සඳහා ලියන ලද්දක්ද,කොහොමද හඳුලා තියන්නේ යන කරුණු ක්‍රමලේඛණය තුළම ඇතුළත් කරනවා.මේවා Java වල යොදන විදිහ කොහොමද බලමු.

### Single line comments

මෙහිදී විවරණය ආරම්භ වන්නේ // යන ලකුණින් වේ.

```
public class main
{
    public static void main(final String args[])
    {
        System.out.print("Deshan Amarasinghe"); // print name
    }
}
```

Single line comment

### Multi line comment

මෙහිදී විවරණය ආරම්භ වන්නේ /\* යන ලකුණින් වන අතර අවසන් වන්නේ \*/ යන ලකුණින් වේ.

```
/*
Author name : deshan jayashanka
email : xxxxxx555@gmail.com
*/

public class main
{
    public static void main(final String args[ ])
    {
        System.out.print("Deshan Amarasinghe"); // print name
    }
}
```

Multi line comments

```

}
}

```

## 5.2 Variables mean?

පරිගණක මතකය තුළ විවිධ අගයන් තබාගෙන පරිශීලනය (access ) කිරීමට විචල්‍ය (variables) යොදා ගන්නවා.

හැමවිචල්‍ය(variables)කටම නමක් හා අගයක් තියනවා ඒවා පරිගණක මතකය තුළ රැඳෙන අතර පරිශීලනය (access ) කිරීමට ඒවා යොදා ගන්නවා.

```

public class main
{
    public static void main(final String args[])
    {
        int number1 = 100;
        float number2 = (float) 50.5;
        float answer = number1 * number2;
        System.out.print("Answer is : "+answer);
    }
}

```

Number1, number2 and answer are variables

## Variable types use in java

- 01) දත්ත ඒකක විචල්‍ය( Byte Variables)
- 02) අක්ෂර විචල්‍ය(Char Variables)
- 03) කෙටි පූණි සංඛ්‍යා විචල්‍ය( Short Variables)
- 04) පූණි සංඛ්‍යා විචල්‍ය( Int Variables)
- 05) දිගු පූණි සංඛ්‍යා විචල්‍ය( Long Variables)
- 06) කෙටි දශමය සංඛ්‍යා විචල්‍ය( Float Variables)
- 07) දිගු පූණි සංඛ්‍යා විචල්‍ය( Double Variables)
- 08) බූලීය විචල්‍ය( Boolean Variables)

Type	Size (in bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	$-2^{31}$ to $2^{31}-1$
long	64	$-2^{63}$ to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true or false

ජාවා විචල්‍යයන් ( Java Variables) භාවිතයේ දී ඔබ දැනගත යුතු කරුණු මොනවාද ??

- විචල්‍යයන් ( Variables) ලබා දෙනකුට අපි විචල්‍ය වර්ගය (Variable Type), විචල්‍ය නාමය (Variable Name), විචල්‍යයට අගයක් (Variable Value), වැනි ඒවා ලබා දෙන්න ඔනේ.

```
int number = 100;
// int = Variable data type
// number = Variable Name
// 100 = Variable Value
```

- මෙහිදී byte හා int වැනි විචල්‍ය විචල්‍ය නාම සඳහා යොදා ගත නොහැකිය.

```
int = 100;
float = 10.98;
byte = 65;
```

🚦 num, Num, NUM යන විචල්‍යයන් එකිනෙකට වෙනස් ජාලා විචල්‍යයන් ලෙස සැලකේ.

```
int NUM = 100;
```

```
int num = 100;
```

```
int Num = 100;
```

more about variables : [https://www.w3schools.com/java/java\\_variables.asp](https://www.w3schools.com/java/java_variables.asp)

## Chapter 06 : Java Data types

### Data Types Use In JAVA

විවිධ දත්ත වර්ග අපිට **code** කිරීමේදී හමු වෙනවා. ඉතින් **java** වලත් අපිට විවිධ දත්ත වර්ග භාවිත වෙනවා. ඒවා පහත උදාහරණ එක්කම ඔයාලට බලාගන්න පුලුවන්.

#### 6.1 Numerical Data Types.

අපිට ගන්න ඉලක්කම් අපේ **variable** එකකට දා ගන්න අවශ්‍ය නම් අපි **int, float, double, long** දත්ත ප්‍රරූපය භාවිත කරනවා.

(Numeric data types are numbers stored in database columns. These data types are typically grouped by: ... The exact numeric types are INTEGER , BIGINT , DECIMAL , NUMERIC , NUMBER , and MONEY . Approximate numeric types, values where the precision needs to be preserved and the scale can be floating)

Example :

```
int num1 = 100;

float num2 = 70.3f;

double num3 = 100.453;

long num4 = 2345609874343l;
```

Simple example 01 :

```
public class script1
{
    public static void main(String args[])
    {
        int a = 4;

        int b = 2;

        int c = a + b;

        int d = a - b;

        int e = a / b;

        int f = a * b;

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

```

        System.out.println(d);

        System.out.println(e);

        System.out.println(f);
    }
}

```

Expected output;

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
6
2
2
8

Process finished with exit code 0

```

Simple example 2 :

```

public class Statement
{
    public static void main(String[] args) {
        double number1 = 100.5;

        double number2 = 45.9;

        double number3;

        number3 = number1 + number2;

        System.out.print("Answer is : " + number3);
    }
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Answer is : 146.4
Process finished with exit code 0

```



## 6.2 List Data Type (array)

අපිට ජාව වල ලැයිස්තුවක් අවශ්‍ය නම් අපි පහත ආකාරයෙන් එය සකසා ගන්නව.මේකේ කියන විශේෂත්වය තමයි අපිට එක් වරකට එක් ලැයිස්තුවකට දැමිය හැක්කේ එක් දත්ත ප්‍රරූප වර්ගයක් විතරයි.උදාහරණයක් විදියට අපි ඉලක්කම් දාන ලැයිස්තුවකට අපිට වචන දාන්න බැහැ.

Example :

```
int a [] = { 1,3,2,6,5 } ;  
  
String b [] = { "Hello" , "Hi" , "Hey" } ;
```

## 6.3 Strings Data Type

String කියන්නේ ඇත්තම String කියන class එකේ object එකක් කියලා කියන්න පුළුවන්.ඒක භාවිතා කරන්නේ අකුරු ඒ කියන්නේ characters නිරූපණය කරන්නයි.මේක අයත් වෙන්නේ java.lang කියන package එකට.මේක අපි ජාවා වල code එකක් ලියනකොට import කරන්න අවශ්‍ය වෙන්නේ නම් නෑ.මොකද හේතුව කිව්වොත් ඒක ඉබේම import වෙලා තියෙන්නේ default. තව මේකත් අපි දන්න අනිත් class වගේම තමයි.මේකටත් constructor තියෙනවා ඒ වගේම methods තියෙනවා. හැබැයි ඊට අමතරව තවත් මෙන්ම මෙහෙම දේකුත් තියෙනවා.ඒ තමයි අකුරු සම්බන්ධ එකතු කිරීම් වලට + සහ += කියලා දෙකක් තියෙනව.මේ එකතු කිරීමට අපි කියනවා concatenation කියලත්.මොකද මේකෙදි එකතු කරන්නේ අකුරුනේ.ඉලක්කම් නෙමේ.

Example :

```
SLIIT + SCHOOL OF COMPUTING = SLIIT SCHOOL OF COMPUTING
```

අන්න ඒ වගේ..විශේෂයෙන්ම කියන්න ඕනේ කරන කාරණාවක් තියෙනවා. ඒ තමයි String කියන්නේ final class එකක්.ඒ කියනේ ඒකේ method override කරන්නවත් class එක extend කරන්නවත් බෑ.අපිට String object හදා ගන්න ක්‍රම දෙකක් තියෙනවා.

Method 01 :

```
public class string  
{  
    public static void main(String[] args)  
    {  
        String name = "R.W.D.J AMARASINGHE";  
    }  
}
```

මේකේදී එක String object එකක් සහ එක reference variable එකක් හැඳෙනවා. අපි දැන් හදපු එක ඒ කියන්නේ "R.W.D.J AMARASINGHE" කියන එක pool එකට යනවා. ඒ වගේම "name" කියන එක මගින් ඒක refer කෙරෙනවා.

## Method 02:

```
public class string
{
    public static void main(String[] args)
    {
        String name = "R.W.D.J AMARASINGHE"; // method 01

        String name2 = new String("R.W.D.J AMARASINGHE"); // method 02
    }
}
```

මෙහිදී String object දෙකක් හැඳෙනවා. එක reference variable එකක් තමයි පවතින්නේ. මොකද හේතුව කියනවා නම් අපි java වල new කියන keyword එක භාවිතා කරන නිසා තමයි අලුත් String object එකක් ඇති වෙන්නේ. මෙහිදී සාමාන්‍යයෙන් memory එකේ ඒ කියන්නේ RAM එකේ මේක හැදිලා "name2" වලින් එය refer වෙනවා. "R.W.D.J AMARASINGHE"; කියන එක pool එකට යනවා.

## More about Java Strings :

01. String කියන්නේ **immutable** object වර්ගයක්

02. String දක්වන්න ඔබේ double quotation ඒ කියන්නේ "" කියන පෙරලි කොමාවන් ඇතුළේ.

03. String අපිට String variable වලට assign කරන්න පුළුවන්.

04. අපිට String method වලට එහෙමත් නැත්නම් constructor වලට parameter එකක් ලෙසට pass කරන්න පුළුවන්.

## Immutable :

එකවරක් create කරගන්නායින් පස්සේ String ආයිත් වෙනස් කරන්න බෑ කියන එකයි. ඒකේ කිසිම method එකකින් String එක වෙනස් කරන්න බෑ. කෙලින්ම කියනවා නම් String හැදූවොත් හැදූවාමයි.. එහෙම object වලට අපි කියනවා immutable කියලා. මේවා පහසුවක් වෙනවා එක අතකට සමහර reference point වලදී. ඒ නිසා මේවගේ reference වෙනස් කිරීමේ නිසා String object එකට වෙනස් කිරීමක් සිදුවෙන්නේ නෑ.

Example :

```
public class immutable
{
    public static void main(String[] args)
    {
        String name ="DESHAN";

        name.concat("Jayashanka");

        System.out.println("Name is : "+name);
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" ".
Name is : DESHAN

Process finished with exit code 0
```

දැක්කා නේද කලින් අපි **define** කරපු **"name"** ට වෙච්ච දේ.අපි ඒකට **concat** කියලා තවත් වචනයක් **add** කරන්න උත්සාහ කලා.ඒත් **"name"** තිබ්බ විදිහමයි.ඒත් හිතන්නකෝ අපි මෙහෙම වැඩක් කරනවා කියලා ඉස්සෙල්ල වගේ නැතුව.

Example:

```
public class immutable
{
    public static void main(String[] args)
    {
        String name ="DESHAN";

        name = name.concat(" Jayashanka");

        System.out.println("Name is : "+name);
    }
}
```

## Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Name is : DESHAN Jayashanka

Process finished with exit code 0
```

🚦 ඉතින් අපි බලමු **immutability** එකේ වාසි ගැන...අංක එකටම පහසුව තමයි අවම **memory** ප්‍රමාණයක් භාවිතා කිරීම.

🚦 **immutability** වල අවාසි බලමු අපි දැන්...මේකේදීන් ප්‍රධානම අවාසිය ලෙසට දක්වන්න පුළුවන් වෙන්නේ අවම කාර්යක්ෂමතාවයයි.ඒ කියන්නේ අපිට ඉතා සුළු වශයෙන් කරන්න වෙන වෙනසකට පවා අපිට අලුත් **String object** එකක් හදන්න වෙනවා.

## Empty String

ඒ කියන්නේ **characters** මොකවත් නෑ කියන එකයි.ඒ නිසා ඒකේ **length** එක වෙන්නේ 0.

Example :

```
public class simple
{
    public static void main(String[] args)
    {
        String name = "";

        String name2 =new String("");
    }
}
```

ඒ වුනත් මේ කියන්නේ පහලින් දක්වන ජාතියේ එහෙම නෙවී හරිද...!

```
private String word;
```

මෙතනදී නම් **word** කියන එක හිස් නෙමේ ඇත්තෙන්ම **null** කියලා තමයි කියන්න වෙන්නේ...අර උඩ දෙවෙනියට තියෙන්නේ **String word=new String();** කියලා.ඒක නම් සෑහෙන්න දුලබ විදිහට තමයි භාවිතා කරන්නේ.ඇත්තෙන්ම භාවිතාවක් නැති තරම් වගේ.ඒ කියන්නේ **argument** නොමැතිව **constructor** භාවිතා කිරීම වගේ දෙයක්.නමුත් අනිත් අපි දක්වපු ක්‍රමය ඒ කියන්නේ අර ඊට ඉහලින් දක්වපු **String word=""**; විධිය නම් භාවිතා වෙනවා.මොකද **class** එකේ ඉහලින්ම **define** කරලා තියලා පස්සේ භාවිතා කරන්න පුළුවන්වෙන්නේ **re-assign** කරන්න එහෙමත්.

අනිත් programming language වල වගේම java වලත් මේ Strings වලට වැඩි දාත්න පුළුවන්. අකුරු ඔක්කොම capital කරන්න. simple කරන්න. අකුරු මාරු කරන්න. වෙනත් දිග බලන්න, වගේ ඒවා කරන විදිය අපි දැන් බලමු.

### Find length of the given word or string variable :

වෙනක වාක්‍යක දිග බලන්න තියන in build function එක තමා length කියන්නේ.

Syntax :

```
int object_name = string_variable_name.length();
```

Example :

```
public class string
{
    public static void main(String[] args)
    {
        String name = "R.W.D.J AMARASINGHE";

        int len = name.length(); // length function

        System.out.print("Length Of "+name+" = " +len);
    }
}
```

Expected Output :

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Length Of R.W.D.J AMARASINGHE = 19
Process finished with exit code 0
```

### Covert All lowercase latter to uppercase latter :

මේකටත් ජාවා වල in build function එකක් තියනවා.toUpperCase( ) කියන්නේ ඒකේ නම.

Syntax :

```
String object_name = String_variable_name.toUpperCase();
```

Example Code :

```
public class string
{
    public static void main(String[] args)
    {
        String name = "R.W.D.J Amarasinghe";

        int len = name.length(); // length function

        System.out.print("Length Of "+name+" = " +len+"\n");

        String toupper = name.toUpperCase(); //convert to uppercase

        System.out.print("Convert "+name+" to Uppercase = "+toupper+"\n");
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program
Length Of R.W.D.J Amarasinghe = 19
Convert R.W.D.J Amarasinghe to Uppercase = R.W.D.J AMARASINGHE

Process finished with exit code 0
```

Covert All uppercase latter to lowercase latter :

Syntax :

```
String object_name = String_variable_name.toLowerCase();
```

Example :

```
public class string
{
    public static void main(String[] args)
    {
        String name = "R.W.D.J Amarasinghe";

        int len = name.length(); // length function
```

```

System.out.print("Length Of "+name+" = " +len+"\n");

String toupper = name.toUpperCase(); //convert to uppercase

System.out.print("Convert "+name+" to Uppercase = "+toupper+"\n");

String to_lower = name.toLowerCase(); //convert to lowercase

System.out.print("Convert "+name+" to Lowercase = "+to_lower+"\n");
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program
Length Of R.W.D.J Amarasinghe = 19
Convert R.W.D.J Amarasinghe to Uppercase = R.W.D.J AMARASINGHE
Convert R.W.D.J Amarasinghe to Lowercase = r.w.d.j amarasinghe

```

## Find the index of String :

programming වලදී අපි ගන්න කරන්න පටන් ගන්නේ 0න්. ඉතින් අපිට අවශ්‍ය අකුරකට හිමි වෙලා තියෙන ස්ථානය බලන්න අපිට indexOf( ) කියන function එකෙන් බලන්න පුළුවන්.

Syntax :

```

int object_name = String_variable_name.indexOf("letter");

```

Example :

```

public class string
{
    public static void main(String[ ] args)
    {
        String name = "R.W.D.J Amarasinghe";

        int len = name.length(); // length function

        System.out.print("Length Of "+name+" = " +len+"\n");
    }
}

```

```

String toupper = name.toUpperCase(); //convert to uppercase

System.out.print("Convert "+name+" to Uppercase = "+toupper+"\n");

String to_lower = name.toLowerCase(); //convert to lowercase

System.out.print("Convert "+name+" to Lowercase = "+to_lower+"\n");

int index = name.indexOf("s"); //find the index

System.out.print("Index Of s = "+index+"\n");

    }
}

```

Expected Output :

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program
Length Of R.W.D.J Amarasinghe = 19
Convert R.W.D.J Amarasinghe to Uppercase = R.W.D.J AMARASINGHE
Convert R.W.D.J Amarasinghe to Lowercase = r.w.d.j amarasinghe
Index Of s = 13

Process finished with exit code 0

```

## Concatenating Strings With another String

The String class includes a method for concatenating two strings:

Syntax:

```
string1_name.concat(string2_name);
```

Example Code : Concatanation\_java.java

```

public class Concatanation_java
{
    public static void main(String[ ] args)
    {
        String word01 = "SLIIT";
    }
}

```



```

String word02 = " School Of Computing";

System.out.print(word01.concat(word02));
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
SLIIT School Of Computing
Process finished with exit code 0

```

Another way for concatenation :

```

public class Concatanation_java
{
    public static void main(String[] args)
    {
        String word01 = "SLIIT";

        String word02 = " School Of Computing";

        System.out.print(word01.concat(word02) + "\n"); // method 01

        String word3 = word01 + word02; //method 02

        System.out.print(word3);
    }
}

```

Expected output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
SLIIT School Of Computing
SLIIT School Of Computing
Process finished with exit code 0

```

## Java - String substring() Method :

This method has two variants and returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string or up to endIndex - 1, if the second argument is given.

Syntax :

```
string_variable_name.substring(beginIndex);
```

Example : Sub\_string.java

```
public class Sub_string
{
    public static void main(String[] args)
    {
        String word1 = "Java - String substring() Method";

        System.out.print(word1.length() + "\n"); // get length of word

        System.out.print(word1.substring(10)); // Create substring
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
32
ing substring() Method
Process finished with exit code 0
```

## Java - String compareTo() Method :

This method compares this String to another Object.

- Return Value :

The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

Syntax ;

```
String_variable_name1.compareTo(String_variable_name2);
```

Example : String\_compare.java

```
public class String_compare
{
    public static void main(String[] args)
    {
        String word1 = "SLIIT SCHOOL OF IT";

        String word2 = "SLIIT SCHOOL OF IT";

        String word3 = "SLIIT SCHOOL";

        String word4 = "SLIIT SCHOOL OF IT AND COMPUTER SCIENCE";

        String word5 = "UNIVERSITY KALANIYA";

        System.out.print("Compare " + word1 + " with " + word2 + " = " +
word1.compareTo(word2) + "\n");

        System.out.print("Compare " + word1 + " with " + word3 + " = " +
word1.compareTo(word3) + "\n");

        System.out.print("Compare " + word1 + " with " + word4 + " = " +
word1.compareTo(word4) + "\n");

        System.out.print("Compare " + word1 + " with " + word5 + " = " +
word1.compareTo(word5) + "\n");

    }
}
```

Expected Output:

```
Compare SLIIT SCHOOL OF IT with SLIIT SCHOOL OF IT = 0
Compare SLIIT SCHOOL OF IT with SLIIT SCHOOL = 6
Compare SLIIT SCHOOL OF IT with SLIIT SCHOOL OF IT AND COMPUTER SCIENCE = -21
Compare SLIIT SCHOOL OF IT with UNIVERSITY KALANIYA = -2
```

### charAt()

මෙතනදී අපි **method** එකේ වරහනට දෙන ඉලක්කමට අයිති **index** එක **String** එකෙන් බලලා අදාළ **character** එක **return** කරනවා අපිට.

### equalsIgnoreCase().

මේ කියන්නේ **String** දෙකක් දීලා සමානද කියලා බලන එකමයි. වෙනස් නියෙන්නේ මේ **method** එක දැම්මම වචන දෙකේ **case** එක අත ඇරලා තමයි බලන්නේ. ඒ කියන්නේ **simple capital** අත ඇරලා බලන්නේ.

### length()

**String** එකේ නියෙන **characters** ගාන අපිට දෙනවා

### replace()

මේකෙන් අපිට නියෙන **String** එකක **characters** වලට වෙනත් **characters** ගාණක් **replace** කරන්න පුළුවනි. අපි **method** එකේදී දෙන්න ඕනෙ අපිට **String** එකේ මොන අකුර, මොන අකුරට **replace** වෙන්න ඕනේද කියලා.

