Java Notes _Tejus Sir
Siddhant_PJCM5

Siddhant Java

* Q> What is a Computer?

• 07/01/2022 Friday

• Computer is an electronic device which is madeup of hardware and software components, which has the capability or which can do certain task such as storing data, processing data and retrieve the data.

08/01/2022 Saturday

*** Hardware:-**

- Moniter
- Mouse
- Keyboard
- RAM
- ROM
- SMPS (Switch Mode Power Supply)
- Proccessors
- SSD (solid state drive)
- Graphics card.
- I/O devices
- Fans & speakers.

*** Software:-** • Operating system (OS).

* Q> What is a Language:-

• It is a mode of communication.

* Programming Language:-

• The language which is used to command a computer to do specific task, is known as a programming language.

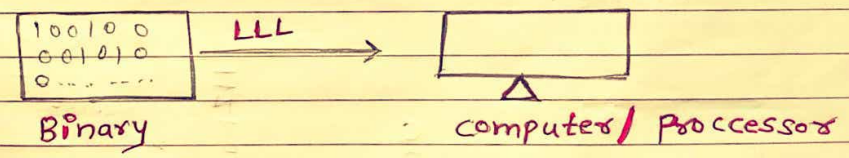
* The language which works the computer, is a "Binary language". i.e. 1 and 0.
• The computer knows binary language only. i.e. understanding language of computer.

* Levels of Programming language:-

• There are three levels in programming langauge.

* 1> Low level langauge / Machine level langauge:-

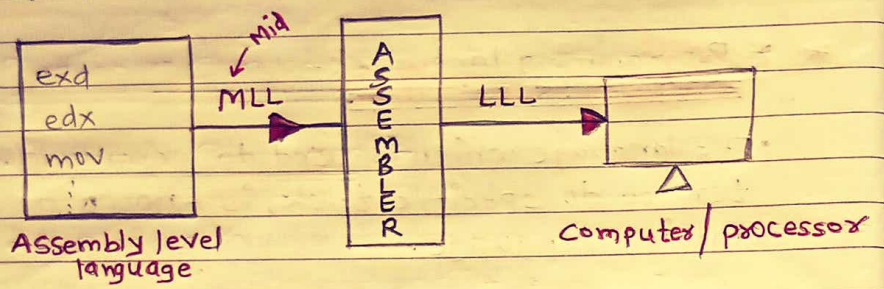
• The language which ~~was~~ is directly understandable to a computer, is known as the Low level or Machine level langauge.
• Ex:- Binary language. (0 & 1)



* 2> Mid Level language / Assembly level language:-

• The language which contains predefined set of words which is understandable to a computer, with an intermediate software known as Assembler.

Monday
10/01/2022



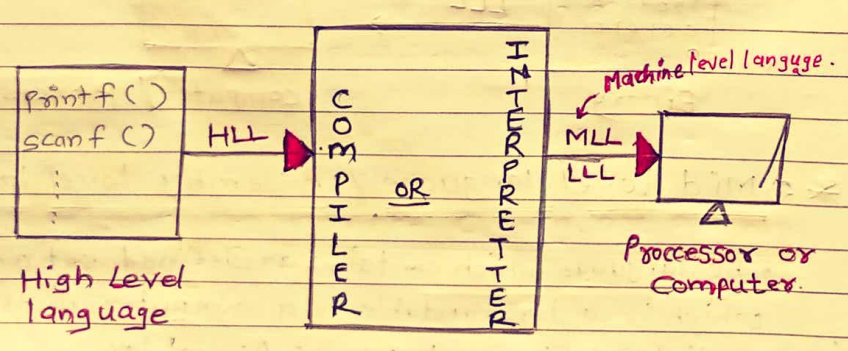
* What is an Assembler ?→

- Assembler is an intermediate software which is used to convert assembly level lang. to machine understandable language. i.e. into binary langaage.

* 3) High Level Language :-

- The language which is easily readable, understandable, instructable ~~was~~ by a programmer, was called as "High level language".

- i.e. The language which is closer and similar to human language, was called as "High level language".



* * Work of compiler / Interpretter :-

- compiler or interpretter is an intermediate software, which is used to convert High level language to machine level language.

* High level languages :-

- C
- C++
- Python
- Ruby
- Pearl / Perl
- C# (C-sharp)
- PHP
- Golang
- Kotlin
- JavaScript
- PHP
- Swift
- Java
- Dotnet.

* * Evalution of Java :-

* Tuesday
11/01/2022

- The father of Java language is James Gosling.

- 1st version of Java is "jdk 1.0". (1996)

- Latest version of Java is "jdk 17". (14 Sep. 2021)

- Stable version of Java is "jdk 1.8 or jdk 8", by Sun Microsystems. (2014)

- Owner of Java language is "Oracle". (since 2010)

Siddhant Java

Friday
14/01/2022

** Platform Dependency & Independency :-

* What is Platform ?:-

- A platform is a basic hardware and software on which applications will run.

- Softwares are classified in two types :-
  1) Platform dependent
  2) Platform Independent.

* 1) Platform Dependent Softwares :-

- The software developed can be used in only one specific platform, is known as the "Platform dependent software."

- Ex :- C and C++ softwares.

* 2) Platform Independent Software :-

- The softwares developed on one platform, can be used in any platform, such software are called as "platform Ind. softwares."

- Ex :- Java, Python, softwares.