### substring()

අපිට මේ **method** එකෙන් පුළුවන් **String** එකක නියෙන කොටසක් උපුටා ගන්න.

### toLowerCase()

මේක නම් ඉතින් ලොකුවට කියන්න ඕනේ නෑනේ. අපි දෙන වචනෙක නියෙන **capital** අකුරු ටික **simple** වලට හරෝන එක තමයි මෙතනදී වෙන්නේ.

### toUpperCase()

මේකේදීත් අපි දෙන වචනෙක නියෙන **simple** අකුරු ටික **capital** වලට හරෝන එක වෙන්නේ.

### toString()

අපිට නියෙන කුමක් හෝ **value** එකක් **String** එකක් බවට හරවන්න මේ **method** එක උපයෝගී කර ගන්න පුළුවන්.

### trim()

මේයා භාවිතා වෙන්නේ අපිට නියෙන **String** එකක අවසානේටම නියෙන **whitespace** ඒ කියන්නේ අවසානේට හිදැස් නියපු තැන් ටික අහක් කරගන්න.

## More string methods :

Sr.No.	Method & Description
1	<u>char charAt(int index)</u> Returns the character at the specified index.
2	<u>int compareTo(Object o)</u> Compares this String to another Object.
3	<u>int compareTo(String anotherString)</u> Compares two strings lexicographically.
4	<u>int compareToIgnoreCase(String str)</u> Compares two strings lexicographically, ignoring case differences.
5	<u>String concat(String str)</u> Concatenates the specified string to the end of this string.
6	<u>boolean contentEquals(StringBuffer sb)</u> Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	<u>static String copyValueOf(char[] data)</u> Returns a String that represents the character sequence in the array specified.
8	<u>static String copyValueOf(char[] data, int offset, int count)</u> Returns a String that represents the character sequence in the array specified.
9	<u>boolean endsWith(String suffix)</u> Tests if this string ends with the specified suffix.
10	<u>boolean equals(Object anObject)</u> Compares this string to the specified object.
11	<u>boolean equalsIgnoreCase(String anotherString)</u> Compares this String to another String, ignoring case considerations.
12	<u>byte[] getBytes()</u> Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	<u>byte[] getBytes(String charsetName)</u>

	Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	<u><code>void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)</code></u> Copies characters from this string into the destination character array.
15	<u><code>int hashCode()</code></u> Returns a hash code for this string.
16	<u><code>int indexOf(int ch)</code></u> Returns the index within this string of the first occurrence of the specified character.
17	<u><code>int indexOf(int ch, int fromIndex)</code></u> Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
18	<u><code>int indexOf(String str)</code></u> Returns the index within this string of the first occurrence of the specified substring.
19	<u><code>int indexOf(String str, int fromIndex)</code></u> Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20	<u><code>String intern()</code></u> Returns a canonical representation for the string object.
21	<u><code>int lastIndexOf(int ch)</code></u> Returns the index within this string of the last occurrence of the specified character.
22	<u><code>int lastIndexOf(int ch, int fromIndex)</code></u> Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
23	<u><code>int lastIndexOf(String str)</code></u> Returns the index within this string of the rightmost occurrence of the specified substring.
24	<u><code>int lastIndexOf(String str, int fromIndex)</code></u> Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
25	<u><code>int length()</code></u>

	Returns the length of this string.
26	<u>boolean matches(String regex)</u> Tells whether or not this string matches the given regular expression.
27	<u>boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)</u> Tests if two string regions are equal.
28	<u>boolean regionMatches(int toffset, String other, int ooffset, int len)</u> Tests if two string regions are equal.
29	<u>String replace(char oldChar, char newChar)</u> Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
30	<u>String replaceAll(String regex, String replacement)</u> Replaces each substring of this string that matches the given regular expression with the given replacement.
31	<u>String replaceFirst(String regex, String replacement)</u> Replaces the first substring of this string that matches the given regular expression with the given replacement.
32	<u>String[] split(String regex)</u> Splits this string around matches of the given regular expression.
33	<u>String[] split(String regex, int limit)</u> Splits this string around matches of the given regular expression.
34	<u>boolean startsWith(String prefix)</u> Tests if this string starts with the specified prefix.
35	<u>boolean startsWith(String prefix, int toffset)</u> Tests if this string starts with the specified prefix beginning a specified index.
36	<u>CharSequence subSequence(int beginIndex, int endIndex)</u> Returns a new character sequence that is a subsequence of this sequence.
37	<u>String substring(int beginIndex)</u> Returns a new string that is a substring of this string.
38	<u>String substring(int beginIndex, int endIndex)</u> Returns a new string that is a substring of this string.

39	<u>char[] toCharArray()</u> Converts this string to a new character array.
40	<u>String toLowerCase()</u> Converts all of the characters in this String to lower case using the rules of the default locale.
41	<u>String toLowerCase(Locale locale)</u> Converts all of the characters in this String to lower case using the rules of the given Locale.
42	<u>String toString()</u> This object (which is already a string!) is itself returned.
43	<u>String toUpperCase()</u> Converts all of the characters in this String to upper case using the rules of the default locale.
44	<u>String toUpperCase(Locale locale)</u> Converts all of the characters in this String to upper case using the rules of the given Locale.
45	<u>String trim()</u> Returns a copy of the string, with leading and trailing whitespace omitted.
46	<u>static String valueOf(primitive data type x)</u> Returns the string representation of the passed data type argument.

More reading : [https://www.tutorialspoint.com/java/java\\_strings.htm](https://www.tutorialspoint.com/java/java_strings.htm)



## Chapter 7 : Java Operators

### Type Of Operators

01) අංක ගණිත මෙහෙවන (Arithmetic Operator)

02) සම්බන්ධක මෙහෙවන (Relational Operator)

03) තකීක මෙහෙවන (Logical Operator)






04) බිටුමය මෙහෙවන (Bit Operator)

05) ආධින මෙහෙවන ( Conditional Operator)

ඉහත ඒවා (Java) වල නියත මෙහෙවන (Operator) වේ. අපි ඉස්සල්ලාම බලමු ,

### 7.1-Arithmetic Operator

අංක ගණිත මෙහෙවන (Arithmetic Operator) ලෙස අපි එදිනෙදා ජීවිතයේ දී යොදාගන්නා එකතු කිරීම (+) , අඩු කිරීම(-) , බෙදීම (/), ගුණ කිරීම(\*) , ආදිය වේ. එනමුත් ජාවා වල දී අංක ගණිත මෙහෙවන (Arithmetic Operator) එක පිලිවලකට සිදුවේ ඒවා පහත පරිදිය.

Operator	Operation
	Addition
	Subtraction
	Multiplication
	Division
	Modulo Operation (Remainder after division)

Example : arithmetic.java

```
public class arithmetic
{
    public static void main(String[] args)
    {
        int number1 = 10;

        int number2 = 20;
```

```

int number3 = 50;

int result ;

result = number1 + number2; // Addition

System.out.print("Result = "+result+"\n");

result = number2 - number1; // Subtraction

System.out.print("Result = "+result+"\n");

result = number1 * number2; // Multiplication

System.out.print("Result = "+result+"\n");

result = number2 / number1; // Division

System.out.print("Result = "+result+"\n");

result = number3 % number2; // Modulo Operation (Remainder after division)

System.out.print("Result = "+result+"\n");
}
}

```

Expected Output :

```

Result = 30
Result = 10
Result = 200
Result = 2
Result = 10

```

Note :

ජාවා ප්‍රකාශනයක් ක්‍රියාත්මක වන්නේ දකුණු පස සිට වම් පසට වේ. එහි දී අපි විචල්‍යාන්ත අගයන්(value) ලබා දීමේදී මෙසේ ය

$x = 10 + 10 ;$

`int result = number + number02;`

### 7.1.1-Short -Circuit Arithmetic Operator

අංක ගණිත මෙහෙවන (Arithmetic Operator) කෙටිකර යොදා ගැනීම කෙටිමං අංක ගණිත මෙහෙවන ( Short -Circuit Arithmetic Operator) ලෙස දැක්විය හැකිය . ඒවා පහත පරිදිය.කෙටිමං අංක ගණිත මෙහෙවන ( Short -Circuit Arithmetic Operator) වලට අගයන් (value) ලබා දීමේදී මෙසේ ය.

**a=a+c** එකේ කෙටිමං අංක ගණිත මෙහෙවනය **a+=c ;**

**a=a-c** එකේ කෙටිමං අංක ගණිත මෙහෙවනය **a-=c ;**

**a=a/c** එකේ කෙටිමං අංක ගණිත මෙහෙවනය **a/=c ;**

**a=a+1** එකේ කෙටිමං අංක ගණිත මෙහෙවනය **a++ ;**

**a=a-1** එකේ කෙටිමං අංක ගණිත මෙහෙවනය **a- ;**

Example : short\_circuit\_arithmetic.java

```
public class short_circuit_arithmetic
{
    public static void main(String[] args)
    {
        int num01 = 5 , num02 = 10;

        num01 += num02;

        System.out.print("Result = "+num01+"\n");

        num01 -= num02;

        System.out.print("Result = "+num01+"\n");

        num01 ++;

        System.out.print("Result = "+num01+"\n");

        num02 --;

        System.out.print("Result = "+num02+"\n");
    }
}
```

Expected Output :

```

Result = 15
Result = 5
Result = 6
Result = 9

Process finished with exit code 0

```

## 7.2 -Relational Operator

Operator	Description	Example
<code>==</code>	Is Equal To	<code>3 == 5</code> returns <b>false</b>
<code>!=</code>	Not Equal To	<code>3 != 5</code> returns <b>true</b>
<code>&gt;</code>	Greater Than	<code>3 &gt; 5</code> returns <b>false</b>
<code>&lt;</code>	Less Than	<code>3 &lt; 5</code> returns <b>true</b>
<code>&gt;=</code>	Greater Than or Equal To	<code>3 &gt;= 5</code> returns <b>false</b>
<code>&lt;=</code>	Less Than or Equal To	<code>3 &lt;= 5</code> returns <b>false</b>

මෙවා විචලය කිහිපයක අගයන් (Variables Value) සංසන්දනය කිරීමට යොදා ගනී.

Example : relational\_operators.java

```

public class relational_operators
{
    public static void main(String[] args)
    {
        int number1 = 10 , number2 = 10 , number3 = 20;

        System.out.print(number1 == number2); // Is Equal Tols Equal To
    }
}

```

```

System.out.print("\n");

System.out.print(number1 != number2); // Not Equal To

System.out.print("\n");

System.out.print( number1 > number2); // Greater Than

System.out.print("\n");

System.out.print( number1 < number2); // Less Than

System.out.print("\n");

System.out.print( number1 <= number2); // Less Than or Equal To

System.out.print("\n");

System.out.print( number1 >= number2); // Greater Than or Equal To
}
}

```

Expected Output :

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
true
false
false
false
true
true
Process finished with exit code 0

```

### 7.3 – Logical Operator

මෙහි දී ප්‍රධාන තනික මෙහෙවන (Logical Operator) තුනක් භාවිත කරනවා , එවා AND, OR, NOT වේ.

තනික මෙහෙවන (Logical Operator) ක්‍රියා කරන්නේ කොහොමද ?

- 1) AND ( & ) -: මෙහි දී ආදාන (input) දෙකම සත්‍යය උව හෝත් ප්‍රතිදානය(Output) සත්‍යය වේ.
- 2) OR ( | ) -: මෙහි දී ආදාන (input) දෙකෙන් එකක් සත්‍යය උව හෝත් ප්‍රතිදානය(Output) සත්‍යය වේ.
- 3) NOT ( ! ) -: මෙහි දී ආදාන(input) එකක් සත්‍යය උව හෝත් ප්‍රතිදානය(Output) අසත්‍යය වේ.

Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	true only if both expression1 and expression2 are true
(Logical OR)	expression1    expression2	true if either expression1 or expression2 is true
! (Logical NOT)	!expression	true if expression is false and vice versa

Example :

```

public class logical_operator
{
    public static void main(String[] args)
    {
        int num1 = 5;

        int num2 = 10;

        //Logical AND

        System.out.print((num1 < num2) && (num2 > num1)); //true

        System.out.print("\n");

        System.out.print((num1 > num2) && (num2 > num1)); //false

        System.out.print("\n");

        //Logical OR

        System.out.print((num1 > num2) || (num2 > num1)); //true

        System.out.print("\n");
    }
}

```

```

System.out.print((num1 > num2) && (num2 < num1)); //false

System.out.print("\n");

//Logical NOT

System.out.print(!(num2 < num1)); //true

System.out.print("\n");

System.out.print(!(num2 > num1)); //false

System.out.print("\n");

// Complex One

System.out.print(! ((num2 > num1) && ((num1 > num2) || (num2 > num1)))); //false
}
}

```

Expected Output :

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
true
false
true
false
true
false
false
Process finished with exit code 0

```

Note :

කෙටිමං තකික මෙහෙවන (Short -Circuit Logical Operator) මොනාද බලමු. තකික මෙහෙවන (Logical Operator) කෙටිකර යොදා ගැනීම හා ක්‍රියාකරිත්වය ඉක්මන් කරගැනීම සඳහා කෙටිමං තකික මෙහෙවන ( Short -Circuit Logical Operator) ලෙස දැක්විය හැකිය . ඒවා පහත පරිදිය කෙටිමං අංක ගණිත මෙහෙවන ( Short -Circuit Arithmetic Operator) වලට අගයන් (value) ලබා දීමෙදි මෙසේ ය.

AND එකේ කෙටිමං අංක ගණින මෙහෙවනය &&

OR එකේ කෙටිමං අංක ගණින මෙහෙවනය ||

## 7.4 – Bitwise Operators

පරිගනකය තුළ දත්ත ගබඩා වන්නේ ද්වීමය ආකරයට වන නිසා ඒවා හැසිරවීමට බිට් මය මෙහෙවන (Bit Operator) යොදා ගනී.

Note :

Bitwise operators in Java are used to perform operations on individual bits. For example,

Bitwise complement Operation of 35

35 = 00100011 (In Binary)

~ 00100011

-----  
11011100 = 220 (In decimal)

Here, ~ is a bitwise operator. It inverts the value of each bit (0 to 1 and 1 to 0).

The various bitwise operators present in Java are:

Operators	Description	Use
&	Bitwise AND	op1 & op2
	Bitwise OR	op1   op2
^	Bitwise Exclusive OR	op1 ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	op1 << op2
>>	Bitwise Shift Right	op1 >> op2
>>>	Bitwise Shift Right zero fill	op1 >>> op2

Below example from : [greekforgeeks.com](http://greekforgeeks.com)



```

public class bitwise
{
    public static void main(String args[])
    {
        // Initial values
        int a = 5;
        int b = 7;

        // bitwise and
        // 0101 & 0111=0101 = 5
        System.out.println("a&b = " + (a & b));

        // bitwise or
        // 0101 | 0111=0111 = 7
        System.out.println("a|b = " + (a | b));

        // bitwise xor
        // 0101 ^ 0111=0010 = 2
        System.out.println("a^b = " + (a ^ b));

        // bitwise and
        // ~0101=1010
        // will give 2's complement of 1010 = -6
        System.out.println("~a = " + ~a);

        // can also be combined with
        // assignment operator to provide shorthand
        // assignment
        // a=a&b
        a &= b;
        System.out.println("a= " + a);

        // left shift operator
        System.out.println("a<<2 = " + (a << 2));

        // right shift operator
        System.out.println("b>>2 = " + (b >> 2));

        // unsigned right shift operator
        System.out.println("b>>>2 = " + (b >>> 2));
    }
}

```

Expected output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"  
a&b = 5  
a|b = 7  
a^b = 2  
~a = -6  
a= 5  
a<<2 = 20  
b>>2 = 1  
b>>>2 = 1
```

More about bit wise operators : <https://www.geeksforgeeks.org/bitwise-operators-in-java/>

## Chapter 8 : User Inputs and Outputs ( java Scanner Class)

### Java Scanner Class

අපි ජාවා වලදී **Scanner class** පාවිච්චි කරන්නේ මොනවා හරි අගයන් සහිත දේවල් අපේ භාවිතා කරන විචල්‍යයකට ඇතුළත් කරන්නයි. මේ **class** එක තියෙන්නේත් අපි කවුරුත් දන්න **java.util** කියන **package** එක ඇතුළේ. අපි මේ **class** එක පාවිච්චි කරන්න යනවා නම් අනිවාර්යෙන්ම ඊට කලියෙන් **java.util.Scanner class** එක **import** කරගන්න වෙනවා.. මේකෙදි පරිශීලකට ඒ කියන්නේ **user**'ට පුළුවන් වෙනවා විචල්‍ය වලට අවශ්‍ය කරන අගයන් යතුරු-පුවරුව භාවිතා කරලා එහෙමත් නැත්නම් වෙනත් ක්‍රමයකින් ලබා දෙන්න. අපි ඒ විධි ගැන ඉස්සරහට බලමු.. **Scanner Object** එකක් හදා ගන්නේ මෙහෙමයි.

```
import java.util.Scanner;

public class scanner
{
    public static void main(String[] args)
    {
        Scanner name = new Scanner(System.in);
    }
}
```

මේ **class** එකෙන් තියෙනවා අපි භාවිතා කරන **methods set** එකක්ම. ඒවත් දැනත් ඉන්න ඕනි ඉතින් **scanner** එකෙන් වැඩක් ගන්න නම්.

### Input Numerical Values

#### 1. nextInt()

මේ ඇවිල්ලා අපි **Scanner class** එකට දෙනවනම් **Integer / ඉලක්කම්** වලින් මොනවා හරි, ඒ වගේ **input** තියන් ඉදන් ආපිට දෙන **method** එක.

#### 2. nextFloat()

මේකත් ඇවිල්ලා කලින් වගේම **Scanner class** එකට දෙනවනම් **Float / දශම** වලින් මොනවා හරි, ඒ වගේ **input** තියන් ඉදන් ආපිට දෙන **method** එක.

#### 3. nextLong()

එකම **seen** එක ආයි ආයි කියන්න ඕන නැති නිසා මෙහෙම කියන්නම්. මේ **Long** ඒ කියන්නේ පොඩ්ඩක් ලොකු ඉලක්කම් දානකොට **input** තියන් ඉදන් ආපිට දෙන **method** එක.

#### 4. nextDouble()

මේ **Double** ඒ කියන්නේ පොඩ්ඩක් ලොකු දශම දානකොට **input** තියන් ඉදන් ආපිට දෙන **method** එක.

Example :

```
import java.util.Scanner;

public class input_numerics
{
    public static void main(String[] args)
    {
        Scanner number = new Scanner(System.in);

        System.out.print("Enter Integer Value : "); //input integer value

        int num = number.nextInt();

        System.out.print("Integer Value : "+num+"\n");

        System.out.print("Enter Float Value : "); //input float value

        float num2 = number.nextFloat();

        System.out.print("Float Value : "+ num2+"\n");

        System.out.print("Enter Long Integer Value : "); //input long value

        long num3 = number.nextLong();

        System.out.print("Long Value : "+ num3+"\n");

        System.out.print("Enter Double Value : "); //input double values

        double num4 = number.nextDouble();

        System.out.print("Long Value : "+ num4+"\n");

    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Enter Integer Value : 343
Integer Value : 343
Enter Float Value : 456.77
Float Value : 456.77
Enter Long Integer Value : 345678
Long Value : 345678
Enter Double Value : 456.34567890567890
Long Value : 456.3456789056789
```

## Input String Values

### 1. next()

මෙක පාවිච්චි වෙනකොට අර කලින් වගේම තමයි. හැබැයි අපි දෙන වචන සෙට් එකක් තිබ්බොත් ගන්නේ පළවෙනි වචනේ. ඒක ඉලක්කමක්ද අකුරු ටිකක්ද කියන එක අදාල නෑ. ඒ කියන්නේ **space** එකක් හම්බවෙනකන් බලලා **space** එකට කලින් ටික ගන්නවා. අරන් අපිට **return** කරනවා

### 2. nextLine()

මෙතනදි කලින් කතාවමයි. වෙනස ඇවිල්ලා **space** නෙමේ කෙලින්ම මුළු පේළියම ගන්නවා. අරන් අපිට **return** කරනවා.

Example:

```
import java.util.Scanner;

public class input_strings
{
    public static void main(String[] args)
    {
        Scanner name = new Scanner(System.in);

        System.out.print("Your Name : ");

        String name2 = name.next();

        System.out.print("My Name Is : "+name2+"\n");

        //*****

        Scanner name4 = new Scanner(System.in);
```

```

System.out.print("Your Name : ");

String name3 = name4.nextLine();