* Why software made of C lang. are platform dependent ? :-

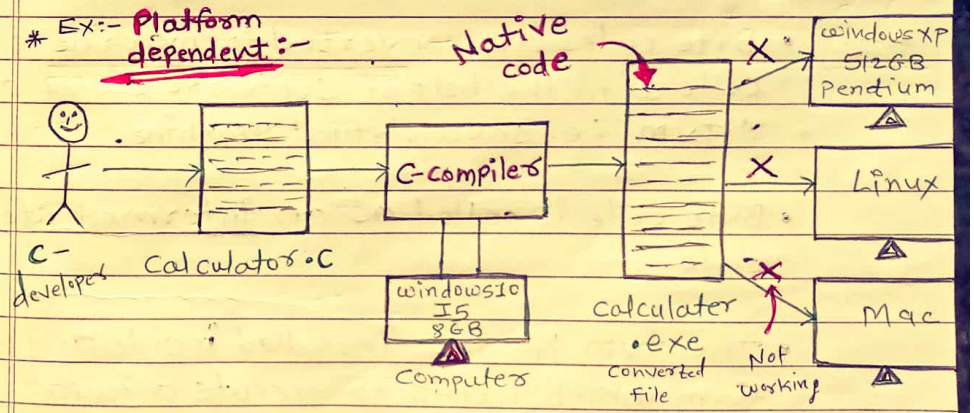
- When a program written & compiled in C & C++ the code is directly converted to machine understandable langauge.

- This code is generated as ".exe" file.

- The .exe file generated can run only on one specific platform.
- i.e. When a program is compiled in windows operating system, the exe file can run only in windows op. system, but not on Linux O.S. or Mac op. system or Solaris O.S.

- C-compiler converts C language directly to "native machine level language".
- This is restricted to one type of platform.

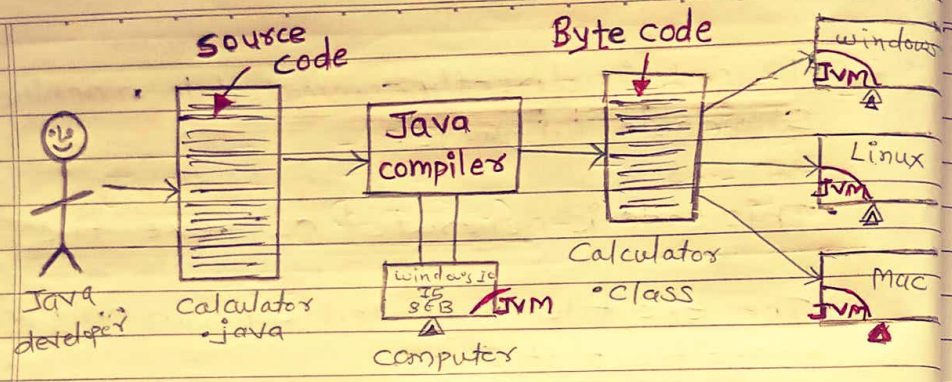
* Ex :- Platform dependent :-



* Why softwares made in using Java, are platform independent :-

- When a program is written using java and compiled, the java compiler creates an the

Siddhant Java



intermediate file, known as "class file".

- This ~~class~~ class file consist "Byte codes".

- Byte codes are non-executable codes, i.e. it is neither understandable to programmer nor the computer.

- Byte codes are converted to machine level code with the help of software called as "JVM" i.e. Java Virtual Machine.

- Byte code is called as an intermediate code.

- The JVM has to be installed inside any platform which needs to execute a java application.
- Hence, we can achieve platform independency using Java.

Tuesday
18/01/2022

* JDK Architecture :-

* JVM :-

- JVM stands for Java Virtual Machine.
- It is an interpretter, which reads .class files (bytes codes) line by line and it executes.

- JVM is present inside JRE.

* JRE :-

- JRE stands for Java Runtime Environment.
- It is used only to run Java codes.
- If we want to just run a java application, we can use JRE.

- JRE consist of JVM as well as some Built in library files, to run the java codes.

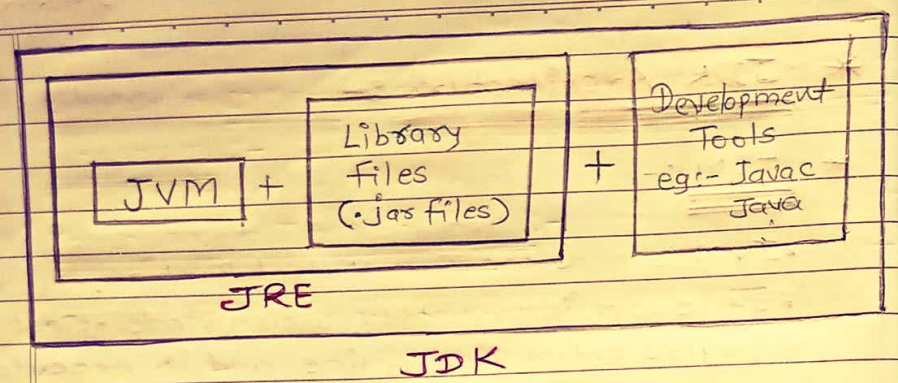
* JDK :-

- JDK stands for Java Development Kit.
- It is a Package provided by Oracle community.

- Basically, we use JDK to develop Java applications.
- It consist of JRE as well as software tools used to develop java applications.

Siddhant Java

Thursday
20/01/2022



* Commands of Command Prompt :-

*** 1) Cd folder name :-**

- It is used to change the current working directory.

*** 2) dir :-**

- This command is used to explore or open the current working directory.

*** 3) Cd .. :-**

- This command is used to change the directory from child folder back to the parent folder.

*** 4) drive name : :-**

- It is used to change from one drive to the another drive.

*** 5) Cls :-**

- This command is used to clear the screen.

*** 6) mkdir folder name OR md folder name :-**

- This command is used to create a new folder.

* Terminologies :-

*** 1) Work directory :-** (e.g. A, B, C OR D).

- The folder which currently in use, is called "work directory".

*** 2) Relative Path :-** (e.g. A→B, B→C OR C→D)

- If the address to the destination is given from current working directory, it is known as "relative path".

*** 3) Absolute Path :-** (e.g. A→D ..... (Total))

- It is the complete path starting from the drive name (root folder) to the destination.