System.out.print("My Name Is : "+name3+"\n");
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Your Name : deshan jayashanka
My Name Is : deshan
Your Name : deshan jayashanka
My Name Is : deshan jayashanka

```

අපිට මේ **Class** එක පාවිච්චි කරලා **Inputs Store** කරන් ඉඳලා ගන්නවා වෙනුවට මේකේ එහෙම ඊලඟට ගන්න තියෙන **Input** එක මොකක්ද කියලා ප්‍රකාශනයක් අනුමාන කරවා ගන්නත් පුළුවනි.

### 1. hasNextInt()

ඊලඟට ගන්න තියෙන්නේ **int** අගයක් ඒ කියන්නේ පූර්ණ ඉලක්කම් සහිත අගයක් නම් **true** වෙනවා. නැත්නම් **false** වෙනවා.

### 2. hasNextDouble()

ඊලඟට ගන්න තියෙන්නේ **boolean** අගයක් ඒ කියන්නේ දශම අගයක් නම් **true** වෙනවා. නැත්නම් **false** වෙනවා.

### 3. hasNext()

ඊලඟට ගන්න මොනවා හරි තියේ නම් **true** වෙනවා. නැත්නම් **false** වෙනවා.

### 4. hasNextLine()

ඊලඟට එහෙමපිටින්ම **line** ඊකක් තියෙනම් **true** වෙනවා. නැත්නම් **false** වෙනවා.

## Chapter 9 : Control Statements in Java

Control Statements use in java:

- Decision Making Statements
- Simple if statement
- if-else statement
- Nested if statement
- Switch statement
- Looping statements
- While
- Do-while
- For
- For-Each
- Branching statements
- Break
- Continue

### If – else Statements

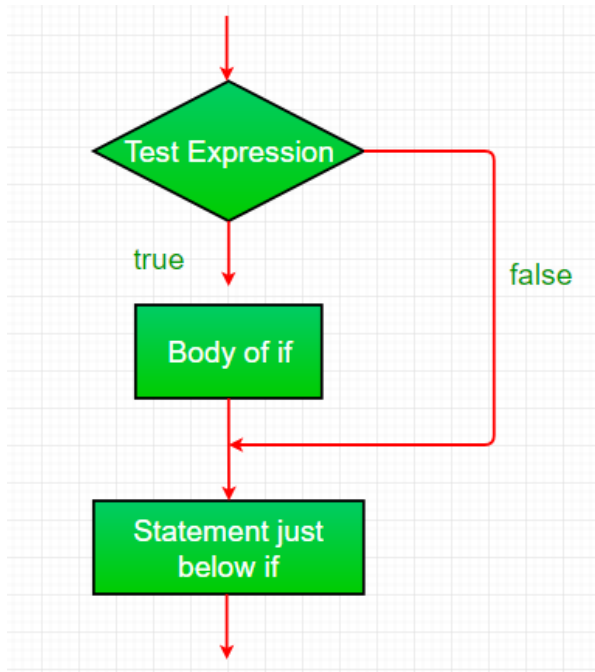
මෙම තීරණ ගැනීම සඳහා භූලීයානු මෙහෙවන **if** පාලන වයුහය තුළ යොදා ගනිමින් මෙම **IF\_else** පාලන ප්‍රකාශ ( **IF\_else Control Statements**) ක්‍රියාත්මක වේ. **programming** වලදී අපිට විවිධ තෝරා ගැනීම් සිදු කරන්න වෙනවා. උදාහරණයක් විදියට පරිශීලකයා ඇතුළත් කරන සංඛ්‍යාව පහට වැඩිද කියලා බලන්න නම් අපිට යම් කිසි තෝරා ගැනීමක් කරන්න වෙනවා. මේ සිද්ධියට අපි වරණ පාලනය කියලා කියනවා. **if, else** යන විධාන දෙක අපි මේ වරණ පාලනයට භාවිතා කරනවා.

#### Type of IF Control Statements

- #1) IF Statements
- #2) if -else Statements
- #3) if -else -if Statements
- #4) Nested if Statements

## If Statement:

if පාලන වස්තු ( IF\_Control Statements) තුළ ඇති තාක්ෂික ප්‍රකාශනය හි අගය සත්‍ය නම් , if පාලන ප්‍රකාශනය තුළ ඇති අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ.

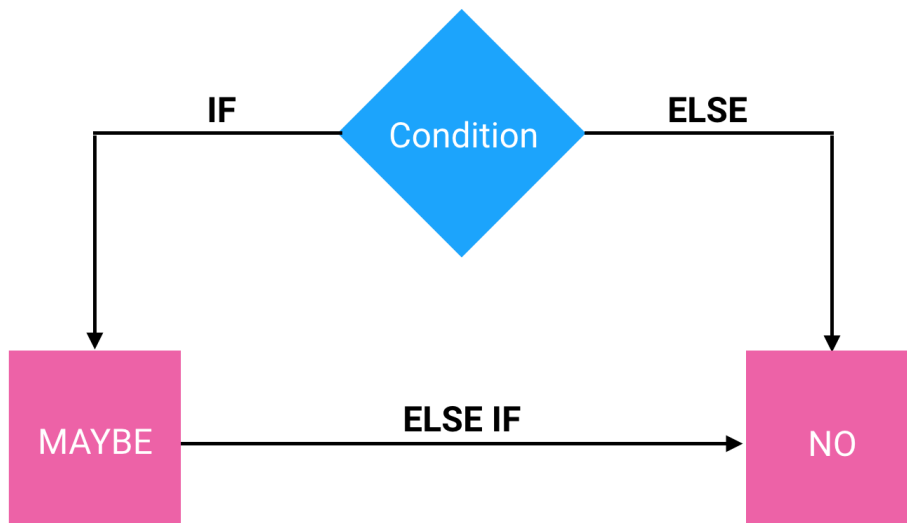


If architecture :

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```



## if – else Statement :



IF\_ELSE පාලන වස්තු ( IF\_ELSE Control Statements) තුළ ඇති තාක්ෂණික ප්‍රකාශනය හි අගය සත්‍ය නම් , if පාලන ප්‍රකාශනය තුළ ඇති අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ. එසේ නොමැති නම් else ප්‍රකාශනය තුළ ඇති අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ.

Syntax :

If else architecture :

```
public static void main(String args[ ])  
{  
    if (condtion)  
    {  
        //Statement  
    }  
    else  
    {  
        //Statement  
    }  
}
```

Example :

Write a java program to check whether a number is even or odd.

```
import java.util.Scanner;

public class pro2
{
    public static void main(String[] args)
    {
        Scanner innum = new Scanner(System.in);

        System.out.print("Enter your number :");

        int number = innum.nextInt();

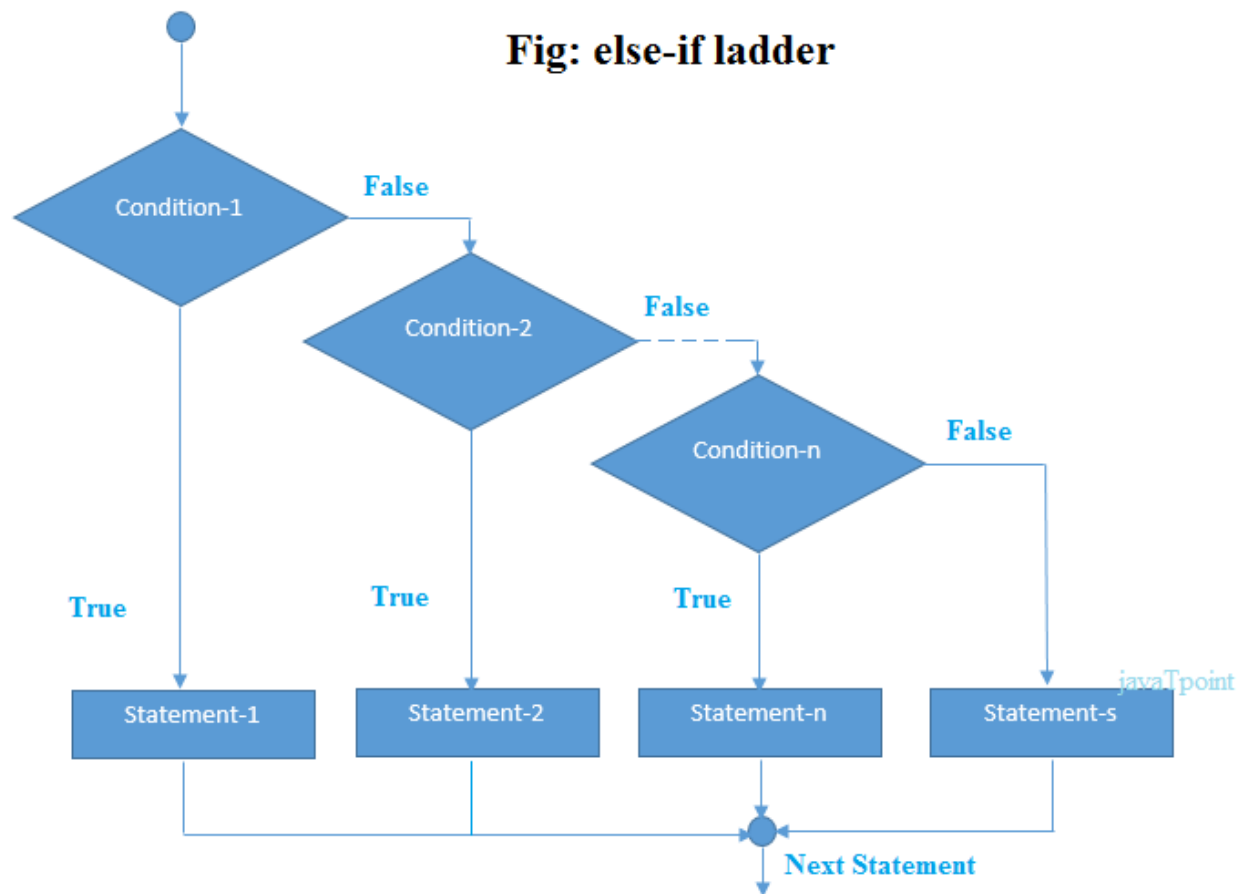
        int reminder = number % 2;

        if (reminder == 0)
        {
            System.out.print("Even Number");
        }
        else
        {
            System.out.print("Odd Number");
        }
    }
}
```

Expected Output:

```
PS C:\Users\Jayashanka Deshan> & 'c:\Users\Jayashanka Deshan> java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 pro2
Enter your number :69
Odd Number
PS C:\Users\Jayashanka Deshan> 
```

## Else if Statement :



පලමු IF\_ පාලන ව්‍යුහය ( IF Control Statements) තුළ ඇති තාක්ෂික ප්‍රකාශනය හි අගය සත්‍ය නම් , ඒ තුළ ඇති අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ එසේ නොමැති නම් else if පාලන ප්‍රකාශනය තුළ ඇති තාක්ෂික ප්‍රකාශනය හි අගය සත්‍ය නම් එහි අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ. එසේ නොමැති නම් else ප්‍රකාශනය තුළ ඇති අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ.

Else if architecture:

```
public static void main(String args[ ])  
{  
    if (condtion){  
        //Statement  
    }  
    else if{  
        //Statement  
    }  
}
```

```
else{  
    //Statement  
}  
}
```

Example :

Write a Java program to check whether a number is negative, positive or zero.

```
import java.util.Scanner;  
  
public class pro3  
{  
    public static void main(String[] args)  
    {  
        Scanner proin = new Scanner(System.in);  
  
        System.out.print("Enter your Number : ");  
  
        int number = proin.nextInt();  
  
        if(number == 0)  
        {  
            System.out.print("You Entered Zero");  
        }  
        else if (number > 0)  
        {  
            System.out.print("You Entered positive number");  
        }  
        else  
        {  
            System.out.print("You Entered Negative Number");  
        }  
    }  
}
```

Expected Output:

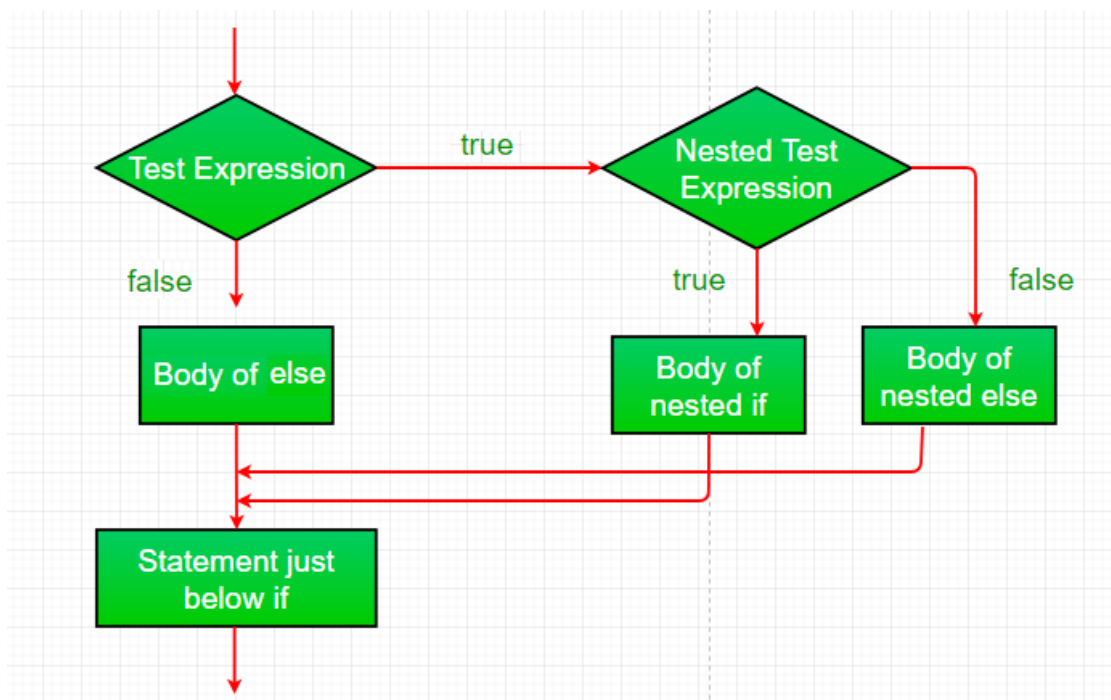
```

Enter your Number : 8
You Entered positive number
PS C:\Users\Jayashanka Deshan> & 'c:\Users\
' '-XX:+ShowCodeDetailsInExceptionMessages'
Enter your Number : 0
You Entered Zero
PS C:\Users\Jayashanka Deshan> & 'c:\Users\
' '-XX:+ShowCodeDetailsInExceptionMessages'
Enter your Number : 7
You Entered positive number
PS C:\Users\Jayashanka Deshan> & 'c:\Users\
' '-XX:+ShowCodeDetailsInExceptionMessages'
Enter your Number : -89
You Entered Negative Number
PS C:\Users\Jayashanka Deshan>

```

## Nested if Statement :

පලමු IF\_ පාලන වයුහය ( IF Control Statements) තුළ ඇති තාක්ෂික ප්‍රකාශනය හි අගය සත්‍ය නම් , ඒ තුළ ඇති අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ එසේ නොමැති නම් තවත් if යොදා ඒ if පාලන ප්‍රකාශනය තුළ ඇති තාක්ෂික ප්‍රකාශනය හි අගය සත්‍ය නම් එහි අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ. එසේ නොමැති නම් තවත් if යොදා ඒ පාලන ප්‍රකාශනය තුළ ඇති තාක්ෂික ප්‍රකාශනය හි අගය සත්‍ය නම් එහි අනිකුත් විධාන සහ ප්‍රකාශන ක්‍රියා කරනු ලැබේ.(ඉහත විදිහට ඔබට අවශ්‍යය ප්‍රමාණයට if( condition ) යොදා ගත හැකිය . )



Nested if architecture:

```

if(Condition1)
{
    /// Executes when condition 1 is true
    if(Condition2)
    {
        /// Executes when condition 2 is true
    }
}

```

Example :

```

public class nested_if_demo
{
    public static void main(String args[ ])
    {
        int i = 10;

        if (i == 10)
        {
            // First if statement
            if (i < 20)
                System.out.println("i is smaller than 15");

            // Nested - if statement
            // Will only be executed if statement above
            // it is true
            if (i < 12)
                System.out.println("i is smaller than 12 too");
            else
                System.out.println("i is greater than 15");
        }
    }
}

```

Expected Output:

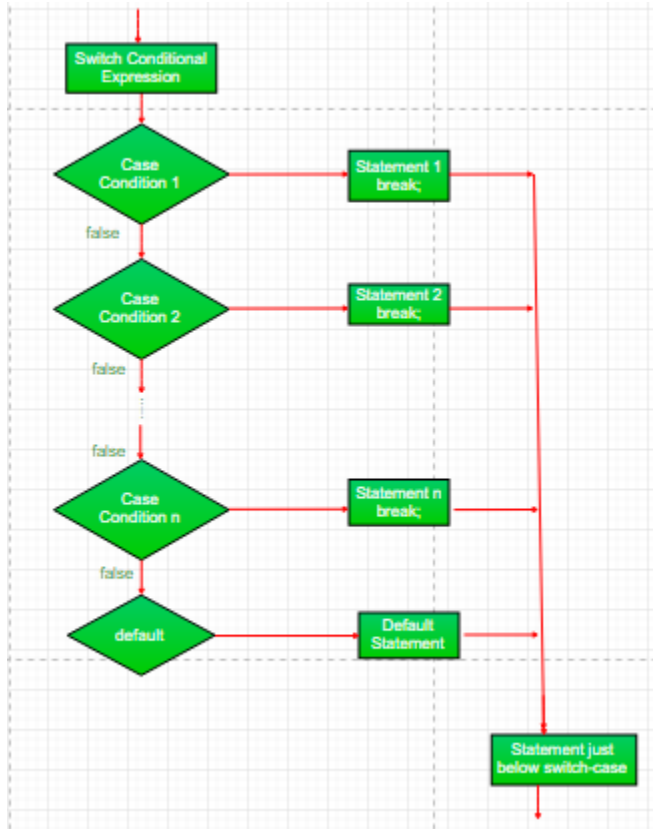
```

i is smaller than 15
i is smaller than 12 too
PS C:\Users\Jayashanka Deshan> 

```

## Switch Case Control Statement :

switch-case එක if පාලන ප්‍රකාශනයට (If Control Statements) බොහෝ දුරට සාමාන්‍ය වෙනවා. මේ switch-case ආරම්භ වෙන්නේ switch(X) ලෙස ,මෙකේ X කියන්නේ විචල්‍යයක්. මෙහි දී විචල්‍යයන් සඳහා (byte),(short),(int), (Char) හා විනා කළ හැකිය . (long),(float),(double),(boolean ) හා විනා කළ නොහැකිය.



**SWITCH(X)** ආරම්භයට පසුව සහල වරහන් තුළ අනෙකුත් විධානයන් යෙදිය යුතුය . මෙම X විචල්‍යයේ අගය මත මේම විධානයන් හා ජරකාශන කිරීමක් වන ස්ථානය තීරනය වෙන්නේ. මෙහි විවිධ ආරම්භක ස්ථාන තියන්න පුළුවන් . එම ස්ථාන **CASE** සමග X විචල්‍යයට ගන්න පුළුවන් අගය ලබාදීම මගින් සලකුණු කරයි. මෙහි X අගයන් සමාන විට විධානයන් හා ජරකාශන කිරීමක් වන්නේ **case default** ස්ථානයෙනි .

Switch case architecture:

```
switch (Expression)
{
    case value:
        statement1;
        break;
    default:
        statement2;
        break;
}
```

Example :

```
public class Switch_Case
{
    public static void main(String args [ ])
    {
        int number = 100;

        switch (number)
        {
            case 20:
                System.out.print("Number is 20");
                break;
            case 200:
                System.out.print("Number is 100");
                break;
            default:
                System.out.print("Any Other Number");
        }
    }
}
```



```
}  
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"  
Any Other Number  
Process finished with exit code 0
```

**SWICH CASE** පාලන ප්‍රකාශනයේ ක්‍රියාකාරීත්වය ( Switch Control Statements)

විවිධාකාර ආකාර ක්‍රමලේඛනය ලිවීමට යොදා ගන්න පුලුවන් ඒ සඳහා Switch CASE වගී කිහිපයක් තියනවා ඒවා පහත පරිදි වේ .

1)java Switch Statement is fall-through

2)java Switch Statement with String

3)java Nested Switch Statement

4)java Enum in Switch Statement

5)java Wrapper in Switch Statement

Short Introductions :

java Switch Statement is fall-through

මෙම වගීයේ ප්‍රකාශන( Switch Statement is fall-through) මගින් සියල්ලම නැවතීම (Break statement) නොමැතිව සියල්ලම ක්‍රියාත්මක වෙනවා .

Example :

```
public class Switch_Case_02  
{  
    public static void main(String args[])  
    {  
        int number =30;  
  
        switch(number)  
        {  
            case 10:  
                System.out.println("10");  
            //case 20:  
            //    System.out.println("20");  
            //case 30:  
            //    System.out.println("30");  
            //case 40:  
            //    System.out.println("40");  
            //case 50:  
            //    System.out.println("50");  
            //case 60:  
            //    System.out.println("60");  
            //case 70:  
            //    System.out.println("70");  
            //case 80:  
            //    System.out.println("80");  
            //case 90:  
            //    System.out.println("90");  
            //case 100:  
            //    System.out.println("100");  
            //default:  
            //    System.out.println("Default");  
        }  
    }  
}
```

```

    case 20:
        System.out.println("20");

    case 25:
        System.out.println("25");

    case 30:
        System.out.println("35");

    default:
        System.out.println("Not in number 10,20,,25,35");
}
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
35
Not in number 10,20,,25,35

Process finished with exit code 0
|

```

### Java Switch Statement with String

මෙම වග්ගයේ ප්‍රකශන( Switch Statement with String ) මගින් String යොදා ගනිමින් තමයි වැඩ කරන්නේ එහෙත් මේම පහසුකම ජාවා SE 7 සංස්කරණයේ සිට අලුතින් හඳුන්වා දී ඇත .