* Relative path to a destination changes from location to location.

* Absolute path for a destination is same from any location.

Siddhant Java

Siddhant Java

* Steps to create, compile, execute the Java Program :-

* 1) To create a Java program :-

- To create a java program, we need some editors such as :-

Basic Editor	IDE (Integrated Dev Int. environment)
• Notepad	• Eclipse / eclipse
• word	• Net beams
• Notepad ++	• Intellij
• Edit plus	• VS code
• Sublime	• Atom

- After, writting the java program, we need to save java file with ".java" extension.

* 2) Generating .class files :-

- Compile the .java file in order to get .class files.

* Work of a compiler :-

1) If the syntax (rules) are correct, the compiler will generate .class files.

2) If syntax (rules) are incorrect, the compiler does not creat .class files, instead it shows "compile time error".

* 3) To execute .class files :-

- Here, execute the .class files to run java program.

* Structure of Java program :-

* 1) Class Block :-

a) Variables.
b) Methods.

* 2) Method Block :-

a) statements

*** e.g. :-**

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

Class Block

Method block

a) * **Syntax of Print statement :-**

- System.out.println (data/value) ;

b) * **Command to compile a java program :-**

- javac FileName.java

c) * **Command to Execute a class File :-**

- java FileName

* **Program :- (Basic Hello World Program) :-**

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

* **Command Prompt steps :-**

1) Compile :- c → Program → javac FirstProgram.java

2) Execute :- c → program → java FirstProgram

* **Introduction Program :-**

```
Class Introduction
{
    public static void main (String [] args)
    {
        System.out.println ("Name - S.R.Shinde") ;
        System.out.println ("Qualification - B.E. Mech") ;
        System.out.println ("college - NMCE, Peth") ;
        System.out.println ("Year of passout - 2019") ;
        System.out.println ("Native place - Sangli") ;
        System.out.println ("Expected Salary - 4.5 to 5 LPA") ;
    }
}
```

* **Command Prompt steps :-**

1) Compile :- c → Program → javac Introduction.java

2) Execute :- c → Program → java Introduction.

Siddhant Java

Saturday
22/01/2022

** Tokens :-

- Tokens are the smallest unit of a programming language, which helps the programmer to create programs, (Instructions).

- Tokens are of two types :-
  1) Keyword
  2) Identifier.

* 1) Keyword :- (51+ words)

- Compiler aware words are called "Keywords".
- Keywords have predefined meaning
- A Keyword cannot be altered by programmer.
- All keywords are written in lower case.
- We cannot use a keyword for any other
- purpose.

* e.g. :- class, public, static, void, etc.

* 2) Identifier :-

- Names given by the programmer, to the component of java, such as class name, method name, variable name, package name, interface name, etc are called as the "Identifiers".

* Rules to name an Identifier :-

Monday
24/01/2022

- **NOTE :-** Rules are mandatory to follow, else we get errors.

1) An identifier cannot start with the number
   - e.g. :- 1Program ✗ , Program1 ✓.

2) Character spaces are not allowed while naming an identifier.
   - e.g. :- First Program ✗ , FirstProgram ✓.

3) Special characters are not allowed while naming an identifier, other than, $ and _ .
   - e.g. :- a) @Program ✗ , $program ✓
     b) Program* ✗ , my_Program ✓

4) We cannot use a keyword in place of an identifier.

** Conventions to name an Identifier :-

- They are nothing but the culture to be followed, while naming an identifier. (Industry std.)

- Conventions are not checked by compiler, i.e. we dont get any compile time error, if conventions are not followed.

- But, it is highly recommanded, to follow the conventions.

Siddhant Java

Siddhant Java

* Conventions for Class Name :-

1) If the class name is single word, the first character of that word must be in the upper case.

- e.g.:- a) program ; introduction ✗
  b) Program ; Introduction ✓

2) If the class name is multi-word, then every first character of the word must be in upper case.

- e.g.:- a) myfirst program ✗
  b) MyFirstProgram ✓

* Conventions to Name the Methods :-

1) If method name is single word, the first character of that word must be in lower case. • e.g.:- main ( ), addition ( ).

2) If method name is multiword, then the first character of first word must be in lower case and from the second word, every first character must be in the upper case.

- e.g.:- a) additionOfTwoNumbers ( ).

- This style of writing is known as "camel case." ( ).

Tuesday
25/01/2022

* * Literals (Data) :-

Literals

- Single Value Data (Primitive data)
  - a) Numbers.
  - b) Characters.
  - c) Boolean, etc.
- Multi-Value Data (Non-Primitive data)
  - a) String.
  - b) Arrays.
  - c) Class, etc.

* 1) Single Value Data :-

* a) Numbers :- 1) Integers → -2, -1, 0, 1, 2, 3 ....
  2) Decimals → -1.5, -0.5, 1.7, 8.9, ...

* b) Characters :- 1) Digits → (0-9)
  2) Alphabates → Upper Case (A-Z)
  → Lower case (a-z)
  3) Special characters. (&, *, #, $ .....)

* c) Boolean :- 1) true.
  2) False.

* Decimal to Binary :-

1) $(10)_{10}$ →
2 | 10
2 | 5 → 0
2 | 2 → 1
1 → 0

∴ $(10)_{10} \rightarrow (1010)_2$

2) $(31)_{10}$ →
2 | 31
2 | 15 → 1
2 | 7 → 1
2 | 3 → 1
2 | 1 → 1

∴ $(31)_{10} \rightarrow (11111)_2$

Sundaram

Wednesday
26/01/2022

* Binary to Decimal :-

1) $(1010)_2 \rightarrow (?)_{10}$ => 1 0 1 0

$0 \times 2^0 = 0$
$1 \times 2^1 = 2$
$0 \times 2^2 = 0$
$1 \times 2^3 = 8$
10

$\therefore \boxed{(1010)_2 = (10)_{10}}$

2) $(11111)_2 \rightarrow (?)_{10}$

1 1 1 1 1

$1 \times 2^0 = 1$
$1 \times 2^1 = 2$
$1 \times 2^2 = 4$
$1 \times 2^3 = 8$
$1 \times 2^4 = 16$
31

$\therefore \boxed{(11111)_2 = (31)_{10}}$

* Primitive Data :- (Basic data) :-

- Primitive data are of three types as :-
  1) Numbers
  2) ~~Alphabat~~ characters.
  3) Boolean.

* 1) Numbers :-
a) Integers → -1, -2, 0, 1, 2 ....
b) Decimal → -1.5, -5.2, 0.5, 2.5 ....

* 2) Characters :-

- Characters are enclosed with single quote. ' '
- e.g. :- 'a', '$', 'z', '#'.

- Characters are divided into :-
  1) Alphabates
  2) Digits
  3) Special characters.

1) Alphabates :- a) Upper case → ASCII (65-90)
b) Lower case → ASCII (97-122)

2) Digits :- 0-9 = ASCII (48-57)

3) Special characters :-

- Other than alphabates and digits, the rest all are called as "special characters".

* 3) Boolean :-

- Boolean is used to represent logical values.
- It has two keywords as :-
  1) true
  2) false

* * String :-

- Group of characters are called as "string".
- Length of the string can be anything.
- They must be enclosed with " " double quotes & they case sensitive.
- e.g. :- a) "Hello World".
  b) "siddhant R. Shinde".

Siddhant Java

Siddhant Java

* Program :-

```
class Introduction1
{
public static void main (String [] args)
{
System.out.println ("Siddhant R. Shinde");
System.out.println ("Percentage - " 7.5");
System.out.println ("Blood Group - " 'A');
System.out.println (true);
}
}
```

***NOTE :-**
1) For string → " "
2) For Number → No any, direct put number
3) Character → ' '
4) Boolean → simply take true or false.

* Data Types :-

- Data type helps the programmer to create containers (variables) to store data of specific type.

- In Java, data types are classified into
  1) Primitive data type.
  2) Non-Primitive data type.

* * I) Primitive data type :-

- The data types which is used to create containers (variables) to store primitive value i.e. single value.

- e.g. :- Integer numbers, decimal numbers, characters and booleans, etc.

* * II) Non-Primitive data type :-

- The data type which are used to create containers (variables) to store non-primitive data i.e. multivalue.

- e.g. :- String, class, arrays, etc.

Primitive Data	Data Type	size	Default values
Integer	byte	1 byte	0
	short	2 byte	0
	int	4 byte	0
	long	8 byte	0l/0L
Decimal	Float	4 byte	0.0f / 0.0F
	double	8 byte	0.0 / 0.0d
Character	char	2 byte	\U0000 ∵ (\U0000)
Boolean	boolean	1 bit	False

- 1 bit → 0 or 1
- 8 bit → 1 byte
- 1024 byte → 1 Kb
- 1024 Kb → 1 Mb
- 1024 Mb → 1 GB
- 1024 GB → 1 TB

* Ascending values for data type according to memory size :-

byte < short < int < long < float < double < char

* Saturday
29/01/2022 *

* Variables :-

- Variables are the containers created by the programmer to store value.
- We can create a variable in java, with the help of data types.

*

* Syntax to create variable :- (Declaration)

data type variable name ;

* NOTE:- We cannot use variable without declaring (creating).

* Also, data type Var1, Var2, Var3 ..... ; ie int a,b,c;

* Example to create a variable :-

- Float a ; • byte b ; • boolean c ;
- String m ; • int k ; , etc.

* Syntax to variable Initilization :-

variable name = data / literal / expresion / var ;

* Example for variable initilization :-

- a = 9.81f ; • b=20 ; • c = false ;
- m = "India" ; , • k = 100 , etc.

* Here, 1) m = false ; ✗ } Not acceptable.
2) b = 9.81f ; ✗

i.e. we cannot put data, without its proper data type.

```
*  class Var1                          ← variable program
   {
     public static void main (String [] args)
     {
        Float a ;          // variable declaration
        a = 0.01 f ;       // variable initilization
        System.out.println (a) ;

        byte b ;           // variable declaration
        b = 10 ;           // variable initilization.
        System.out.println (b) ;
     }
```

Siddhant Java

Siddhant Java

* Syntax for Variable Initilization and the declaration :-

```
data type  variable = value/exp/var ;
```

* e.g.:- 1) String s = "sita"
System.out.println (s);

2) int i = 50 ;

3) Float a = 0.015f ;

4) boolean b = true ;

5) char c = 'c'

* * Program :-

```
class Var2
{
  public static void main (String [] args)
  {
    String s = "siddhant R. shinde" ;
    System.out.println (s);
    long k = 8554861952l;
    System.out.println (k);
    String m = "10/08/1997" ;
    System.out.println (m);
    char c = 'A' ;
    System.out.println (c);
    boolean b = true ;
    System.out.println (b);
    int x = 2019 ;
    System.out.println (x) ;
    double y = 500000.0;
    System.out.println (y);
  }
}
```

* Corrected Program :-

```
class Var3
{
  public static void main (String [] args)
  {
    String name = "S.R. shinde" ;
    float salary = 10000.00f ;
    long phno = 8554861952l ;
    int yop = 2019 ;
    char blood = 'A' ;
    String dob = "10/08/1997" ;
    boolean student = true ;

    System.out.println (name) ;
    System.out.println (salary) ;
    ———— / ———— (phno) ;
    ———— / ———— (yop) ;
    ———— / ———— (blood) ;
    ———— / ———— (dob) ;
    ———— / ———— (student) ;
  }
}
```

Sundaram

Siddhant Java

* Types of Variables :- (According Visibility) :-

- Variable
  - Global variable
    - Static
    - Non-static
  - Local variable

* I) Global variables :-

- Variables declared in the class block, are called as "global variables".