```

public class Switch_Case_03
{
    public static void main (String args[])
    {
        String a ="Lara Croft";

        int place =0;

        switch(a)

```

```

{
    case "Jakson":
        place=1;
        break;
    case "Sentourian":
        place=3;
        break;
    case "Lara Croft":
        place=2;
        break;
    default:
        place=0;
        break;
}
System.out.print("your place is "+place);
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
your place is 2
Process finished with exit code 0
|

```

### Java Nested Switch Statement

නැවත නැවත කිහිපවරක්ම එක් Switch ප්‍රකාශනයක් ඇතුළත තවත් Switch ප්‍රකාශන යොදාගැනීම **Nested Switch Statement** ලෙස දැක්වීමට පුළුවන්



```
}  
  
}  
}  
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"  
INDEX : AA1689  
NAME : DESHAN JAYASHANKA  
PLACE : 10  
Process finished with exit code 0
```

Example Questions and and Answers:

Examples and Answers :

01. Write a java program to check whether a number is even or odd.

```
import java.util.Scanner;  
  
public class pro2  
{  
    public static void main(String[] args)  
    {  
        Scanner innum = new Scanner(System.in);  
  
        System.out.print("Enter your number :");  
  
        int number = innum.nextInt();  
  
        int reminder = number % 2;  
  
        if (reminder == 0)  
        {  
            System.out.print("Even Number");  
        }  
    }  
}
```

```

        else
        {
            System.out.print("Odd Number");
        }
    }
}

```

02. Write a Java program to check whether a number is negative, positive or zero.

```

import java.util.Scanner;

public class pro3
{
    public static void main(String[] args)
    {
        Scanner proin = new Scanner(System.in);

        System.out.print("Enter your Number : ");

        int number = proin.nextInt();

        if(number == 0)
        {
            System.out.print("You Entered Zero");
        }
        else if (number > 0)
        {
            System.out.print("You Entered positive number");
        }
        else
        {
            System.out.print("You Entered Negative Number");
        }
    }
}

```

03. Write a java program to find maximum between two numbers.

```

import java.util.Scanner;

import javax.lang.model.util.ElementScanner14;

public class pro4 {

    public static void main(String[] args)
    {
        Scanner pro = new Scanner(System.in);

        System.out.print("Enter First Number : ");

        int num1 = pro.nextInt();

        Scanner pro2 = new Scanner(System.in);

        System.out.print("Enter Second Number : ");

        int num2 = pro2.nextInt();

        if (num1 == num2)
        {
            System.out.print("Both Numbers are equal");
        }
        else if(num1 > num2)
        {
            System.out.print(num1+ " grater than " +num2);
        }
        else
        {
            System.out.print(num2+" grater than " +num1);
        }
    }
}

```

04. Write a java program to input electricity unit charges and calculate total electricity bill according to the given condition

For first 50 units Rs. 0.50/unit  
 For next 100 units Rs. 0.75/unit  
 For next 100 units Rs. 1.20/unit  
 For unit above 250 Rs. 1.50/unit

```

import java.util.Scanner;

public class pro5
{
    public static void main(String[] args)
    {
        Scanner profive = new Scanner(System.in);

        System.out.print("Enter Your units Number : ");

        double number = profive.nextDouble();

        if (number <= 50)
        {
            double prof = number * 0.50;

            System.out.print("Your Monthly Payment is "+prof+" for "+number+" Units ");
        }
        else if(number<=150)
        {
            double extra = number - 50;

            double prof = (extra * 0.75) + (50 * 0.50);

            System.out.print("Your Monthly Payment is "+prof+" for "+number+" Units ");
        }
        else if(number<=250)
        {
            double extra = number -150;

            double prof = (extra * 1.20)+(100 * 0.75)+(50 *0.50);

            System.out.print("Your Monthly Payment is "+prof+" for "+number+" Units ");
        }
        else
        {
            double extra = number - 250;

            double prof = (extra * 1.50)+(100 * 1.20)+(100 * 0.75)+(50 * 0.50);

            System.out.print("Your Monthly Payment is "+prof+" for "+number+" Units ");
        }
    }
}

```



```
}  
  
}  
  
}
```

05. Write a java program to check whether a character is uppercase or lowercase alphabet

```
import java.util.Scanner;  
  
public class pro6  
{  
    public static void main(final String[] args)  
    {  
        final Scanner alphabet = new Scanner(System.in);  
  
        System.out.print("Enter Character You Want To Scan : ");  
  
        final char inputchar = alphabet.next().charAt(0);  
  
        // Verification inputs  
  
        if((inputchar>='A'&& inputchar<='Z')||(inputchar>='a'&& inputchar<='z'))  
        {  
            System.out.print(inputchar+" Valid Character ");  
  
        }  
        else  
        {  
            System.out.print(inputchar+" You Entered Invalid Character ");  
  
            System.exit (0); // terminate system //  
  
        }  
  
        // main process //  
  
        if(inputchar>='A'&& inputchar<='Z')  
        {  
            System.out.print(inputchar+" is Uppercase Character");  
        }  
        else if (inputchar>='a'&& inputchar<='z')  
        {
```

```

        System.out.print(inputchar+" is Lowercase Character");
    }
}
}

```

## 06.JAVA Program to Make a Simple Calculator to Add, Subtract, Multiply or Divide

```

// JAVA Program to Make a Simple Calculator to Add, Subtract, Multiply or Divide
import java.util.Scanner;

public class pro7
{
    public static void main(String args[])
    {
        Scanner num1 = new Scanner(System.in);

        System.out.print("Enter First Number :");

        float n1 = num1.nextInt();

        Scanner num2 = new Scanner(System.in);

        System.out.print("Enter Second Number :");

        float n2 = num2.nextInt();

        System.out.print("Addition : +\n");

        System.out.print("Subtract : -\n");

        System.out.print("Multiply : *\n");

        System.out.print("Devide   : /\n");

        Scanner ope = new Scanner(System.in);

        System.out.print("Choose a Operator From Above list :");

        char intchar = ope.next().charAt(0);

        if (intchar=='+')

```

```

    {
        float ans = n1 + n2;

        System.out.print("Your Answer "+n1+" + "+n2+" = "+ans);
    }

    else if (intchar=='-')
    {
        float ans = n1 - n2;

        System.out.print("Your Answer "+n1+" - "+n2+" = "+ans);
    }

    else if (intchar=='*')
    {
        float ans = n1 * n2;

        System.out.print("Your Answer "+n1+" x "+n2+" = "+ans);
    }

    else if (intchar=='/')
    {
        float ans = n1 / n2;

        System.out.print("Your Answer "+n1+" / "+n2+" = "+ans);
    }

}

}

```

07. Write a java program to convert days into years, weeks and days.

//Write a java program to convert days into years, weeks and days.

```

import java.util.Scanner;

public class pro9
{
    public static void main(String args[])
    {

```

```

Scanner IN = new Scanner(System.in) ;

System.out.print("Enter Number Of Days: ") ;

int days = IN.nextInt() ;

if (days < 7)
{
    System.out.print(days+" :DAYS");
}
else if (days >7 && days < 30)
{
    int exd=(days % 7);

    int weeks = (days - exd)/7;

    System.out.print(weeks+" :WEEKS | "+exd+" :DAYS");
}
else if (days > 30 && days < 365)
{
    int exm = (days % 30);

    int months = (days-exm)/30;

    days = (exm % 7);

    int weeks = (exm - days)/7;

    System.out.print(months+" :MONTH | "+weeks+ " :WEEKS | "+days+ " :DAYS");
}
else
{
    int exx = (days % 365);

    int years = (days-exx)/365;

    int exm = exx % 30;

    int months = (exx-exm)/30;

    int exd = exm % 7;

    int weeks =(exm-exd)/7;
}

```

```

        System.out.print(years+ " :YEARS | " +months+ " :MONTHS | "+weeks+ " :WEEKS | "+exd+" :DAYS");

    }

}

}

```

08. Write a java program to find maximum between three numbers

```

//Write a java program to find maximum between three numbers

import java.util.Scanner;

public class pro10
{
    public static void main(String[] args)
    {
        Scanner n1 = new Scanner(System.in);

        System.out.print("Enter Number One : ");

        int num1 = n1.nextInt();

        Scanner n2 = new Scanner(System.in);

        System.out.print("Enter Number One : ");

        int num2 = n1.nextInt();

        Scanner n3 = new Scanner(System.in);

        System.out.print("Enter Number One : ");

        int num3 = n1.nextInt();

        if(num1 < num2)
        {
            if(num2 < num3)

```

```

        {
            System.out.print(num3+" Higher Than "+num2+" and "+num1);
        }
        else
        {
            System.out.print(num2+" Higher Than "+num3+" and "+num1);
        }
    }
    else
    {
        if(num1 < num3)
        {
            System.out.print(num3+" Higher Than "+num1+" and "+num2);
        }
        else
        {
            System.out.print(num1+" Higher Than "+num3+" and "+num2);
        }
    }
}
}

```

09. Write a Java program to input any alphabet and check whether it is vowel or consonant

Methods:

01. Use ASCII Table

02. Use Normal Method

```

//Write a Java program to input any alphabet and check whether it is vowel or consonant

// ASCII method

import java.util.Scanner;

public class pro12
{
    public static void main(String[] args)
    {

```

```

Scanner alph = new Scanner(System.in);

System.out.print("Enter Character in Alphabet : ");

char alphabet = alph.next().charAt(0);

// Verification inputs

if((alphabet>='A'&& alphabet<='Z')||(alphabet>='a'&& alphabet<='z'))
{
    System.out.print(alphabet+" Valid Character \n");
}
else
{
    System.out.print(alphabet+" You Enterd Invalid Character \n");

    System.exit (0); // terminate system //

}

//main process//

17 if(alphabet==97||alphabet==101||alphabet==105||alphabet==111||alphabet==1
    ||alphabet==65||alphabet==69||alphabet==73||alphabet==79||alphabet==85)
    {
        System.out.print(alphabet+" is Vowel");
    }

    else
    {
        System.out.print(alphabet+" is Consonant");
    }

}

}

```

```

//Write a Java program to input any alphabet and check whether it is vowel or con
sonant

```

```

//Normal method

import java.util.Scanner;

public class pro11
{
    public static void main(String[] args)
    {
        Scanner alph = new Scanner(System.in);

        System.out.print("Enter Character in Alphabet : ");

        char alphabet = alph.next().charAt(0);

        // Verification inputs

        if((alphabet>='A'&& alphabet<='Z')||(alphabet>='a'&& alphabet<='z
'))
        {
            System.out.print(alphabet+" Valid Character \n");
        }
        else
        {
            System.out.print(alphabet+" You Enterd Invalid Character \n")
;

            System.exit (0); // terminate system //

        }

        //main process//

        if(alphabet=='a' || alphabet=='e' || alphabet=='i' || alphabet=='o' || alphabet=='u'
|| alphabet=='A' || alphabet=='E' || alphabet=='I' || alphabet=='O' || alphabet=='
U')
        {
            System.out.print(alphabet+" is Vowel");
        }
        else

```



```
    {  
        System.out.print(alphabet+" is Consonant");  
    }  
  
}
```

## Chapter 10 : LOOPS

### Introduction

පුනර්කරණයක් කියන්නේ නැවත නැවත යෙදීමක්. උදාහරණයක් ගත්තොත් අපි එකේ සිට දහයට **print** එකක් ගන්න ඕනි නම් එකේ ඉඳලා දහයට එක එක **variable** දීලා ඒක **print** කර ගන්නවට වැඩිය අපි පුනර්කරණයක් භාවිතා කරලා ඒක **print** කර ගන්න එක පහසුයි.

### MAIN LOOPS TYPES :

1. WHILE LOOPS
2. FOR LOOPS
3. DO WHILE
4. FOR EACH LOOP

### Usages Of Loops:

- කිසියම් **array** එකක් තුළ ඇති **elements** (අවයව) එකින් එක පිළිවෙලින් පරීක්ෂා කිරීම.
- විවිධ ඇල්ගොරිතම (algorithm) නිර්මාණයේදී.
- කිසියම් දත්ත සමූහයක් විශ්ලේෂණය කිරීමේදී

### While Loop

**while loop** එකේදී කරන්නේ යම් තොරතුරක් සත්‍ය වන තාක් පුනර්කරණය ක්‍රියාත්මක කිරීමයි. ඒ කියන්නේ ඉලක්කම පහට වඩා අඩු වෙනකන් ප්‍රින්ට් කරන්න කියනවා වගේ එකක්. පහට වඩා වැඩි වුණ ගමන් පුනර්කරණය නවතිනවා.

### Note :

මෙහිදී **expression** එක **true** ලෙස පවතින තාක් **loop** එක ක්‍රියාත්මක වීම සිදුවේ (නැවත නැවත සිදුවේ). අපි දැන් **while loop** එක ක්‍රියාත්මක වන අන්දම විස්තරාත්මකව බලමු.

මෙහිදී **expression** එකෙහි **boolean** අගය පරීක්ෂා කිරීම සිදුවනවා. එය **false** නම් **while** එක ඇතුළේ ඇති **statement** එක **execute** කිරීම සිදු නොකර කෙලින්ම **loop** එකෙන් පිටතට යාම සිදු කරනවා. එය **true** නම් **while loop** එක ඇතුළේ ඇති **statement execute** කර අවසානයේ නැවත **expression** එක පරීක්ෂා කරනවා. මේ ආකාරයට දිගින් දිගටම මෙය සිදුවනවා. සාමාන්‍යයෙන් **while loop** එකක් යොදාගන්නේ **loop** එක කොපමණ වාර ගණනක් ක්‍රියාත්මක විය යුතුද යන්න කලින් අනුමාන කළ නොහැකි අවස්ථා වලදීය.

Syntax :

```
while (Expression)
{
    // do something here
}
```

Example :

```
public class example_01
{
    public static void main(String args[ ])
    {
        int number = 1;

        while (number < 6)
        {
            System.out.println(number);

            number++;
        }
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
1
2
3
4
5
```

Explain Above Code :

උඩ තියන කෝඩ් එකෙන් වෙන්වෙන්නේ **number** කියලා **variable** එකක් අරන් ඒක පුනර්කරණයක් ඇතුළේ එක එක වැඩි කරනවා. **number variable** එක හයට වැඩි වුන ගමන්ම පුනර්කරණය නතර වෙනවා.

**number++** කියන එක වෙනුවට අපිට කියන **number = number + 1** එකත් පාවිච්චි කරන්න පුළුවන්. ඒකෙනුත් වෙන්වෙන්න **number** විචල්‍යයට එකක් එකතු වෙන එකමයි.

## FOR LOOP

සාමාන්‍යයෙන් **for loop** එකක් භාවිතා කරන්නේ **loop** එක ක්‍රියාත්මක වන වාර ගණන දන්නා විටදීය. පුනර්කරණ අතරින් වඩාත් පහසුම පුනර්කරණය වෙන්නේ **for** පුනර්කරණය.ඇත්තටම මේකත් **while** පුනර්කරණය වගේමයි.පොඩි වෙනසක් තියෙන්නේ.අපි පහළ උදාහරණෙන් එක බලමු.

Syntax:

```
for(initialization ; Expression ; Update_Statement )
{
    //do something
}
```

Example :

```
public class example_10_02
{
    public static void main(String args[])
    {
        for (int i = 1 ; i < 6 ; i = i + 1)
        {
            System.out.println(i);
        }
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
1
2
3
4
5
```

EXPLAIN ABOVE CODE :

ඔයාලට පේනවා ඇති **while loop** එකේදී අපි **lines** තුනක් අරන් ලියපු **condition** එක **for loop** එකේදී වරහන් ඇතුළේ ලියනවා.

## DO WHILE LOOP

**do while** පුනර්කරණයක් ඇත්නම්ම **while** පුනර්කරණය වගේමයි. පොඩ් වෙනසක් තියෙන්නේ. මෙහිදී **expression** එක පරීක්ෂා කිරීම සිදුකරන්නේ **statements execute** කිරීමෙන් පසුවයි. අපි බලමු ඒකත් පහත උදාහරණෙන්.

Syntax:

```
do
{
    //do some thing
}
while(expression);
```

Example :

```
public class example_10_03
{
    public static void main(String args[])
    {
        int num = 1;

        do
        {
            System.out.println(num);

            num ++;

        }
        while (num < 6);
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
1
2
3
4
5

Process finished with exit code 0
```

## FOR EACH LOOP

**for each loop** ගැන කරන්න කලින් අපි **arrays** ගැන පොඩ්ඩක් මතක් කරන් ඉමු. java වල අපි **array** එකක් හදා ගන්න විදිය පහළ උදාහරණෙන් මතක් කර ගමු.(chapter 10 – more about array)

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

**for each loop** එකේදී අපි කරන්නේ **array 1** ඇතුළේ තියන දත්ත එක වරකට එක බැගින් **variable** එකක ගබඩා කර ගන්න එක.ඇත්තටම **arrays** සහ **dictionaries** යන **data type** වල අපිට **for each loop** එක භාවිතා කරන්න පුළුවන්.

Syntax:

```
for(element : array )
{
    //do something
}
```

Example;

```
public class example_10_04
{
    public static void main(String args[])
    {
        int array[] = {1,2,3,4,5,6,7,8,9};

        for (int number : array)
        {
            System.out.println(number);
        }
    }
}
```

Expected Output:

```
1
2
3
4
5
6
7
8
9
```

## Chapter 11 : Array

තනි වර්ගයක [same data type] අගයන් නිශ්චිත ප්‍රමාණයක් රඳවා තබා ගැනීමට භාවිතා කරනා වස්තු විශේෂයක් [container object] ..ඉතින් කොහොමද මේ ජාවා array එකක් නිර්මාණය කරගන්නේ. මූලිකවම අපිට පියවර තුනක් අවශ්‍ය වෙනවා ඒ සඳහා.

1. array එක හැඳින්වීම [declare an array]
2. array එක ගොඩ නැගීම [construct an array]
3. array එකේ ප්‍රාථමික අදියර [initialize an array]

### Declare an array

අපි දන්න බලාපොරොත්තු වෙන **data type** එක එක්ක **array variable** එක හැඳින්වීම විතරයි.වෙනස තියෙන්න අගට අපි කොටු වරහනත් එකතු කරනවා.

Example:

```
String name [ ];  
int number [ ];
```

### Construct Java array

දැන් බලමු කොහොමද අර කිව්ව array එක **construct** කරගන්නේ කියලා.

මොකුත් කරන්න නැ..තියෙන්නේ **new** කියලා දන්න විතරයි.කලින් **declare** කරද්දි දාපු **data type** එකත් ඉතින් මේ පැත්තටත් දානවා එතකොට.

```
int number [ ] = new int[10];
```

අපිට මේක දෙපාරකට දන්නත් පුළුවන්..මෙන්න මෙහෙමයි එහෙම කරන්නේ.