* II) Local Variables :-

- A variable declared in the method block, or any other block except class block, are called as "Local variables".

* Characteristics of Local Variables :-

* 1) A variable declared inside the block stays local to that block.
  i.e. A local variable is always restricted to be used inside the same declared block.

* e.g. :-

```
Class Local1
{
    public static void main(String [] args)
    {
        {
            int a = 10 ;
            System.out.println (a) ;        // 10
        }
        System.out.println (a) ;            // Compile time error.
    }
}
```

* 2) A local variable cannot be used without initilization.
  - i.e. We should put some value into the variable to use it.

* e.g. :-

```
Class Var4
{
    public static void main (String [] args)
    {
        int a ;
        System.out.println (a) ;            // CTE
    }
}
```

Sundaram
Sundaram

* **NOTE :-** A local variable will not be assigned with default value, hense, we cant use a local variable with any initialization.

- If we try to use, we get compile time error i.e. CTE.

* 3) We cannot have more than one variable of same name inside the same block, else we get compile time error.

* e.g.:-

```
class Var4a
{
  public static void main(string[]args)
  {
      int a = 20 ;
      boolean a = true ;   // CTE
      System.out.println (a) ;
  }
}
```

Tuesday
1 Feb, 2022

* * Operators :-

- Operators are the symbols, which are used to do certain operations on values or variables.

- The operators are classified according to the number of operands required as :-

1) Unary Operator :- 1 operand / values.
2) Binary operator :- 2 operands / values.
3) Ternary operator :- 3 operands / values.

- Operators are also classified according to their operations i.e. on their task what they perform as :-

1) Arithmetic operator.
2) Relational operator.
3) Assignment operator.
4) Bitwise operator.
5) Increment & Decrement operator.
6) Compound assignment Operator.
7) Logical operator.
8) Miscellaneous operator.
9) Conditional operator.

* 1) Arithmetic Operators :- (+, -, ×, ÷, %)

- All arithmetic operators are binary operators.

Siddhant Jaya

- Arithmetic operators are used to perform basic arithmetic operation on the operands.

* a) Addition (+) :-

- This operator is used to find the ~~som~~ sum of two operands or values.

Operand 1	Operator	Operand 2	Result	Type
int	+	int	✓	int
long	+	long	✓	long
char	+	char	✓	int
float	+	float	✓	float
double	+	double	✓	double
boolean	+	boolean	NA	CTE
String	+	String	✓	String

(Concatenation)

* Concatenation :-

- Concatenation is the operation of joining two strings together.
- For concatenation in java, addtion (+) of two strings is done.

* b) Substraction :- (-)

- This operator is used to find difference between two operands or values.

Op. 1	Operator	Op. 2	Result	Type
int	-	int	✓	int
long	-	long	✓	long
char	-	char	✓	int
float	-	float	✓	float
double	-	double	✓	double
boolean	-	boolean	NA	CTE
String	-	String	NA	CTE

* c) Product (Multiplication) (*) :-

- This operator is used to find product of two operands or two values.

Op. 1	Operator	Op. 2	Result	Type
int	×	int	✓	int
long	×	long	✓	long
char	×	char	✓	int
float	×	float	✓	float
double	×	double	✓	double
boolean	×	boolean	NA	CTE
String	×	String	NA	CTE

* d) Division (/) :-

- This operator is used to dividation of two operands or two values. It gives quotient as a result.

Siddhant Java

Op. 1	Operator	Op. 2	Result	Type
int	/	int	✓	int
long	/	long	✓	long
float	/	float	✓	float
double	/	double	✓	double
char	/	char	✓	int
boolean	/	boolean	NA	CTE
string	/	string	NA	CTE

* e) Modulus :- (%)

- This operator is used to dividation of two operands or values, which gives remainder as its result.

Op. 1	Operator	Op. 2	Result	Type
int	%	int	✓	int
long	%	long	✓	long
float	%	float	✓	float
double	%	double	✓	double
char	%	char	✓	int
boolean	%	boolean	NA	CTE
string	%	string	NA	CTE

*e.g.:-

```
      5  ← quotient →  2
3 | 10                3 | 7
  -10                   -6
   00  ← Remainder →    1
```

(division Result)

(Modulus Result)

Assignment

* Programs :- b) (For substraction) :-

b1)
```
Class Op11
{
  public static void main (string [] args)
  {
      int a = 30 ;
      int b = 10 ;
      int result = a-b ;
      System.out.println (result) ;
  }
}
```

b2)
```
Class Op12
{
  public static void main (string [] args)
  {
      long a = 30l ;
      long b = 10l ;
      long result = a-b;
      System.out.print ln (result) ;
  }
}
```

b3)
```
Class Op13
{ public static void main (string) [] args
  {   char a = 'y' ;
      char b = 'x' ;
      int result = a-b ;
      System.out. println (result) ;
  }
}
```

Siddhant Java

b4)
```
Class Op14
{
    public static void main (string [] args)
    {
        float a = 12.00 f ;
        float b = 2.00 f ;
        float result = a-b ;
        System.out.println (result) ;
    }
}
```

b5)
```
Class Op15
{
    public static void main (string [] args)
    {
        double a = 42.00 ;
        double b = 12.00 ;
        double result = a-b ;
        system.out.println (result) ;
    }
}
```

b6)
```
Class Op16
{
    public static void main (string [] args)
    {
        boolean a = true ;
        boolean b = False ;
        System.out.println (a-b) ;   // CTE
    }
}
```

b7)
```
Class Op17
{
    public static void main (string [] args)
    {
        String a = "I love" ;
        String b = "India" ;
        System.out.println (a-b) ;   // CTE
    }
}
```

c) For Product (Multiplication) :- (*)

c1)
```
Class Op21
{
    public static void main (string [] args)
    {
        int a = 10 ;
        int b = 5 ;
        int result = a*b ;
        System.out.println(result) ;
    }
}
```

c2)
```
Class Op22
{ public static void main (string [] args)
    {
        long a = 10l ;
        long b = 5l ;
        long result = a*b ;
        System.out.print ln (result) ;
    }
}
```