```
int [ ]number;  
number = new int[10];
```

**construct** කරගන්නා කියන්නේ ඉතින්; අපිට තියෙන්න හෙමිනිට හෙමිනිට අගයන් දාගෙන දාගෙන යන්නමයි. එහෙම දාන්නේ **array** එකේ **index** එක තියෙන ස්ථානයන් සමඟ හඳුන්වා දෙමින්. **array** වල මතක තියා ගන්න ඕනේ කරන කාරණාවක් තියෙනවා. ඒක තමයි **array** එකක් **zero based indexed**. (පලවෙනියට **array** එකට දාන **value** එක වැටෙන තැන **array** එකේ **0** වෙනි ස්ථානට. ඊළඟට **1** වෙනි, දෙවෙනි ඔහොම ඔහොම තමයි යන කථාව.)

Example :

```
public class example_11_01
{
    public static void main (String args[])
    {
        String [ ] name;

        name = new String[3];

        name[0] = "DESAHN";

        name[1] = "JAYASHANKA";

        name[2] = "AMARASINGHE";

        System.out.print("MY NAME IS : "+name[0]+" "+name[1]+" "+name[2]);
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
MY NAME IS : DESAHN JAYASHANKA AMARASINGHE
Process finished with exit code 0
|
```

හදන්න පාරකුයි ආයිත් **data** දාන්න තව පාරකුයි යනවට වැඩියෙන් අපිට ලේසි වෙනවා එක පාරම ඒ වැඩ දෙකම කරගන්න එක..ඉතින් ජාවා අපිට ඒකටත් ඉඩ දීලා තියෙනවා.ඒක කරන්නේ මෙහෙමයි.

```
String name[ ] = {"DESHAN","JAYASHANKA","AMARASINGHE"};
```



## How to find length of array

Syntax:

```
array_name.length;
```

example :

```
public class example_11_02
{
    public static void main (String args[ ])
    {
        String name[] = {"DESHAN","JAYASHANKA","AMARASINGHE"};

        System.out.print("Length Of Array: "+name.length);

    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Length Of Array: 3
Process finished with exit code 0
|
```

## Chapter 12 : Array List

අපි **Arrays** ගැන දැනගෙන හිටියත් ඒ **array** වලදී අපිට කරන්න බැරි ගොඩක් දේවල් අපිට මේ **arraylist** හරහා කරන්න පුළුවන්. අපි පොඩ් ප්‍රැක්ටිකල් එකක් එක්කම මේක ගැන ඉගෙන ගමු.

අපි අපේ කඩ්ට ගෙනියන බඩු ලැයිස්තුව ජාවා වලින් පොඩ් ඇප්ලිකේෂන් එකක් විදියට **code** කරමු.

එකකොට අපිට මේ භාණ්ඩ ලැයිස්තුවට භාණ්ඩ **add** කරන්න භාණ්ඩ වෙනස් කරන්න වගේම භාණ්ඩ අයින් කරන්නත් පුළුවන් වෙනත් ඕනි.

### Step 01:

මුලින්ම **input** එකක් ගන්න ඕනි **module** එකයි. **Array list module** එකයි **import** කරගෙන ඉමු.

```
import java.util.Scanner;

import java.util.ArrayList;

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

    }

}
```

### Step 02:

දැන් අපි අපේ **list** එකයි **input** එකක් ගන්න ඕනි වෙන **object** එකයි හදාගමු.

```
private static Scanner sc = new Scanner(System.in) ;

private static ArrayList groceryList = new ArrayList() ;
```

### Step 03 :

මුලින්ම අපි අපේ **list** එකට භාණ්ඩ ඇඩ් කර ගන්න **method** එක හදා ගමු.

```
public static void addItem()
{
    System.out.print("Enter Your First Item : ") ; // method for add items
```

```
String item = sc.nextLine();

    groceryList.add(item) ;
}
```

මෙතන වැදගත් දේ වෙන්නේ **groceryList.add(item)** ; කියන line එක. මේ **add** keyword න් අපිට අපේ **list** එකට ඕනිම දෙයක් ඇඩ් කරගැනීමේ හැකියාව ලැබෙනවා.

දැන් අපි අපේ **list** එකේ තියන භාණ්ඩයක් අයිත් කරන්න ඕනි **method** එක ලියමු.

```
public static void removeItem()
{
    System.out.print("Enter The index of removable item : ") ; // method for remove
    items

    int itemIndex = sc.nextInt() ;

    groceryList.remove(itemIndex-1) ;
}
```

**groceryList.remove(itemIndex - 1)** ; මෙන්න මේ line එක තමා මේ **method** එකෙ වැදගත්ම line එක වෙන්නේ. අපිට අපේ **list** එකෙන් මොකක් හරි **item** එකක් අයිත් කර ගන්න ඕනි චුනාම අපිට මේ **remove** කියන keyword එක භාවිත කරන්න පුළුවන්.

මෙතනදි මන් **itemIndex - 1** කියලා දීල තියෙනව. ඒ මොකද කිව්වොත් අපි **list** එකෙ ගනන් කරන්න ගත්තේ එක ඉඳන් චුනාට **lists** ඇත්තටම පටන් ගත්තේ **0** කියලා ඔයාල දන්නවා. අන්න ඒ නිසා තමා අපි මෙකෙන් එකක් අඩු කරලා ඇත්ත **index** එක හරියට හදා ගන්නවා.

දැන් අපි යමු **list** එකේ භාණ්ඩ **update** කරන්න පුළුවන් වෙන **method** එකට.

```
public static void updateItem() // method for update items
{
    System.out.print("Enter The index of update item : ") ;

    int itemIndex = sc.nextInt() ;

    System.out.print("Enter the update item name : ") ;

    String updateItemName = sc.nextLine() ;
```

```
groceryList.set(itemIndex-1,updateItemName) ;  
}
```

groceryList.set(itemIndex-1,updateItemName) ; කියන එකෙන් මන් update item එකට අදාළ item එක මාරු කරනවා .

දැන් අපි යම් අපේ main method එකට.අපි කියමු යමක් ඇඩ් කරන්න ඕනි නම් අන්ක 1 ද., delete කරන්න ඕනි නම් 2 ද, update කරන්න ඕනි නම් 3 input කරන්න ඕනි කියලා.

```
public static void main(String[] args)  
{  
    Scanner input = new Scanner(System.in);  
  
    System.out.print("Press 1 For ADD \nPress 2 For Delete \nPress 3 For Update\nInput  
:");  
  
    char choice = input.next().charAt(0);  
  
    if (choice == '1')  
    {  
        addItem();  
    }  
    else if(choice == '2')  
    {  
        removeItem();  
    }  
    else if(choice == '3')  
    {  
        updateItem();  
    }  
    else  
    {  
        System.out.print("Wrong InPut");  
  
        System.exit(0);  
    }  
}
```

Full Code :

```
import java.util.Scanner;

import java.util.ArrayList;

public class array_list_demo
{
    private static Scanner sc = new Scanner(System.in) ;

    private static ArrayList groceryList = new ArrayList() ;

    public static void addItem()
    {
        System.out.print("Enter Your First Item : ") ; // method for add items

        String item = sc.nextLine();

        groceryList.add(item) ;
    }

    public static void removeItem()
    {
        System.out.print("Enter The index of removable item : ") ; // method for remove
items

        int itemIndex = sc.nextInt() ;

        groceryList.remove(itemIndex-1) ;
    }

    public static void updateItem() // method for update items
    {
        System.out.print("Enter The index of update item : ") ;

        int itemIndex = sc.nextInt() ;

        System.out.print("Enter the update item name : ") ;

        String updateItemName = sc.nextLine() ;

        groceryList.set(itemIndex-1,updateItemName) ;
    }
}
```

```

public static void main(String[] args)
{
    Scanner input = new Scanner(System.in);

    System.out.print("Press 1 For ADD \nPress 2 For Delete \nPress 3 For
Update\nInput :");

    char choice = input.next().charAt(0);

    if (choice == '1')
    {
        addItem();
    }
    else if(choice == '2')
    {
        removeItem();
    }
    else if(choice == '3')
    {
        updateItem();
    }
    else
    {
        System.out.print("Wrong InPut");

        System.exit(0);
    }
}
}

```

NOTE :

This Code is demo code there for You  
Catch any error or any bug please  
inform me :  
[djayashanka750@gmail.com](mailto:djayashanka750@gmail.com)

Expected Output:

When choose 01:

```

"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"
Press 1 For ADD
Press 2 For Delete
Press 3 For Update
Input :1
Enter Your First Item : cookies

Process finished with exit code 0

```

## 13 OOP : Object-Oriented Programming

**Object** කියන්නේ මොකක්ද හැබෑ. අපි ඉගෙනගන්න යන මෘදුකාංග වල **Object** කියන්නේ අපි මේ දකින ලෝකයේ ඇත්ත වශයෙන්ම පවතින වස්තූන් වලටයි. එබඳු මේ ලෝකයේ වස්තූන් වලට **state** හා **behavior** ඒ කියන්නේ ස්වභාවයන් සහ හැසිරීම් පවතිනවා. විද්‍යාත්මකව මේ මෘදුකාංග වල **Object** කියන්නේ **memory location** එකක්. ඒ කියන්නේ අපි කවුරුත් **RAM** එක කියන ප්‍රාථමික මතකයෙන් ගන්නාවූ ඉඩ ප්‍රමාණයක්. අපි **Object** එකක් නිර්මාණය කරන සෑම විටදීම **RAM** එකෙන් ඉඩක් මේ වෙනුවෙන් වෙන් කරනු ලබනවා.

ඇත් අර **state behavior** කථාව අරන් බලමු. අපි සත්‍ය ලෝකයෙන් උදාහරණයක් ලෙසට බයිසිකලය කියන එක ගත්තොත් බයිසිකලයට තියෙන **state** මොනාද?

ඒ තමයි ඇත් ඉන්න ගියර් එක, ඇත් වේගය අන්න ඒ වගේ දේවල්. තව.. හැසිරීම් එහෙමත් නැත්නම් ක්‍රියාවන් තමයි ගියරය මාරු කිරීම, පැඩලය මාරු කිරීම, තිරිංග තද කිරීම. මෙන්න මේ අපි කිව්ව **state , behavior** ගැන හොඳට ඔලුවට දා ගන්න තරමට ඔයාලට ලේසි වෙනවා **Object orient** එකේ තියෙන සංකල්ප ටික හරියට ඔලුවට දාගන්න.

මුලින්ම අපි ඉගෙනගනිමු **class** එකක් කියන්නේ මොකද්ද කියලා. ඒක හරියට **Object** එකක් තනන සැලැස්මක පිටපතක් එහෙමත් නැත්නම් ආකෘතියක් කියන්න පුළුවන්. නිදසුන් ලෙසට ගත්තොත්; බයිසිකලය කියලා කියන්නේ බයිසිකලය කියන නමින් හදාගත්තු **class** එකේ **instance** එකක්.

Example:

```
class Bicycle { //class 01

    int speed = 0;

    int gear = 1;

    void changeGear(int newGear) // method 01
    {
        gear = newGear;
    }

    void speedUp(int increment) // method 02
    {
        speed = speed + increment;
    }

    void applyBreak(int decrement) // method 03
    {
        speed = speed - decrement;
    }
}
```

```

void printState() // method 04
{
    System.out.println("gear is : " + gear + " & speed is : " + speed);
}
}

class TestSCJP // class 02
{
    public static void main(String[] args) // main method
    {
        Bicycle bicycle = new Bicycle();

        bicycle.printState();
    }
}

```

main OOP Concepts use in Java :

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

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

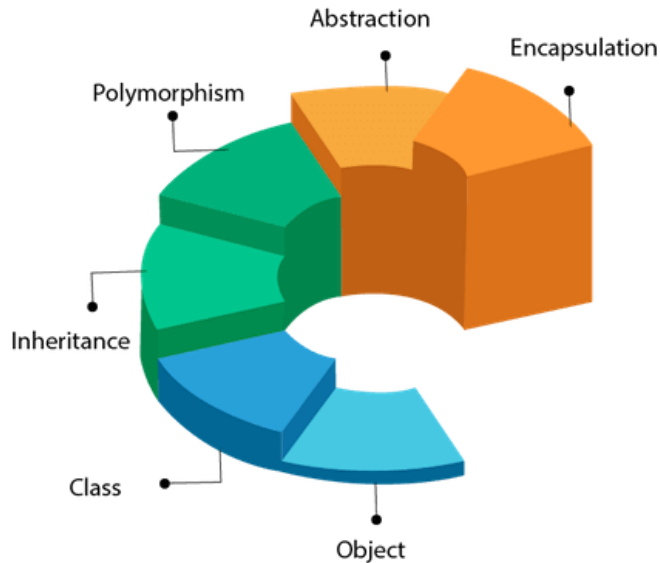
Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- Coupling
- Cohesion
- Association
- Aggregation



- Composition

## OOPs (Object-Oriented Programming System)



Continue Reading : <https://www.javatpoint.com/java-oops-concepts>

### 13.1 Class and Objects :

**class** එහෙමත් නැත්නම් පන්තියක් කියන එක අපිට පැහැදිලි කරන්න පුළුවන්.යම් කිසි එකිනෙකට අනන්‍ය වූ විශේෂයක් අපිට එක **class** එකක් විදියට ගන්න පුළුවන්.උදාහරණයක් විදියට ක්ෂීරපායින් කියන එක අපිට තනි පන්තියක් විදියට ගන්න පුළුවන්,එහෙමත් නැත්නම් මත්ස්‍යයන් කියන සත්ත්ව කාණ්ඩය අපිට එක පන්තියක් විදියට ගන්න පුළුවන්.

**Object** එකක් යනු **data** සහ **method** වල එකතුවක් ලෙස හඳුන්වන්න පුළුවන්. සැබෑ ලෝකයේ වස්තූන් **Java** තුල නිරූපණය කරන්නේ **objects** භාවිතයෙන්. **classes** තමයි **object** එකේ **design** එක පිළිබඳ තීරණය කරන්නේ. උදාහරණයක් විදියට අපි බල්ලෙක් ගමු.බල්ලාට වලිගයක් කකුල් හතරක් කටක් තියනවා.එතකොට බල්ලා කියන පන්තිය ඇතුළත් වන **objects** තමා මේ වලිගය,කකුල් හතර,කට කියන්නේ.

**class** එක භාවිත කර **objects** සෑදූ පසු ඒවා එක එකක් වෙන වෙනම පරිගණකයේ ප්‍රධාන මතකයේ(RAM) තැන්පත් වීම සිදුවනවා. ඉන්පසු අදාල කාර්යය ඉටුකර ගැනීම සඳහා අපට ඒවා යොදාගන්න පුළුවන්.

Example : television.java

```

public class television
{
    String brand;

    float inch;

    String technology;

    void channel()
    {
        String channel_name;

        int channel_number;

        float frequency;
    }

    void display()
    {
        String technology;

        int resolution;

        float frequency;
    }

    void details()
    {
        String colour;

        String manufacture_day;

        int price;

        float discount;

        float warranty;
    }
}

```

Class name : television

Objects in television class : channel, display ,details

අපි දැන් බලමු **class** එකකින් **objects** සාදාගැනීමට අදාළ **Java syntax** මොනවාද කියලා. උදාහරණ සඳහා ඉහත **television class** එකම යොදා ගනිමු.

**class** එකකින් **object** එකක් සාදා ගැනීම සඳහා **new** Keyword එක භාවිතා කෙරෙනවා.

Syntax 01:

```
Class_name Object_Name;
```

```
Object_Name = new ClassType(); //default constructor
```

```
Class_name Object_Name;
```

```
Object_Name new Class_Type(parameter-list); // With parameters
```

Example : default constructor(use television class)

```
public class television
{
    String brand;

    float inch;

    String technology;

    void channel()
    {
        String channel_name;

        int channel_number;

        float frequency;
    }

    void display()
    {
        String technology;

        int resolution;

        float frequency;
    }

    void details()
    {
        String colour;

        String manufacture_day;

        int price;

        float discount;

        float warranty;
    }
}
```

Class name : Television class

Objects in this class :channel

Display

details

Use above class :

```
public class main_television
{
    public static void main(String args[ ])
    {
        television tv;
        tv = new television();
    }
}
```

} television = class name  
// tv = object name we given  
//television() = class type

Syntax 02 :

```
Class_name Object_Name = new ClassType(); //default constructor
Class_Type Object_Name = new ClassType(parameter-list); // With parameters
```

Example : default constructors

```
public class main_television
{
    public static void main(String args[ ])
    {
        television tv = new television()
    }
}
```

} television = class name  
// tv = obj name we given  
//television( ) = class type

Explain above process :

දැන් අපි බලමු අපි සකසාගත් **object** එක පරිගණකයේ ප්‍රධාන මතකයේ(RAM) තැන්පත් වීම සිදුවන්නේ කොහොමද කියලා.

මුනිත්ම **new keyword** එක ගැන බලමු.

**new keyword** එක මගින් සිදුවන්නේ දී ඇති **class name** එකෙන් **object** එකක් RAM එක තුළ සාදා එයට **reference** එකක් (යොමුවක්) අදාළ **variable** එක වෙත යැවීමයි.

ඉහත උදාහරණය අනුව ගතහොත්, **unique(අනන්‍ය) television object** එකක් RAM එක තුළ සාදා එයට **reference** එකක් tv නමැති **variable** එකට **assign** කිරීම සිදුකරයි.

මෙහිදී ඔබ තේරුම් ගත යුතු කරුණ වන්නේ ඇති **Java** හි භාවිතා වන **variable** දරාගෙන(hold) සිටින්නේ **object** එකක් නොව **object** එක සඳහා **reference** එකකි. තව දුරටත් පැහැදිලි කරන්නේ නම් **Java variable** එකක් තුළ ඇත්තේ **object** එක **RAM** එක තුළ පවතින ස්ථානයට අදාළ **memory**

address එකකි. memory address එකක් සඳහා Ex : 0x23FF50 එනම් object එක හඳුනා ගැනෙනුයේ අදාල memory address එකෙනි.

## Variable Types Use in Classes

1. Local variables

2. Instance variables

3. Class variables

**Local variables** : constructors, methods වනේ ඒවා ආනුලභ භාවිතා කරන විචල්‍යය(variables) නම්ප නමින් හඳුන්වනවා.method එක call කරාම පමණක් අපිට නම්ප විචල්‍ය භාවිතා කරන්න පුළුවන්.

**Instance variables** : class ආනුලභ ටයන variables නම්ප නමින් හඳුන්වනවා.අනේ උදාහරණයක් ගන්නාත් String breed , int age කියන්නන් instance variables .නම්පවා අපිට ඒ class එක ආනුලභ ටයන method ආනුලභ call කරන්න පුළුවන්.

**Class variables** : class එක ආනුලභ ටයන තවත් variable වර්ගයක්.static, non-static වියට නම්ප variables වර්ග කරන්න පුළුවන්.

## 13.2 Methods

Method එකක් යනු function/procedure සඳහා ලබාදී ඇති OOP term එකයි.

### Why methods are important ?

1.වැඩසටහනක් modules වලට කැඩීමේදී උපයෝගී කරගනී.

2.Code reusability සංකල්පය එනම් කේත නැවත නැවත, අදාල ස්ථානයන්හිදී භාවිතා කිරීමට යොදාගත හැක.

3.Method එකක් ක්‍රියාත්මක(execute) වනුයේ එය call කළ විට පමණි (explicitly invoked).

## Method Declaration

Method එකක් සෑම විටම පැවතිය යුත්තේ class එකක් තුලයි. method එකක් declare කිරීමෙන් පසු අපට අවශ්‍ය විටක එය invoke කළ හැකිය. එතෙක් එය තුල ඇති උපදෙස් ක්‍රියාත්මක නොවේ. පහත syntax එකට අනුව method එකක් declare කිරීම සිදුකෙරේ.

Syntax:

```

Modifiers returnType Method_Name (parameter-list) // method header
{
    /*
    Body
    */
    return expression;
}

```

## Method Header:

### modifiers

මෙයට **method scope** යයිද කියනවා. මෙමගින් **method** එක **access** කිරීම පිළිබඳ තීරණ ගනු ලබනවා.

මෙම **access modifiers** වනුයේ **public, private, protected, default/friendly** යන ඒවායි.

### Return Type

**method** එකෙන් එය **call** කළ ස්ථානයට යවන(**return**) පණිවුඩය කුමන දත්ත ආකාරයක් ඇති එකක්ද යන්න මෙමගින් සඳහන් කෙරේ.

**method** එක **invoke** කරනවිට **method** එකට ලබාදෙන දත්ත **parameter-list** එක තුළ අන්තර්ගත වේ.

### method body

**method** එකෙන් සිදුකරන කාර්යයට අදාළ කේත මෙතුළ අඩංගුවේ.

### return expression

**method** එකෙන් එය **call** කළ ස්ථානයට පණිවුඩය යැවීම මෙමගින් සිදුකෙරේ. මෙම **statement** එක **execute** වී අවසන් වූ වහාම **method** එකෙහි **execute** වීම අවසන්වේ. සෑම **method** එකක්ම **values return** කිරීම අනිවාර්ය නොවේ. මෙසේ **values return** නොකරන **method** වලදී **modifier** එක ලෙස **void** යන්න භාවිතාවේ.

## How to call method :

මෙය **method** එක සඳහා **values pass** කිරීම ලෙසද හඳුනවනු ලැබේ.

**syntax:**

```

name_of_method(arguments);

```

**argument-list** එකෙහි ඇති **arguments** හරියටම **method** එකේ **parameter-list** එකෙහි ඇති **parameters** වලට අනුරූප විය යුතුය.

**variable pass** කරන්නේ නම් මෙම **arguments** අගයන් සහිත තාත්ත්වික විචල්‍යයන්(**real variables**) විය යුතුය. මේ නිසා මෙම **arguments pass** කිරීමට පෙර එම **variables initialize** කර තිබිය යුතුය.

## Call selected method in the class

මෙහිදී කෙලින්ම **method** එකේ නම භාවිතාකරමින් **invoke** කළ හැක. අවශ්‍ය නම් **this keyword** එකද භාවිතාකළ හැකිය.