Siddhant Java

C3)
```
Class Op23
{
    public static void main (string [] args)
    {
        char a = 'y' ;
        char b = 'x' ;
        int result = a*b ;
        System.out.println (result);
    }
}
```

C4)
```
Class Op 24
{
    public static void main (string [] args)
    {
        float a = 12.00f ;
        float b = 6.00f ;
        float result = a*b ;
        System.out.println (result) ;
    }
}
```

C5)
```
Class Op25
{
    public static void main (string [] args)
    {
        double a = 24.00 ;
        double b = 12:00 ;
        double result = a*b;
        System.out.println (result) ;
    }
}
```

C6)
```
Class Op26
{
    public static void main (string [] args)
    {
        boolean a = true ;
        boolean b = false ;
        System.out.println (a*b); // CTE
    }
}
```

C7)
```
Class Op27
{
    public static void main (string [] args)
    {
        String a = "I love" ;
        String b = "India" ;
        System.out.println (a*b) ;  // CTE
    }
}
```

d) For division (/) :-

d1)
```
Class Op31
{
    public static void main (string [] args)
    {
        int a = 30;
        int b = 10 ;
        int result = a/b ;
        System.out.println (result);
    }
}
```

Siddhant Java

Siddhant Java

d2)

```
Class Op32
{
  public static void main (string [] args)
  {
    long a = 30l
    long b = 10l
    long result = a/b ;
    System.out.println (result);
  }
}
```

d3)

```
Class Op33
{
  public static void main (string [] args)
  {
    char a = 'y' ;
    char b = 'x' ;
    int result = a/b ;
    System.out.println (result) ;
  }
}
```

d4)

```
Class Op34
{
  public static void main (string [] args)
  {
    Float a = 18.00f ;
    Float b = 6.00f ;
    Float result = a/b ;
    System.out.println (result);
  }
}
```

d5)

```
Class Op35
{
  public static void main (string [] args)
  {
    double a = 12.00 ;
    double b = 4.00;
    double result = a/b ;
    System.out.println (result) ;
  }
}
```

d6)

```
Class Op36
{
  public static void main (string [] args)
  {
    boolean a = true ;
    boolean b = false ;
    System.out.println (a/b) ;      // CTE
  }
}
```

d7)

```
Class Op37
{
  public static void main (string [] args)
  {
    String a = "I love" ;
    String b = "India" ;
    System.out.println (a/b) ;      // CTE
  }
}
```

e) For Modulus (%) :-

e1)
```
Class Op41
{
  public static void main (String [] args)
  {
    int a = 7 ;
    int b = 3 ;
    int result = a%b ;
    System.out.println (result);
  }
}
```

e2)
```
Class Op42
{
  Public static void main (String [] args)
  {
    long a = 25l ;
    long b = 12l ;
    long result = a%b ;
    System.out.println (result);
  }
}
```

e3)
```
Class Op43
{
  Public static void main (String [] args)
  {  char a = 'y' ;
     Char b = 'x' ;
     int result = a%b ;
     System.out.println (result);
  }
}
```

e4)
```
class Op44
{
  public static void main (String [] args)
  {
    Float a = 28.00f ;
    float b = 21.00f ;
    float result = a%b ;
    System.out.println (result) ;
  }
}
```

e5)
```
class Op45
{
  public static void main (String [] args)
  {
    double a = 15.25 ;
    double b = 8.25 ;
    double result = a%b ;
    System.out.println (result) ;
  }
}
```

e6)
```
class Op46
{
  public static void main (String [] args)
  {
    boolean a = true ;
    boolean b = False ;
    System.out.println (a%b) ;    // CTE
  }
}
```

Siddhant Java

e7)

```
Class Op47
{
  public static void main (String [] args)
  {
    String a = "I love";
    String b = "India";
    System.out.println(a%b); // CTE
  }
}
```

* Combined program for operators :-

(Here, int is considered)

```
Class Op50
{
  public static void main (String [] args)
  {
    int a = 30;
    int b = 10;
    int result1 = a+b;
    int result2 = a-b;
    int result3 = a*b;
    int result4 = a/b;
    int result5 = a%b;
    System.out.println(result1);
    System.out.println(result2);
    System.out.println(result3);
    System.out.println(result4);
    System.out.println(result5);
  }
}
```

Wednesday
09/03/2022

** 2) Assignment Operator :-

- Assignment operator is a binary operator.
- Assignment operator is used to store value into a variable.

* Syntax :- Variable = value / exp. / var.;

- It works from Right to Left (←).

* Rules :-

1) An assignment operator should have a variable in the LHS, else we get compile time error.

- e.g. :- 1) 10 = 10; → ✗
  2) int a = 10; → ✓

2) The type of value to be stored should be same as the type of variable provided.

- e.g. :- 1) char c = 'a' * 'b'; → ✗ (∵ a×b = int)
  2) double d = 3.142; → ✓

- If the type of the value to the type of variable is not same, there are two possibilities. :-

* Type Casting :-

- The process of converting one type of data to another type, is called "Type casting".

Siddhant Java

Siddhant Java

* Type casting *

- Primitive type casting
  - Widening
  - Narrowing
- Nonprimitive type (Derived type casting)
  - Upcasting
  - down casting

* A) Primitive type casting :-

- The process of converting one type of the primitive data to another primitive type, is called "primitive type casting".