```
method_Name();  
  
//OR  
  
this.method_Name();
```

## calling methods out of the class

මෙහිදී **method** එක අයත් **object** එකේ නමද යොදාගැනේ

```
object_name.Method_Name(args_list);
```

example :

```
import java.util.Scanner;  
  
public class Main  
{  
    public static void main(String args[ ])  
    {  
        Scanner numbers = new Scanner(System.in);  
  
        System.out.print("Enter Number 01 : ");  
  
        int num1 = numbers.nextInt();  
  
        System.out.print("Enter Number 02 : ");
```

```

    int num2 = numbers.nextInt();

    operations maths_operations = new operations();

    maths_operations.addition(num1,num2);

}
}

```

## How to call Static Methods

```
Class_Name.method_Name();
```

Example :

```

public class random
{
    public static void main(String args[])
    {
        System.out.print(Math.random());
    }
}

```

Example :

Write simple java class and main class for find area of circle :

```

public class area
{
    public void area(double r)
    {
        double area = 2 * 3.4 * r;

        System.out.print("Area Of Circle "+ area);
    }
}

```

} area class

Main Class:

```
import java.util.Scanner;
```



```

public class Main
{
    public static void main(String args[ ])
    {
        Scanner value = new Scanner(System.in);

        System.out.print("please Enter R value :");

        double r = value.nextDouble();

        area obj = new area();

        obj.area(r);
    }
}

```

Expected Output :

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
please Enter R value :7
Area Of Circle 47.6
Process finished with exit code 0

```

Another Example Of Class Object and method Concept :

Write simple java class and main class for perform main mathematical operations ( addition, subtraction, multiplication, division)

Class name : operations

Objects in this class : addition, subtraction, multiplication, division

```

package Class_Object;

public class operations
{
    void addition(int x,int y) // addition
    {
        int total = x + y;

        System.out.print("Total is : " +total+"\n");
    }

    void subtraction(int x,int y) //subtraction
    {
        int ans = x - y;
    }
}

```

```

        System.out.print("Subtraction is : " +ans+"\n");
    }

    void multiplication(int x,int y) //multiplication
    {
        int ans = x * y;

        System.out.print("Multiplication is : " +ans+"\n");
    }

    void division(int x,int y) //division
    {
        int ans = x / y;

        System.out.print("Division is : " +ans+"\n");
    }
}

```

Main class :

```

package Class_Object; //package name

import java.util.Scanner;

public class Main
{
    public static void main(String args[ ])
    {
        Scanner numbers = new Scanner(System.in);

        System.out.print("Enter Number 01 : ");

        int num1 = numbers.nextInt();

        System.out.print("Enter Number 02 : ");

        int num2 = numbers.nextInt();

        operations maths_operations = new operations();

        maths_operations.addition(num1,num2);

        maths_operations.subtraction(num1,num2);
    }
}

```

```

    maths_operations.multiplication(num1,num2);

    maths_operations.division(num1,num2);

}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Enter Number 01 : 15
Enter Number 02 : 3
Total is : 18
Subtraction is : 12
Multiplication is : 45
Division is : 5

```

### 13.3 Constructors

**constructor** මගින් සිදුකරනුයේ එය අයත් **class** එකට අදාලව සාදන **object** එක initialize කිරීමයි. දැනටමත් ඔබ දන්නවා **class** එකකින් කොහොමද **object** එකක් සාදාගන්නේ කියලා.මේ සඳහා **new** keyword එක භාවිතාවන බවත් ඔබ දන්නවා ඇති.

#### Features of constructors

- ✚ **constructor** එකක නම එය අයත් **class** එකේම නමම වනවා.Ex: area class එකේ **constructor** එකේ නමත් **Radio** ම වනවා.

14 arguments එකක් හෝ කිහිපයක් තිබිය හැකියි. උදා : **Radio(String band,float frequency)**

return values ගැන කිසිදු සඳහනක් නොමැත.

15 **constructor** එකක් හෝ කීපයක් පැවතිය හැකිය.

#### Duty of constructor

- ✚ **local variable** වල ආරම්භක අගය සැකසීම.(initialize)
- ✚ කීප ආකාරයකට **object** එක සෑදීමට යොදාගත හැකිවීම

Syntax:

```

public Constructor_Name(argument-list)

```

```
{
    //constructor body
}
```

For example :

```
class Circle
{
    int x,y,radius;

    public Circle(int x,int y,int r) // Constructor 01
    {
        this.x = x;

        this.y = y;

        radius= r;
    }

    public Circle(int x,int y) // Constructor 01
    {
        this.x = x;

        this.y = y;

        radius=10;
    }
}
```

ඇන් අපි බලමු **Circle class** එකෙන් **Circle object** එකක් සෑදීමේදී **constructor** එක වැදගත් වන ආකාරය. අපි ඉහත **class** එකේ අන්තර්ගත කරන ලද **constructor** භාවිතාකර **object 2ක්** සාදමු.

```
//using constructor 1
Circle c1=new Circle(12, 34, 7);

//using constructor 2
Circle c2=new Circle(14, 47);
```

✚ **c1 object** එක සෑදීමේදී යෙදෙන්නේ constructor 1 එකයි. එම constructor එකෙහිදී සිදුවන්නේ x,y හා r ලෙසින් arguments 3ක් ලබාගෙන c1 object එකෙහි class variables ලෙස පවතින x, y, හා radius variables initialize කිරීමයි.

✚ **c2 object** එක සෑදීමේදී යෙදෙන්නේ constructor 2 එකයි. එහිදී arguments ලෙස ලබා ගනුයේ x හා y පමණි. radius එක 1 ලෙසට initialize කිරීම මෙම constructor එකෙන් සිදුකරයි. එනම් මෙම constructor එක(constructor 2) භාවිතයෙන් සාදන සෑම object එකකම radius එකෙහි initial value එක 1 වේ.

## this keyword

this keyword එක භාවිතා වන්නේ කිසියම් object instance එකකට අයත් දෑ එම object එක තුලදී refer කිරීමටයි.

මෙහිදී this keyword එක භාවිතා වන්නේ class variables හා arguments refer කිරීමේදී ඇතිවන ගැටලුවෙන් මිදීමටයි. class variables යනු සමස්ථ class එකටම බලපවත්වන ලෙස පවතින variables ය. ඉහත Circle class එකෙහි x, y හා radius යනු Circle class එකෙහි class variables වේ. constructor 1 හි තවත් x හා y ලෙසින් arguments 2ක් ගෙන ඇති බව ඔබට පෙනෙනවා ඇත. එම නමුත් එක x හා y විචල්‍යයන් බලපවත්වනුයේ/වලංගු වනුයේ constructor body එක තුලදී පමණි. this.x ලෙසින් අප refer කරනුයේ class object එකට අයත් x අගයයි. අනෙක් x එක argument එකයි. එනම් this.x = x යන්නෙහි අර්ථය වන්නේ argument එකක් ලෙස සපයන x හි අගය object එකට අයත් x වෙත ලබාදීමයි.

## 13.4 Access Modifiers

මෙහිදී වෙන්වෙන් program entity එකක් තවත් program entity එකක් [classes, interfaces, methods සහ variables] සඳහා හඳුනාගැනීමේ සහ එහි ප්‍රයෝජන ලබාගැනීමේ හැකියාව ලබාදීමයි. ප්‍රධාන වශයෙන් අපිට ජාවා තුලදී access modifiers හතරක් දක්නට ලැබෙනවා.

1. public
2. protected
3. default
4. private

මේ දාලා තියෙන්නේ හැකියාවන් වල ඉහළම අගයේ සිට අවම අගය දක්වා අනුපිළිවෙලින්. ඒ කියන්නේ public කියන එකට තියෙන accessing හැකියාව ගොඩාක් වැඩියි private එකට වැඩියි. private කියන්නේ හරිම සීමිත access ප්‍රමාණයක් තියෙන modifier එකක්.

Access modifier	this(class)	subclass	package	general
public	●	●	●	●
protected	●	●	X	X
default	●	X	●	X
private	●	X	X	X

### 1. Public

අපිට **public** කියන keyword එක top level එකේ **classes, interfaces, methods** සහ **variables** කියන ඕනෙම දේකට **apply** කරන්න පුළුවන්. මේ **public** යොදලා ඒ වගේ හදන ඕනෑම **entity** එකක් අපිට **access** කරන්න පුළුවනි එකම **package** එකේ හෝ වෙනත් **package** එකක ඉදලා උනත්.

## 2.Protected

මේ keyword එකෙන් අපිට හැකියාව තියෙනවා සමාන **package** එක තුල පවතින **class** ඒ කියන්නේ **program entities** භාවිතාවට හැකියාව ලබාදීම. මේක අපිට **public** එක වගේ **classes, interfaces, methods, variables** කියන සියල්ලටම **apply** කරන්න නම් බැ.නමුත් මේකෙන් පුළුවන් **methods** හා **variables** වලට **apply** කිරීම තුලින් **access** කිරීමේ හැකියාව සීමා කිරීම් සිදු කරන්නට හැක. අදාල **package** එකෙන් පිට ඉදන් අපි **protected** කියලා දාපු **methods** හා **variables** **access** කරන්නට හැකියාව නැ.නමුත් පිටින් පිහිටියත් **inherited** කිරීමක් කර ඇත්නම් **access** කිරීමට හැකියාව ලබාදී තියෙනවා.

## 3.Default

මේ කියන්නේ මොකක්ද.?**default** කියලා ඇත්තටම භාවිතා කරන් නැ keyword එකක් **access** සීමා කිරීම සඳහා සාමාන්‍යයෙන්.මේ කියන්නේ කිසිම **access modifier** එකක් අපි **class, method** හෝ **variable** එකක් ඉදිරියෙන් යොදන්නෙ නැත්නම් තමයි ඒක **default** කියන **category** එකට වැටෙන්නේ.

## 4.Private

මෙහිදී අනිවාර්යෙන්ම සමාන **package** එක තුලම පමණයි පෙරකී **default** ලෙසට **access modify** කරන ලද **program entity** එක භාවිතා කිරීමේ හැකියාව තියෙන්නේ. **package** එකට පිටින් නම් භාවිතාවට ඉඩ දෙන්නේ නැ. මේක **apply** කරන්න පුළුවන් වෙන්නේ අනිවාර්යෙන්ම **method** හා **variable** වලට පමණයි.තනි **class** එකක් තුල පමණක්මයි භාවිතා කරන්න පුළුවන් වෙන්නේ **private** කියන keyword දාපු අදාල **program entity** එක.

දැන් අපි **private** සහ **public modifiers** ගැන අධ්‍යයනය කිරීම සඳහා සරල ජාවා වැඩසටහනක් ලියමු.

මේ සඳහා අපි **Employee** කෙනෙකු උදාහරණයට ගනිමු.

**Employee.java**

(Source : [javaxclass.blogspot.com/2010/05/iv-access-modifiers.html](http://javaxclass.blogspot.com/2010/05/iv-access-modifiers.html))

Example :

```

package Chapter_15;

class Employee {
    //instance variables
    public String Name; // Name is a public member
    private float HourlyRate; //private member
    private float HoursWorked; //private member

    //constructor
    public Employee(String name,float hr,float hw){
        Name=name;
        HourlyRate=hr;
        HoursWorked=hw;
    }

    //public method
    public void setHourlyRate(float hr)
    {
        HourlyRate=hr;
    }
    //private method
    private float getTotalSal()
    {
        return (HourlyRate*HoursWorked);
    }
    //public method
    public void showDetails()
    {
        System.out.println("Employee Name : " + Name);
        System.out.println("Total Salary : " + getTotalSal() );
        System.out.println("-----");
    }
} //end of the Employee class

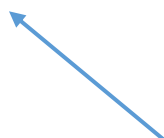
```

🚦 දැන් අපි **Employee class** එක යොදාගෙන පහත කේතය **execute** කර ලැබෙන ප්‍රතිඵලය කුමක්දැයි බලමු.

```

public static void main(String args[ ])
{

```



```
Employee emp = new Employee("Deshan",1500.5f,50.0f);

emp.showDetails();

emp.Name = "Kavee";

emp.showDetails();
}
```

### 3<sup>rd</sup> and 4<sup>th</sup> Line

#### 3<sup>rd</sup> line

මෙහිදී ඉහත සාදාගත් **emp1 object** එකට අයත් **public member** වන **Name** විචල්‍යය **access** කර එහි **name** එක **Kavee** ලෙසට වෙනස් කරනු ලැබේ.

#### 4<sup>th</sup> line

මෙහිදී නැවතත් **emp1 object** එකෙහි **showDetails() method** එක **invoke** කරයි. දැන් **console** එකෙහි දිස්විය යුත්තේ **Name** එක **Kavee** ලෙසටය.

### 1<sup>st</sup> and 2<sup>nd</sup> Line

මෙහිදී **Employee object** එකක් සාදා මුලින්ම එහි **public member** කෙනෙකු වන **showDetails() method** එක **invoke** කරයි.

### Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Employee Name : Deshan
Total Salary  : 75025.0
-----
Employee Name : Kavee
Total Salary  : 75025.0
-----
```

### Example for private Modifier :

මීලඟට **private modifier** එක අධ්‍යයනය කිරීමට පහත කේතය **execute** කරමු :



```
public static void main(String args[ ])
{
    Employee emp1=new Employee("Kavee",1678.25f,50.0f);

    emp1.HourlyRate=1500.00f;
}
```

මෙහිදී emp1 ගේ private modifier එකක් වන HourlyRate විචල්‍යය අපි access කිරීමට යයි. එම නිසා මෙහිදී compile කිරීමේදී දෝශ පණිවුඩය පෙන්වයි.

අපට නිවැරදිව HourlyRate එක වෙනස් කිරීමට කළයුතු වන්නේ අප Employee class එකෙහි අන්තර්ගත කොට ඇති setHourlyRate() නම් public method එක භාවිතා කිරීමයි.

```
public static void main(String args[ ])
{
    Employee emp1=new Employee("Kavee",1778.25f,50.0f);

    emp1.showDetails();

    emp1.setHourlyRate(500.0f);

    emp1.showDetails();
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Employee Name : Kavee
Total Salary  : 88912.5
-----
Employee Name : Kavee
Total Salary  : 25000.0
-----

Process finished with exit code 0
```

## 13.5 Non Access Modifiers

Static :

static variables

**static** විචල්‍යයන් කියන්නේ ඒවා අයිති වෙන්නේ ඒක හැදින්වූ ඒ කියන්නේ **declare** කරපු **class** එකට නිසා..ඒ වගේම තමයි ඒවා **class** එක මගින් තනන කිසිම **instance** එකක කොටසක් වෙන්නේත් නෑ. **class** එක **load** වන වේලාවේදීම **static** විචල්‍යයන් තමන්ගේ **default** අගයන්ට අදාලව **initialize** වෙනවා.හැබැයි ඉතින් අපි ඒකට අගයක් දීලා තිබුනොත් නම් ඒ කියන්නේ **explicitly initialize** කරල නම් ඒ කරපු අගයන් තමයි **load** වෙන්නේත්.

Example :

```
public class Static_Variable
{
    static int number = 10; // Static Variable

    static String name = "DESHAN JAYASHANKA"; // Static Variable

    public static void main(String args[ ])
    {
        System.out.print("MY NAME IS : "+name+"\n");

        System.out.print("INDEX : "+number);

    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
MY NAME IS : DESHAN JAYASHANKA
INDEX : 10
Process finished with exit code 0
|
```

## static methods

මේ කියන්නේත් **static variable** වගේම එකක්.මේකෙදී වෙනසකට තියෙන්නේ ඉතින් කලින් කලා කලේ **variable** එකක්.මේක **method** එකක්.එව්වරයි.එහෙමත් නෙමෙයි ඉතින් තව තියෙනවා දේවල් කීපයක් **static method** ගැන ඉගෙන ගන්න. අපි **static method** වලට කියනවා **class method** කියලත්.class එකක තියෙන **static method** එකකට පුළුවන් **class** එකේ තියෙන අනිත් **static members'**ලා ඒ කියන්නේ **variable** නැත්නම් **methods** කෙලින්ම **access** කරන්නත්.static නොවන ඒ කියන්නේ **non-static members'**ලාට බැ **static method** එකක් **access** කරන්න.කෙලින්ම **class** එකේ නම යොදලා මේ කියන **static method call** කරන්න හැකියාවක් තියෙනවා.

Example:

```
import java.util.Scanner;
```

```

public class Static_methods
{
    static int number1,number2;

    public static void main(String args[])
    {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter Number One : ");

        number1= input.nextInt();

        System.out.print("Enter Number Two : ");

        number2= input.nextInt();

        my_method(number1,number2);
    }

    public static void my_method(int num1,int num2) //static method
    {
        System.out.print(num1+" + "+num2+" = "+(num1+num2));
    }
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Enter Number One : 25
Enter Number Two : 25
25 + 25 = 50
Process finished with exit code 0
|

```

## static initialization block

මේක සාමාන්‍යය block එකක්.ඒ කියන්නේ { } සහල වරහන් වලින් වටවුණු කොටසක්.මේ block එක පටන්ගන්න තැන තියෙනවා static කියලා. මේක runtime system එක මගින් static initialization block එක call කරන්නේ source code එකේ ඒවා තියෙන පිළිවෙලටයි. static members'ලාව විතරයි ඉතින් මේකෙන් ඇතුලේ ඉඳන් access කරන්න පුළුවන්.

Example:

```
public class Static_Initialization_block
{
    static int num = 1;

    static
    {
        System.out.print("Block One : "+(num++)+"\n"); // static initialization block 01
    }

    public static void main(String args[ ])
    {
        System.out.print("Block Two : "+(num++)+"\n");
    }

    static
    {
        System.out.print("Block Three : "+(num++)+"\n"); // static initialization block 02
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Block One : 1
Block Three : 2
Block Two : 3
```

Final :

**final** කියන ජාතියේ විචල්‍යයක් කියන්නේ නියත වර්ගයේ විචල්‍යයක්..එක initialize කලහම ඒ කියන්නේ පලමුවරට ඒ විචල්‍යයට අදාලව අගයක් ලබාදුන් පසුව එය වෙනස් කිරීම සිදු කිරීම කළ නොහැකියි.මේ access modifier එක වුණත් අපිට method, class සහ variables කියන සියල්ලටම apply කිරීමේ හැකියාව ලැබෙනවා.

## final variables

Primitive (ප්‍රාථමික විචල්‍යයන්) - අපි දන්න විචල්‍යයන් වර්ග වන 1. Instance 2. Static 3. Local කියන විචල්‍යය වලට final කියන keyword එක apply කරන්න පුළුවනි. අගය නම් ඉතින් වෙනස් කරන්න බැරි එක පාරක් ආදේශ කලායින් පස්සේ.

Example :

```
package Class_Object;

public class final_variables
{
    final static int age = 20;

    final int number = 1645;

    public static void main(String args[])
    {
        final String name = "DESHAN JAYASHANKA";

        System.out.println("Student Name : "+name);

        System.out.println("Age : "+age);

        System.out.println("Index : "+number); // java: non-static variable number cannot
        be referenced from a static context

    }
}
```

Expected Output:

```
D:\Projects\Intellij\src\Class_Object\final_variables.java:17:39
java: non-static variable number cannot be referenced from a static context
```

Note : we can't use non static variables in static context

Example :

```
public class final_variables
{
    final static int age = 20;
```

```

public static void main(String args[ ])
{
    final String name = "DESHAN JAYASHANKA";

    System.out.println("Student Name : "+name);

    System.out.println("Age : "+age);
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"
Student Name : DESHAN JAYASHANKA
Age : 20

```

## Objects – reference type

reference type එකකට අදාලව final යන keyword එක ලබාදී initialize කළ පසුවද එය වෙනස් කිරීම සිදු කරන්නට නොහැකියි. final යෙදූ reference එකක් හැමතීස්සේම refer වන්නේ එම object එකමයි. ඒ කියන්නේ same object එකමයි.

```

final operations maths_op = new operations();

```

## final methods

කොහේහරි class එකක තිබුණොතින් final keyword එක යොදපු method එකක්, ඒ class එක මගින් සාදාගන්නා වූ sub classes ඒ කියන්නේ inherit කරගත්තු classes වල override කිරීමකට හැකියාවක් නොමැත. ඒකෙන් අදහස් කෙරෙන්නේ final යොදන method එකක හැසිරීම [behavior] එක නොවෙනස් බව සහතික කර ගැනීමටයි.

## final classes

final යෙදුවොත් එම class එක extend කිරීමට නොහැකියි. final class කියන එකෙන් අදහස් කරන්නේ මෙතනින් එහාට class එක inherit වීමක් වෙන්නේ නෑ සහ මේ අවසානම inheritance ධුරාවලි අංගය කියන එකයි.

Example :

```
import java.util.Scanner;

public class final_class
{
    public static void main(String args[ ]) //main class
    {
        Scanner input = new Scanner(System.in);

        System.out.print("Number 01 : ");

        int number1 = input.nextInt();

        System.out.print("Number 02 : ");

        int number2 = input.nextInt();

        final_class fc = new final_class();

        fc.operation(number1,number2);
    }

    final public void operation(int num1,int num2) // final class
    {
        int ans = num1 + num2;

        System.out.println("Answer : "+ans);
    }
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"
Number 01 : 50
Number 02 : 25
Answer : 75

Process finished with exit code 0
|
```

## abstract

මේ කියන්නේ අලුත් ජාතියේ **keyword** එකක්.මේකෙන් කියන්නේ කුමක් හෝ දෙයක් අසම්පූර්ණයි කියන එක හරි ඒ දේ පසුවට සම්පූර්ණ කරන්න පවතින්නක් කියන එකයි. මේක අපිට **class** වලට සහ

non-static method වලට apply කරන්නට පුළුවනි. මේකේදී මතක තියාගන්න **abstract** සහ **final** කියන keyword දෙක එකට භාවිතා කරන්න නුපුළුවන් කියන දේත්.

## abstract class

**abstract class** කියන්නේ ටිකක් වෙනස් ජාතියක **class** එකක්. මේකේදී **class** එක හඳුන්වන කොටම අපි දෙන්න ඕනේ **abstract** කියන keyword එක. **abstract class** එකක අනිවාර්යෙන්ම අඩුම වශයෙන් එක් **abstract method** එකක් තිබිය යුතුම වෙනවා. මෙම **class** වලින් අපිට **object** හදාගන්න බෑ. ඒ කියන්නේ **class** එකෙන් **instance** එකක් හදාගන්න ඉඩ දෙන්නේ නෑ **abstract class** වල. **abstract class** එකක් **abstract** නොවන **method** පවා තිබිය හැකියි. **abstract class** වලින් පසුවට වැඩගන්න නිසා තමයි ඒවා **final** කියන keyword එකත් එක්ක යොදන්න බැහැ කියන්නේ.

Example:

```
abstract class abstract_class // abstract class
{
    public abstract void my_method(); //abstract method

    public void operation1()
    {

    }

    public void operation2()
    {

    }
}
```

## abstract methods

මේවායේ **method** වල තියෙන වෙනස වෙන්නේ **implementation** කියන කොටස නෑ. තේරෙන සිංහලෙන් කිව්වොතින් මේ **method** එකේ **method body** එකක් නෑ. තියෙන්නේ අදාල **method** එක හඳුන්වාදීමක් සහ අවසානේ **semi colon** ";" ලකුණ විතරයි. තවත් වැදගත් දෙයක් තියෙනවා කියන්න අමතක වුණු. මේ **abstract method** වලට **private** කියන **modifier** එක දාන්න විදිහක් නෑ.

Example :

```
abstract void start_process();
synchronized
```

අපිට මේ ගැන කථා කරනකොට දැනගන්න ඕනේ කරනවා **thread** ගැන. **thread** කියන්නේ සමාන්තරව **run** වෙන **method** කිහිපයක් පැවතීම වගේ දෙයක්. හිතන්න අපි **animal** කියලා **object** එකක් හදාගන්නා



කියලා.ඒ වගේ වෙලාවක **thread** කිහිපයකින් එකම **object** එක **access** කරන්න උත්සාහ කරන්න පුළුවන්.ඒ කියන්නේ ඒකේ **method execute** කරන්න වගේ වැඩ.එකම වෙලාවට **eat()** එකයි **drink()** ආන්න ඒ වගේ **method run** කරන්න හදනවා වෙන්න පුළුවන්.

මේ වගේ වෙලාවක අපි ඒ **method** එක **synchronized** කියල නම් සඳහන් කරලා තියෙන්නේ, අපිට පුළුවන් **thread** කිහිපයකින් එකම වෙලාවකදී එකම **object** එකේ **method run** කිරීමට වගේ සැරසෙන එක නවත්තලා; එක **method** එකක් එකම වෙලාවක් තුල එක **thread** එකකින් පමණක් **access** කරන විදිහට සකස් කරගන්න.**method** වලට පමණමයි **synchronized** කියන **keyword** එක යොදන්න පුළුවන්.

Example :

```
public abstract void eat();

public synchronized void drink()
{
}
```

## native

මේ කියන්නේත් තවත් අමුතු ජාතියේ **keyword** එකක්.ඒකෙන් කරන්නේ වෙනත් **platform** වල ඒවායේ **native libraries** ලබාගෙන අපේ ජාවා **program** එකට හැකියාව දෙනවා ඒවා **access** කිරීමට.**method** වලට පමණයි මේකත් **apply** කරන්න හැකියාව තියෙන්නේ.අපි කලින් කලා කරපු **abstract** වගේමයි මේකෙන්..**method** එකේ **body** එකක් නෑ.ඒ වගේම ඉවර වෙන්නේ **semi colon "** ; "**එකකින්.**

Example :

```
native void operation();
```

## volatile

මේක නම් භාවිතා කරන්නේ අපේ **compiler** එකට **field** එකට අදාලව **optimization** එකට උත්සාහ කරන්නට අනවශ්‍ය බව දැනුම් දෙන්න.මොකද ඒක හේතුවෙනවා හිතාගන්න බැරි ප්‍රතිපල ලැබෙන්නටත්.විශේෂයෙන්ම එකම **field** එක **multiple threads** වලින් **access** කරන වෙලාවන් වලදී.**variable** වලට තමයි **apply** කරන්නේ **volatile** කියන **keyword** එක.

Example :

```
volatile static int myMarks;
```

## strictfp

සියලුම JVM වලින් අනන්‍ය වූ ප්‍රතිඵලයක් ලබාගැනුමට මේ keyword එක භාවිතා කරනවා. මේකෙන් බලකරනවා IEEE වලට අදාලව, දශමස්ථාන සම්බන්ධ ගණිතයන් සඳහා විධිමත් හැසිරීමක්. මෙය class, interface සහ method සඳහා apply කිරීමට හැකියි. මෙය method එකක් සඳහා යෙදූ කල එමගින් සහතික කරනවා අපි ලිවූ code එක විධිමත්ව ක්‍රියා කරන බවට.

## 13.6 Inheritance

Article by : [javaxclass.blogspot.com](http://javaxclass.blogspot.com)

**Inheritance** යනු ජාවාහි එන ප්‍රභල සංකල්පයක්. එමෙන්ම වස්තු පාදක වැඩසටහන් (OOP programs) ලිවීමේදී මෙය බොහෝ විට භාවිතා වනවා. C++ වැනි අනෙකුත් OOP සඳහා සහය දක්වන පරිගණක භාෂා වලත් මෙම සංකල්පය සඳහා සහය දැක්වූවද C++ හා Java අතර inheritance හිදී යම් වෙනස්කම් පවතිනවා.

දැම් අපි බලමු මොකක්ද මේ inheritance ක්‍රියාවලිය කියලා, Inheritance යනු යම් class එකකින් සාදන ලද වස්තූන්(objects) වෙනත් class එකකට අයත් objects වල ගුණාංගයන්ද අයත් කර ගැනීමේදී සිදුකරන ක්‍රියාවලියයි. එම නිසා මෙහිදී අදාල ලක්ෂණ අත් කරගත් class එක සහ ලක්ෂණ අත් කරගැනීමට බඳුන් වූ class එක සඳහා පොදු ලක්ෂණ එකක් හෝ කිහිපයක් පවතිනවා. අදාල ලක්ෂණ අත් කරගත් class එක child class එක නොහොත් sub class එක ලෙසත් ලක්ෂණ අත් කරගැනීමට බඳුන් වූ class එක parent class එක නොහොත් super class එක ලෙසත් හැඳින්වෙනවා. මෙම ක්‍රියාවලියේදී extends නම් ජාවා keyword එක භාවිතා වනවා. යම් class එකක් Inherit කිරීමෙන් සාදන ලද නව class එක එහි parent class එකෙහි attributes සහ behavior යනාදිය උකහා ගන්නා නිසා මෙහිදී ඉබේම software re-usability නම් සංකල්පයද ක්‍රියාත්මක වනවා. නමුත් inherit කිරීමෙන් සාදාගන්නා ලද නව class එකෙහි එහි parent class එකේ නොමැති attributes සහ behaviors අන්තර්ගත වීමට පුළුවන්.

### Important Facts :

- ✚ අලුතින් සාදන සෑම class එකක්ම අනිවාර්යයෙන්ම වෙනත් class එකක් extend (inherit) කල යුතුය.
- ✚ extend කරන්නේ කුමන class එකක්ද යන්න විශේෂයෙන් සඳහන් කර නැති විට Object යන class එක inherit වීම සිදුවේ. Object යනු ජාවා class hierarchy එකේ උඩින්ම ඇති super class එකයි.
- ✚ inherit කිරීමකදී කිසිවිටෙකත් constructors උකහා ගැනීම සිදුනොවේ. සියලුම constructors එය define කරන ලද class එක සඳහා විශේෂ වේ. (constructor පිළිබඳ අපි මීට පෙර පාඩමකින් සාකච්ඡා කර තිබේ එම පාඩම සඳහා මෙතන click කරන්න)

✚ යම්කිසි sub class එකක් construct වීමේදී එම class එකෙහි super class එකෙහි constructor එක මුලින්ම call වීම සිදුවේ.

✚ ජාවාහිදී ඕනෑම class එකකට තිබිය හැක්කේ එක් parent class එකක් පමණි. එනම් multiple inheritance සඳහා ජාවා සහය නොදක්වයි. එමනිසා multiple inheritance මගින් සිදුවන කායීය ජාවාහිදී සිදුකරන්නේ interfaces භාවිතයෙනි. අපි ඉදිරි පාඩමකින් ජාවා interfaces පිළිබඳ අධ්‍යයනය කරමු. ජාවා multiple inheritance සඳහා සහාය නොදක්වුවද C++ වලදී නම් multiple inheritance සඳහා ඉඩකඩ සලසා තිබේ.

Syntax :

```
class clid_class/sub_class extends perants_class/main_class
{
}
}
```

Example :

```

class phone
{
    protected String brand_name;

    protected String model_name;

    void call()
    {
        // Code Here
    }
    void message()
    {
        // Code Here
    }
    void time()
    {
        // Code Here
    }
}

class New_Phone extends phone
{
    int wifi_address;

    String Camara_megapixel;

    void music()
    {
        //Code Here
    }
    void GPRS()
    {
        //Code Here
    }
}

```

Super Class / Parents Class

Sub Class / Child Class

ඉහත ඡායා වැඩසටහන අධ්‍යයනයෙන් ඔබට inheritance පිළිබඳ තවදුරටත් වටහාගත හැකිවනු ඇත. එහි class variables සඳහා protected access එකක් ලබාදීමෙන් සිදුකරන්නේ එම data members සඳහා ප්‍රවේශවීම(access) අදාළ class එක තුළදී සහ එහි sub classes වලට පමණක් සීමා කිරීමයි.

## 13.6 Method Overriding and Overloading

### 13.7

#### What is Method overloading ?

ඔබ ඔබගේ IDE එක open කරලා එකම class එකේ එකම නමින් method දෙකක් තියන්න පුළුවන්ද බලන්න. `error(figure : 01)` එකක් එනවා ඔයාට පෙනෙයි. ඒ කියන්නේ එකම නමින් එකම class එකේ method දෙකක් තියන්න බැහැ.

Example :

```
public class Over_loading
{
    public static int demo_method(int num1, int num2)
    {
        int ans = num1 + num2;

        return ans;
    }
    public static int demo_method(int num1, int num2)
    {
        int ans = num1 + num2;

        return ans;
    }
}

public class demo
{
    public static void main(String[] args)
    {
        Over_loading ol = new Over_loading();

        int num1 = 10, num2 = 20;

        ol.demo_method(num1,num1);
    }
}
```

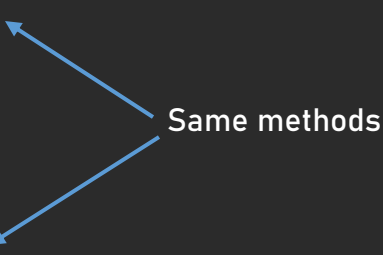


figure 01:

```
D:\Projects\IntelliJ\src\Over_loading.java:9:23
java: method demo_method(int,int) is already defined in class Over_loading
```

ඒ වුනත් එකම නමින් තියන **method** දෙකක් අරන් **method** එකකට අපි **parameter** එකක් දීලා අනිත් **method** එකට **parameter** එකක් නොදී **code** එක ලියලා **run** කරොත් ඔයාලට පෙනෙයි **error** එකක් එන්නේ නෑ කියලා. මෙන්න මේ ක්‍රියාවට කියනවා අපි **method overloading** කියලා.

Example :

```
public class Over_loading
{
    public static int demo_method(int num1, int num2)
    {
        int ans = num1 + num2;

        return ans;
    }
    public static double demo_method(double num1, double num2)
    {
        double ans;

        ans = num1 + num2;

        return ans;
    }
    public static double demo_method(double num1)
    {
        return num1;
    }
    public static void demo_method()
    {
        System.out.println("That Method Is Demo !! ha ha ha");
    }
}

class demo
{
    public static void main(String[ ] args)
    {
        Over_loading ol = new Over_loading();

        int num1 = 10, num2 = 20;

        double num3 = 8.1, num4 = 6.2;

        System.out.println(ol.demo_method(num1,num1));
    }
}
```

```

System.out.println(ol.demo_method(num3,num4));

System.out.println(ol.demo_method(num4));

ol.demo_method( );
}
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"
20
14.3
6.2
That Method Is Demo !! ha ha ha

```

දැන් අපි මේ code එක run කරද්දි argument එකක් නැතුව දුන්නොත් argument ඉල්ලලා නැති method එක call වෙනව.එක argument එකක් දුන්නොත් එක argument එකක් දීලා තියන method එක call වෙනවා.argument දෙකක් දුන්නොත් argument දෙකක් ඉල්ලලා තියන method එක call වෙනවා.

## What is method overriding?

මෙතනදි වෙන්තේ super class එකේ සහා sub class එකේ එකම නමින් method දෙකක් තියනව ඒ වුනාට මේ method දෙක ඇතුලේ තියන statements එකිනෙකට වෙනස්.

උදාහරනයක් විදියට අපි අපිවම ගමු.අපි පරිණාමය වෙලා තියෙන්නෙ වදුරන්ගෙ කියලා ඔයාලා අහලා ඇතිනේ.අපි වදුරව එක class එකක් විදියටත් මනුස්සයව තව class එකක් විදියට ගමු.එතකොට අපේ super class එක වෙන්තේ වදුරා.

වදුරන්ට සහ අපිට දෙගොල්ලොන්ටම ගස් නගින්න පුළුවන්තේ.ඒ වුනාට වදුරො නගින විදියටමද අපි ගස් නගින්නේ ?

නෑ නේද.ඒ කියන්නෙ climb කියන method එක දෙගොල්ලොන්ට බලපාන්නේ දෙවිදියකට.අන්න එතකොට තමා method override වෙනවා කියන්නේ.

Example :

```

class monkey // super class
{
    String climb()
    {
        String text = "climb with 4 hands and tail";
    }
}

```

```

        return text;
    }
}

class men extends monkey // sub class
{
    String climb()
    {
        String text = "climb with 2 hands and help of legs";

        return text;
    }
}

```

මේ උදාහරණයේ විදියට climb කියන method එක override වෙලා තියනවා. අපි man class එකේ object එකක් හදලා එකට climb method 1 call කරොත් call වෙන්නේ man class එකේ තියන climb method එක.

Example:

```

class human
{
    public void functions()
    {
        System.out.println("Dating With Some One");
    }
}

class boy extends human
{
    public void functions()
    {
        System.out.println("Trying Hookup With Some One");
    }
}

class example
{
    public static void main(String[] args)
    {
        human hum = new human();

        hum.functions();

        boy kid = new boy();
    }
}

```



```
    kid.functions();  
}  
  
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"  
Dating With Some One  
Trying Hookup With Some One  
  
Process finished with exit code 0
```

## 13.7 Upcasting and Downcasting

### What is upcasting?

අපි කලින් කතා කළා මතක ඇති inheritance ගැන, එක class එකක උදව්වට තවත් class එකක් extend කරන එක inheritance තමා කියන්නේ. එතකොට උදව්වට ගන්න class එක super class විදියටත් උදව්ව ගන්න class එක sub class කියලත් හඳුන්වනවා.

හරි දැන් මොකක්ද upcast කරනවා කියන්නේ බලමු. super class එකේ reffarance variable එකකට sub class එකේ object එකක් දාන එකට අපි upcasting කියලා කියනවා.

Example :

```
import java.lang.*;

class calculator
{
    void addition(int x,int y) // addition
    {
        int total = x + y;

        System.out.print("Total is : " +total+"\n");
    }

    void subtraction(int x,int y) //subtraction
    {
        int ans = x - y;

        System.out.print("Subtraction is : " +ans+"\n");
    }

    public void multiplication(int x,int y) //multiplication
    {
        int ans = x * y;

        System.out.print("Multiplication is : " +ans+"\n");
    }

    void division(int x,int y) //division
    {
        int ans = x / y;

        System.out.print("Division is : " +ans+"\n");
    }
}
```

```
class scientific_calculator extends calculator
{
    void logs()
    {
        System.out.println("This is Child Class");
    }

    void operation(int number)
    {
        System.out.println("Value : "+number*2);
    }
}

public class teacher
{
    public static void main(String args[])
    {
        int num = 10,num1 = 20;

        calculator cal = new scientific_calculator();

        ((scientific_calculator) cal).logs();
        ((scientific_calculator) cal).operation(num);

        cal.addition(num,num1);

        cal.subtraction(num,num1);

        cal.division(num,num1);

        cal.multiplication(num,num1);
    }
}
```



Upcasting

### Expected Output:

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"  
This is Child Class  
Value : 20  
Total is : 30  
Subtraction is : -10  
Division is : 0  
Multiplication is : 200  
  
Process finished with exit code 0
```

### Example:

```
class test  
{  
    void m1()  
    {  
        System.out.println("m1 method in class One");  
    }  
}  
class Two extends test  
{  
    void m2()  
    {  
        System.out.println("m2 method in class Two");  
    }  
}  
public class one  
{  
    public static void main(String[ ] args)  
    {  
        test o = (test)new Two(); // Upcasting. Here, super class reference o refers to sub  
class object.  
  
        o.m1();  
        // o.m2(); // Compile-time error message.  
    }  
}
```

Expected Output:

```
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"  
m1 method in class One  
  
Process finished with exit code 0
```

## What is Downcasting?

**downcast** කරනවා කියන්නේ අපි උඩ **upcast** කරපු **object** එක ආයෙත් **sub class** එකේ **variable** එකකට දා ගන්න එකට.

Example :

```
import java.lang.*;  
  
class calculator  
{  
    void addition(int x,int y) // addition  
    {  
        int total = x + y;  
  
        System.out.print("Total is : " +total+"\n");  
    }  
  
    void subtraction(int x,int y) //subtraction  
    {  
        int ans = x - y;  
  
        System.out.print("Subtraction is : " +ans+"\n");  
    }  
  
    public void multiplication(int x,int y) //multiplication  
    {  
        int ans = x * y;  
  
        System.out.print("Multiplication is : " +ans+"\n");  
    }  
  
    void division(int x,int y) //division  
    {  
        int ans = x / y;  
  
        System.out.print("Division is : " +ans+"\n");  
    }  
}
```

```

    }
}

class scientific_calculator extends calculator
{
    void logs()
    {
        System.out.println("This is Child Class");
    }

    void operation(int number)
    {
        System.out.println("Value : "+number*2);
    }
}

public class teacher
{
    public static void main(String args[])
    {
        int num = 10,num1 = 20;

        calculator cal = new scientific_calculator();

        ((scientific_calculator) cal).logs();

        ((scientific_calculator) cal).operation(num);

        cal.addition(num,num1);

        cal.subtraction(num,num1);

        cal.division(num,num1);

        cal.multiplication(num,num1);

        scientific_calculator cal_science = (scientific_calculator) cal; // downcasting

        cal_science.logs();

        cal_science.operation(num1);
    }
}

```

```
}  
}
```

## 13.8 Polymorphism

හොඳයි අපි මේ මාතෘකාවට එන්න කලින් වැදගත්ම දෙයක් තමයි **class inheritance, upcasting, method overriding** කියන හැම මාතෘකාවක්ම දැනන් ඉන්න එක.ඒවා මතක නැත්නම් කලින් වලින් ඒක බලා ගන්න.