- Primitive type casting classified as :-
  1) widening
  2) Narrowing

* B) Non-Primitive type casting :-

- The process of converting one type of non-primitive data to another non-primitive type, is called as "Non-primitive type casting".

- It classified as :-
  1) Upcasting.
  2) Downcasting.

* Programs for Widening :-

* 1) byte to other data type conversion :-

```
class Wide1
{
    public static void main (string [] args)
    {
        byte b = 20 ;
        short s = b ;
        int i = b ;
        char c = b ;          // CTE
        long l = b ;
        Float f = b ;
        double d = b ;
        System.out.println (b) ;      // 20
        System.out.println (s) ;      // 20
        System.out.println (i) ;      // 20
        System.out.println (c) ;      // CTE
        System.out.println (l) ;      // 20
        System.out.println (f) ;      // 20.0
        System.out.println (d) ;      // 20.0
    }
}
```

Every byte value may not have character assigned to it.

* Range of data type :-

1) byte → -128 to 128
2) short → -32768 to 32767
3) Other all remaining data types has too much high range, in the terms of 10^{x}. i.e. It ~~is~~ contains high range values.

Siddhant Java

* 2) Short to other data type conversion :-

```
class Wide2
{
  public static void main (String [] args)
  {
    short s = 20 ;
    int i = s ;
    char c = s ;          // CTE
    long l = s ;
    float f = s ;
    double d = s ;
    System.out.println (s) ;   // 20
    System.out.println (i) ;   // 20
    System.out.println (c) ;   // CTE
    System.out.println (l) ;   // 20
    System.out.println (f) ;   // 20.0
    System.out.println (d) ;   // 20.0
  }
}
```

Every short value it may not have character assigned to it.

* 3) int to other data type conversion :-

```
Class Wide3
{
  public static void main (String [] args)
  {
    int i = 20 ;
    char c = i ;          // CTE
    long l = i ;
    float f = i ;
    double d = i ;
    System.out.println (i) ;   // 20
    System.out.println (c) ;   // CTE
    System.out.println (l) ;   // 20
    System.out.println (f) ;   // 20.0
    System.out.println (d) ;   // 20.0
  }
}
```

* 4) char to other data type conversion :-

```
Class Wide4
{
  public static void main (String [] args)
  {
    char c = 'a' ;
    long l = c ;
    float f = c ;
    double d = c ;
    System.out.println (c) ;   // a
    System.out.println (l) ;   // 97
    System.out.println (f) ;   // 97.0
    System.out.println (d) ;   // 97.0
  }
}
```

* 5) long to other data type conversion :-

```
Class Wide5
{
  Public static void main (String [] args)
  {
```

```
        long l = 20l ;
        float f = l ;
        double d = l ;
        System.out.println (l) ;        // 20
        System.out.println (f) ;        // 20.0
        System.out.println (d) ;        // 20.0
    }
}
```

* 6) Float to other data type conversion :-

```
Class Wide6
{
    public static void main (String [] args)
    {
        float f = 20.5f ;
        double d = f ;
        System.out.println (f) ;        // 20.5
        System.out.println (d) ;        // 20.5
    }
}
```

A) 1) * Widening :-

- The process of converting or storing smaller type data into larger type container, is known as "widening".
- Widening is done implicitly (internally) by the compiler.

Thursday
03/02/2022

- Container size of byte data type is -128 to 127
- Container size of short data type is -32768 to 32767
- During widening, the programmer need not to force the compiler to convert smaller type to larger type.

A] 2) Narrowing :-

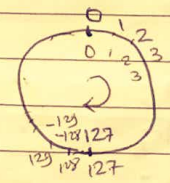
- It is the process of converting or storing larger type values into smaller type container (variables), is known as, "narrowing".
- Narrowing is not done implicitly (internally/directly) by compiler.
- The programmer has to force the compiler to do narrowing, with the help of "type cast operator". (It is an miscellaneous operator) (also unary op.)

* Syntax of Type cast operator :-

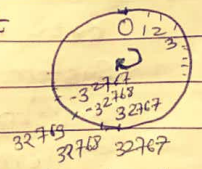
```
datatype variable = (Type to be converted) value ;
```

- Type cast operator is a Unary operator.
- If we try to store an int value (value more than container range), this cycles are followed :-

1) Byte



2) Short



Siddhant Java

i.e. we can consider following program as:-

```
class Narrow
{
  public static void main (string [] args)
  {
     short b = (short) 32769 ;
     System.out.println (b) ;   // -32767

     byte s = (byte) 130
     System.out.println (s) ;   // -126
  }
}
```

** Programs of Narrowing :-

* 1) double to other data type conversion :-

```
class Nar1
{
  public static void main (string [] args)
  {
     double d = 20.5 ;
     Float f = (Float) d ;
     long l = (long) d ;
     int i = (int) d ;
     char c = (char) d ;
     short s = (short) d ;
     byte b = (byte) d ;
     System.out.println (f) ;   // 20.5
     System.out.println (l) ;   // 20
     System.out.println (i) ;   // 20
     System.out.println (c) ;   // ♩
     System.out.println (s) ;   // 20
     System.out.println (b) ;   // 20
  }
}
```

* 2) Float to other data type conversion :-

```
class Nar2
{
  public static void main (string [] args)
  {
     Float f = 10.15f ;
     long l = (long) f ;
     int i = (int) f ;
     char c = (char) f ;
     short s = (short) f ;
     byte b = (byte) f ;
     System.out.println (l) ;   // 10
     System.out.println (i) ;   // 10
     System.out.println (c) ;   //  (empty space)
     System.out.println (s) ;   // 10
     System.out.println (b) ;   // 10
  }
}
```

* 3) long to other data type conversion :-

```
class Nar3
{
  public static void main (string [] args)
  {
```