**super class** එකක **method** එකක් **upcast** කරලා ඒ **upcast** කරපු **reference variable** එක හරහා **sub class** එකේ **method** එකක් **call** කරන එක තමා මේ **polymorphism** කියන්නේ.

අපි මේක තවත් පැහැදිලි කරගන්න උදාහරණයක් ගමු:

අපි වාහන **park** එකකට **small software** එකක් ලියමු.මේ වාහන **park** එකට **car,van,bike** කියන වාහන තුන් වර්ගයක් එනවා කියලා ගමු.එතකොට මේ වාහන තුන් වර්ගය **park** කරන්න ඕනි තුන් විදියකටනෙ.අපි මේක **auto** වෙන විදියට **code** එකක් ලියමු.

මුලින්ම අපි අපේ **main method** එකයි වාහන වර්ග තුනටයි **class** ටික හදා ගමු.

```
public class park  
{  
    public static void main(String[] args)  
    {}  
}  
class car {}  
  
class van {}  
  
class bike {}
```

අපි හිතමු මොනාම හරි ක්‍රමේකට **parking method** එක ඇතුළට වාහනේ ආවා කියලා දැන ගෙන **method** එක **call** වෙනවා කියලා.මේ **class** තුනට අපි **object** තුනක් හදලා **method** ටිකත් **call** කරමුකො.

```
class car  
{  
    void parking()  
    {
```

```

    }
}

class van
{
    void parking()
    {

    }
}

class bike
{
    void parking()
    {

    }
}

public class park
{
    public static void main(String[] args)
    {
        car c = new car();

        c.parking();

        van v = new van();

        v.parking();

        bike b = new bike();

        b.parking();
    }
}

```

දැන් මෙතන පොඩි අවුලක් තියනවා. ඉස්සෙල්ලම **bike** එකක් ආවත් **call** වෙන්නේ **car** එකේ **parking method** එක. මොකද **code** එක **compile** වෙද්දි මුලින්ම ගන්නෙ **car** එකේ **method** එකනි. මෙන්න මෙතනදි තමා අපිට **polymorphism** කියන **concept** එක ඕනි වෙන්නේ.



Example:

```
class vehicle
{
    void parking()
    {
        System.out.println("Parking");
    }
}

class car extends vehicle
{
    void parking()
    {
        System.out.println("Parking Car");
    }
}

class van extends vehicle
{
    void parking()
    {
        System.out.println("Parking Van");
    }
}

class bike extends vehicle
{
    void parking()
    {
        System.out.println("Parking Bike");
    }
}

public class park
{
    public static void main(String[] args)
    {
        vehicle vel = new bike();

        vel.parking();

        vehicle vel2 = new car();

        vel2.parking();
    }
}
```

```

        vehicle vel3 = new van();

        vel3.parking();
    }
}

```

Expected Output:

```

"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"
Parking Bike
Parking Car
Parking Van

Process finished with exit code 0

```

## Encapsulation

මේ කියන්න යන්නේ OOP වල එන ප්‍රධානම ලක්ෂණයක්. යොදාගන්නේ instance වර්ගයේ විචල්‍යයන් ආරක්ෂා කර ගැනීමයි. මෙහිදී හඳුන්වාදෙන විචල්‍යයන් අපි **private** කියලා තමයි හඳුන්වා දෙන්නේ. **Data hiding** කියන වචනෙන් පාවිච්චි කරන්නෙන් ඒකමයි. ඒ කියන්නේ අපි **private** කියලා විචල්‍යයන් හඳුන්වා දුන්නාම ඒක **access** කරන්න පුළුවන් වෙන්නේ ඒ **class** එක ඇතුළේ ඉඳලා විතරමයි. **class** එකෙන් පිටත් ඉඳන් අපිට බෑ එම විචල්‍යයන් **access** කරන්න. එහෙම හඳුන්වාදුන් විචල්‍යයන් **access** කිරීමට **public** කියන **modifier** එක යොදලා හදලා දීලා තියෙනවා **getter** සහ **setter** .[getter කියන්නේ **data** ගන්න ඕනෙ වෙලාවට කථා කරන **method** එක. **setter** කියන්නේ අපිට **data set** කරන්න අවශ්‍යනම් භාවිතා කරන්නක්..]. මේවාට වෙනමම වචන දෙකක් තියෙනවා **accessor** සහ **mutator** කියලත්. මෙමගින් පුළුවන් අපිට ඒ **protected** විචල්‍යයන් **access** කරන්න. **method** වලටනම් **java beans naming conversion** වල කියන රීතීන් ටික පිළිපදින්න ඕනේ.

Example for getter and setter :

```

public class encapsulation
{
    private int speed;

    private int distance;
}

```

```

    public void setSpeed(int data) // set data
    {
        this.speed = data;
    }

    public int getSpeed() // access data
    {
        return speed;
    }

    public void setDistance(int length)
    {
        this.distance = length;
    }

    public int getDistance()
    {
        return distance;
    }
}

class Demo // Demo class
{
    public static void main(String[] args)
    {
        encapsulation encap = new encapsulation();

        encap.setSpeed(100);

        encap.setDistance(90);

        System.out.println(encap.getSpeed());

        System.out.println(encap.getDistance());
    }
}

```

ඉතින් මොකක්ද මේ කියන **encapsulation** එකේ ඇති වටිනාකම.? මේ ටිකත් එහෙනම් බලලාම ඉන්නකො..අපිට මේ යටතේ පුළුවන් **class** එකක තියෙන **fields** **read only** ඒ කියන්නේ කියවීමට පමණක් හෝ **write only** ඒ කියන්නේ ලිවීමට පමණක් ලෙසට සකසන්න පුළුවන්.එමගින් **class** එකක **fields** කෙරේ දත්ත වලට සිදුකරන **store** කිරීම් සම්පූර්ණ පාලන හැකියාවක් මෙමගින් ලැබෙනවා..මේ විදියෙන් ඉතින් අපි වැඩ කරනකොට වාසිය තියෙන්නේ අපේ **encapsulation** විදිහට හදපු **class** එකේ **user** ක්ෂේටියට පෙන්න්නේ **getter setter** විතරයිනේ.ඉතින් එයාල දන්නේ ඒකෙන් මොනවාහරි

අගයක් එවන්න විතරනේ.හැබැයි ඉතින් **call** එකකදි කොහොමද කරන්නේ කියන එක දන්නේ අදාල **class** එක හදපු **dial** එක තමයි.උදාහරණයකුත් එක්ක කියනවා නම් ඉතින් ගන්න අපි ඉස්සෙල්ලා **encapsulation class** එකේ අපි දුන්නේ **setSpeed** කියලා **method** එකක්.ඒකෙන් දුන්නට අපි දන්නේ නෑනේ හැබැට අපි **pass** කරන අගයම **set** වෙනවද එහෙමත් නැත්නම් ඒකට කීයක් හරි එකතු හරි අඩු හරි වෙලා **set** වෙනවද කියලා.අපි බලපු උදාහරණයේදි නම් කරල තිබ්බේ.

```
public int getSpeed() // access data
{
    return speed;
}
```

කියලානේ.හැබැයි එතන මෙහෙම තිබ්බනම්.

```
public int getSpeed()
{
    return (speed+150) * 2;
}
```

අපි දන්නේ නෑනේ.අපි දන්නේ **set** කරන වැඩේ විතරයිනේ.එතකොට අනිත් **part** එක කරන එක්කෙනා **encapsulation** දාල හදනවා.ඉතින් ඒ **data** වල ආරක්ෂාව ඇති වෙනවා ඉබේටම. තව..class එකේ **field** එකක **data type** එක උනත් වෙනස් කරන්න පුළුවන්, ඉතින් එතකොට **user** ලාට ඕනේ වෙන්නේත් නෑ තමන්ගේ **code** එක වෙනස් කරන්න.මේ වගේ කාරණාවන් එක්ක බලනකොට අපිට **maintainability, flexibility, extensibility** කියන ගුණාංග අඩංගු කරගන්න පුළුවන් **encapsulation** නිසා.

## 14. Java I/O Streams

සියලුම පරිගණක භාෂාවන් හි **Input** සහ **Output** ක්‍රියාවලීන් සිදුකර ගැනීමට කිසියම් යාන්ත්‍රණයක් පවතිනවා. උදාහරණයක් වශයෙන් කිසියම් **calculation** එකක් සඳහා ලියන ලද පරිගණක වැඩසටහනකට දත්ත **input** කිරීමට සහ එහි ප්‍රතිඵලය පිටතට ලබාදීම සඳහා යොදාගන්නා ක්‍රම සැලකිය හැකිය.

ජාවා වැඩසටහන් තුළදී **Input** සහ **Output** ක්‍රියාවලීන් සිදුකරගැනීමට විවිධ ක්‍රම පවතී ඒවා නම් **Streams, Files, Channels** යනාදී වශයෙනි. අපි මෙහිදී වැඩිදුරටත් සාකච්ඡා කරනුයේ ජාවා තුළ යෙදෙන **Streams** පිළිබඳවයි.

ස්ට්‍රීම්(**Streams**) මගින් විවිධ **Input** සහ **Output Sources**(ප්‍රභව) නියෝජනය වීම සිදුකෙරේ. එනම් පරිගණක ක්‍රමලේඛකයාට **streams** භාවිතයෙන් **input / output sources** සමග සන්නිවේදනය කළ හැකිය. උදාහරණ ලෙස **Disk Files, Sockets, Devices** සහ වෙනත් **Process** එකක් වුවත් විය හැකිය.



අපි දැන් බලමු ස්ට්‍රීම් එකක ස්වභාවය කුමක්ද කියල. ස්ට්‍රීම් එකක් කිව්වම අපිට එකපාරට මතක් වෙන්නෙ ගලාගෙන යන ස්වභාවය නේ. ඒ වගේම තමයි ස්ට්‍රීම් එකක තිබෙන්නෙ කිසියම් දත්ත ප්‍රවාහයක් ජාවා තුළ ස්ට්‍රීම් භාවිතයේදී එම ප්‍රවාහය මූලික දත්ත ආකාරයෙන් පැවතිය හැකියි නැතිනම් එම ප්‍රවාහය **Object data** ප්‍රවාහයක් උනත් විය හැකියි. **Stream** එකක පවතින්නේ **data sequence** එකක්.

Example :

Binary stream : 10001110001110101

ඇතැම් ස්ට්‍රීම් එතුලින් දත්ත ගමන් කරවීමට සහය දක්වන අතර(උදා : **FileInputStream**,) ඇතැම් ඒවා එතුලින් යන දත්ත වෙනත් ආකාරයකට/ස්වභාවයකට පත්කරනු ලබයි(ex:**GZIPInputStream**, **DeflaterOutputStream**).

**Streams** තුලින් යවන්නාවූ දත්ත වල ස්වභාවය අනුව මූලික වශයෙන් ස්ට්‍රීම් වර්ග 2ක් හඳුනාගත හැකිය.

### 01.Character Streams

මෙමගින් මිනිසාට කියවිය හැකි ආකාරයේ දත්ත ගලා යාම සිදුකෙරේ(උදා : ඉලක්කම්, අකුරු යනාදිය)

### 02. Byte Streams

මේවායේ යවනුයේ **machine-formatted** ආකාරයේ දත්තවේ. මේවා පවතිනුයේ ද්වීමය(binary) ආකාරයෙනි(උදා:110110).

අදාලතාවය අනුව සුදුසු **Stream** ආකාරය තෝරාගත යුතුය. අනෙකුත් සියලුම ස්ත්‍රීම් වර්ගයන්හි පදනම වනුයේ **Byte Stream** වේ.

### සුදුසු Stream එක තෝරාගන්නේ කෙසේද?

- කිසියම් **data source** එකක් **low-level** මට්ටමෙන් ප්‍රවේශනය කිරීමට අවශ්‍ය විට **byte stream** එක සුදුසුය.උදා: **data file copy** කිරීම, **Audio files edit** කිරීම යනාදියේදී.
- **Data source** එක **text data** වලින් සමන්විත නම් සහ **text manipulations** අවශ්‍ය විටදී **character streams** භාවිතාකළ හැකිය.

Example :

```
import java.io.*;

public class java_input_output
{
    public static void main(String args[ ])
    {
        FileReader file_Readr = null;

        FileWriter file_Writer = null;

        try
        {
            file_Readr = new FileReader("Strings.txt");

            file_Writer = new FileWriter("Strings_02.txt");

            int tmp;

            while((tmp= file_Readr.read())!=-1)
            {
                file_Writer.write(tmp);
            }
        }
        catch(FileNotFoundException e)
        {
            System.err.println("File NOT Found : " + e.getMessage());
        }
    }
}
```

```

        catch(IOException e)
        {
            System.err.println("IO Exception : " + e.getMessage());
        }
        finally
        {
            try{
                if(file_Readr != null)
                {
                    file_Readr.close(); //close streams
                }
            }
            catch(IOException e)
            {
            }
            try
            {
                if(file_Writer != null)
                {
                    file_Writer.close(); // close streams
                }
            }
            catch(IOException e)
            {
            }
        }
    }
}
}

```

## Example Above Code :

FileReader සහ FileWriter යනු පිළිවෙලින් String.txt සහ String\_02.txt ට ලිවීමට තෝරාගත් Stream classes වේ. ඒවායින් සාදාගත් file\_Reader සහ file\_Writer objects යොදාගෙන source file එකෙන් කියවීම සහ destination file එකට ලිවීම සිදුකෙරේ. මේවා character streams නිසා කියවීම සහ ලිවීම සිදුකරනුයේ character ආකාරයට ය. while loop එක තුළදී සිදුකෙරෙනුයේ input stream එකෙන් character by character කියවා output stream එකට character by character ලිවීමයි.

එහිදී while loop එකේ condition එක වශයෙන් යොදා ඇත්තේ කියවන ලද character එකෙහි integer අගය -1 ට සමානදැයි බැලීමයි. එම කියවන ලද අගය -1 නම් එයින් සඳහන් වනුයේ අප සිටින්නේ EOF(End of the file) හි බවය(උයිල් එකෙහි අවසානයේ බවය). මේවා try block & catch block එකක් තුළ යෙදීමට හේතුව ස්ත්‍රීම් සමග ගනුදෙනු කිරීමේදී පැනනැගිය හැකි exception හසුකර ගැනීමයි. අවසානයේ

**open** කරගන්නා ලද ස්ත්‍රීම් වසාදැමීම සිදුකර ඇත.මෙසේ ස්ත්‍රීම් **close** කිරීම අනිවාර්යයෙන්ම කළ යුත්තකි. නැතහොත් අප ලියන වැඩසටහන් වල **resource leaks** ඇතිවීමට හේතුවේ. එය වැඩසටහනක ඇති විශාල ගැටලුවකි.

## Write On Text Files

```
import java.io.*;

public class input_stream
{
    public static void main(String[] args) throws FileNotFoundException
    {
        FileOutputStream fos = new FileOutputStream("name.txt",true);

        try
        {
            PrintWriter pw = new PrintWriter(fos);

            BufferedWriter bw = new BufferedWriter(pw);

            bw.write("SLIIT COMPUTING OF ICT");

            bw.newLine();

            bw.close();

            System.out.print("Input Completed !!!");

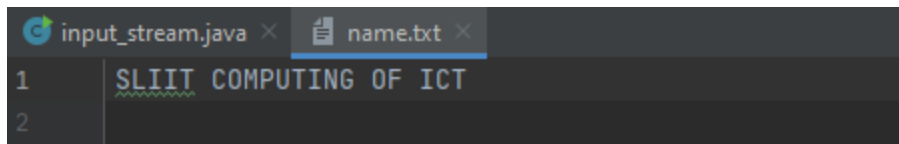
        }
        catch (Exception e)
        {
            System.out.print(e);

        }

    }
}
```

name.txt file :

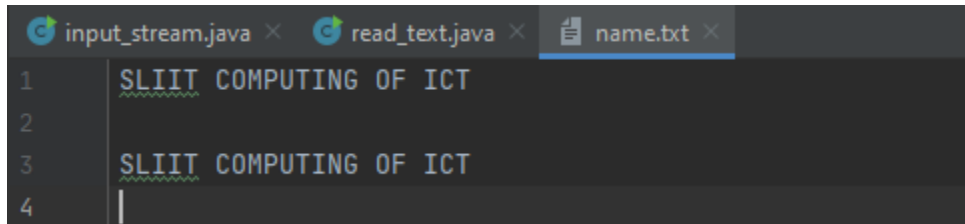




```
input_stream.java × name.txt ×  
1 SLIIT COMPUTING OF ICT  
2
```

## Read From Text File

Target Text File : (name.txt)



```
input_stream.java × read_text.java × name.txt ×  
1 SLIIT COMPUTING OF ICT  
2  
3 SLIIT COMPUTING OF ICT  
4
```

Example code : read\_text.java

```
import java.io.*;  
  
import java.util.Scanner;  
  
public class read_text  
{  
    public static void main(String[] args) throws FileNotFoundException  
    {  
        FileInputStream fis = new FileInputStream("name.txt"); //file input stream  
        try  
        {  
            Scanner print = new Scanner(fis);  
  
            while (print.hasNext())  
            {  
                System.out.print(print.nextLine() + "\n"); //print expected values  
            }  
        }  
        catch (Exception e)  
        {  
            System.out.print("File Not Found !!!! or"+ e);  
        }  
    }  
}
```

expected output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"  
SLIIT COMPUTING OF ICT  
  
SLIIT COMPUTING OF ICT  
  
Process finished with exit code 0
```

=====

Example 00 :

```
import java.io.*;  
import java.util.Scanner;  
public class example_00  
{  
    public static void main(String[] args) throws FileNotFoundException  
    {  
        FileOutputStream fos = new FileOutputStream("details.txt",true);  
  
        try  
        {  
            Scanner input = new Scanner(System.in);  
  
            System.out.print("Full Name : ");  
  
            String name = input.nextLine();  
  
            System.out.print("Age : ");  
  
            int age = input.nextInt();  
  
            PrintWriter pw = new PrintWriter(fos);  
  
            BufferedWriter bw = new BufferedWriter(pw);  
  
            bw.write(name+", "+age);  
  
            bw.newLine();  
        }  
    }  
}
```

```
        bw.close();

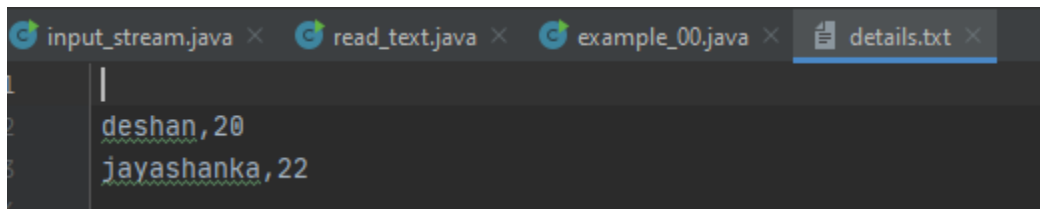
        System.out.print("Input Complete !!");
    }

    catch (Exception e)
    {
        System.out.print(e);
    }
}
}
```

Expected Output :

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
Full Name : jayashanka
Age : 22
Input Complete !!
Process finished with exit code 0
```

Details.txt file :

The screenshot shows an IDE window with four tabs: 'input\_stream.java', 'read\_text.java', 'example\_00.java', and 'details.txt'. The 'details.txt' tab is active and shows the following content:

```
1 |
2 | deshan,20
3 | jayashanka,22
```

Edit By : R.W.D.J Amarasinghe

E mail : [djayashanka750@gmail.com](mailto:djayashanka750@gmail.com)

Date : 2021 - 03 - 25