Siddhant Java

```
        long l = 120l;
        int i = (int) l;
        char c = (char) l;
        short s = (short) l;
        byte b = (byte) l;
        System.out.println(i);      // 120
        System.out.println(c);      // x  symbol
        System.out.println(s);      // 120
        System.out.println(b);      // 120
    }
}
```

* 4) int to other data type conversion :-

```
class Nar4
{
    public static void main (String [] args)
    {
        int i = 153;
        char c = (char) i;
        short s = (short) i;
        byte b = (byte) i;
        System.out.println(c);      // ?  No any
        System.out.println(s);      // 153
        System.out.println(b);      // -103
    }
}
```

* 5) Char to other data type conversion :-

```
class Nar5
{
    public static void main (String [] args)
    {
        char c = 'a';
        short s = (short) c;
        byte b = (byte) c;
        System.out.println(s);      // 97
        System.out.println(b);      // 97
    }
}
```

* 6) Short to other data type conversion :-

```
class Nar6
{
    public static void main (String [] args)
    {
        short s = 363;
        byte b = (byte) s;
        System.out.println(b);      // 107
    }
}
```

* For Assignment Operator :-

- If the variable type and type of value are not same there will be two possibilities :-

1) If the value is of smaller data type & variable is of larger type, the compiler tries to do widening.

Siddhant Java

Friday
04/02/22

2) If widening is not possible, we get a compile time error (compiler implicitly cannot do narrowing.).

* Questions for interview purpose :-

* 1) (What is process of getting output as a=b=c=d=10 ?)

```
class A
{
  public static void main(String [] args)
  {
    int a,b,c,d;
    a=b=c=d=10;
    System.out.println(a);   // 10
    System.out.println(b);   // 10
    System.out.println(c);   // 10
    System.out.println(d);   // 10
  }
}
```

=> a [10] b [10] c [10] d [10] = 10 (1, 2, 3, 4)

- Here first d=10, then from right to left process is done.

* 2)

```
Class B
{ public static void main (String [] args)
  { int a;
    System.out.println (a=12+20);   // 32
  }
}
```

=> Here, first expression is solved & then output is printed.

* * 3] Compound Assignment Operator :- (+=, -=, *=, /=, %=)

- Compound assignment operator is a binary operator.
- It is used to assign a value into a variable by updating the older value from it (in it).
- They are as :- 1) + = 2) - = 3) * = 4) / = 5) % =
- Compound assignment operator performs the operation specified by the additional operator and then assign the result to left operand.

* Syntax for compound addition operator :-

variable += value/exp. ;

- It is very similar to Variable = var + Exp. ;

* e.g. :-

```
Class C
{ public static void main (String [] args)
  { int c = 100;
    c = c+10;
    System.out.println (c);   // 110
    int a = 100;
    a+=10;
    System.out.println(a);   // 110
  }
}
```

Siddhant Java

Siddhant Java

* 4] Relational Operator :-

- Relational operators are binary operators.
- It is used to compaire two values and it helps the programmer in decision making.

==	→	Equality operator
<	→	Less than
>	→	Greater than
<=	→	Less than equal
>=	→	Greater than equal
!=	→	Not equal

* 1) If the First operand is equal to 2nd operant, the equality operator writtens "true".

* 2) < → If LHS less than RHS, the equality op. writtens "true".

* 3) > → If LHS is greater than RHS, equality operator writtens "true".

* 4) <= → If LHS is less than or equal to RHS, then it writtens "true".

* 5) >= → If LHS is greater than or equal to RHS, then it shows "true".

* 6) != → If LHS is not equals to RHS, then it writtens "true".

* NOTE :- The result type of a relational operator is boolean type.

* e.g. :-

```
class Rel1
{
  public static void main (String[] args)
  {
    System.out.println(10==10);      // true
    System.out.println(12==10);      // False
    System.out.println(2<5);         // true
    System.out.println(3.5>3.52);    // false
    System.out.println(10<=11);      // true
    System.out.println(10<=10);      // true
    System.out.println(3.00>3);      // False
    System.out.println(10.00==10);   // true
    System.out.println('a'>=97);     // true
    System.out.println(5!=5);        // False
    System.out.println(1!=2);        // true
    System.out.println("Hi"==10);    // CTE
    System.out.println(true==false); // False
    System.out.println(true>false);  // CTE
    System.out.println("India">="india"); // CTE
    System.out.println("India"=="India"); // true
  }
}
```

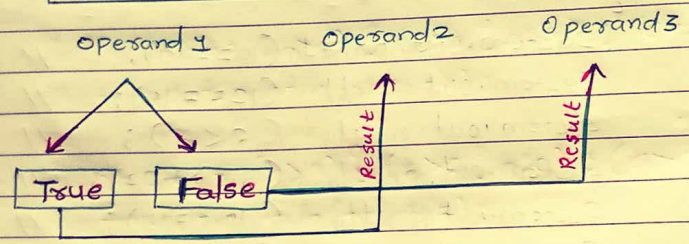
Saturday
05/02/22

* 5] Conditional Operator :-

- Conditional operator is a ternary operator.
- It is used for decision making purpose.

* Syntax :-

```
Condition ? Lit/exp/var : Lit/exp/var ;
```



1) The condition (operand 1) must be of boolean type.
2) If operand 1 writterns "true", then control goes to operand 2.
3) If OP.1 writtens "false", then control goes to operand 3.
4) The result of a conditional operator depends on the type of Operand 2 and operand 3.

* Class Cond1 [1) For even or odd]

```
{ public static void main (String [] args)
  {
      int num = 122 ;
      String ans = (num % 2) == 0 ? "It is even" : "It is odd"
      System.out.println (ans) ;
  }
}
```

// It is even.

* 2) For character :-

```
class Cond2
{
  public static void main (String [] args)
  {
      char c1 = 'a' ;
      char c2 = 'b' ;
      char result = (c1 > c2) ? c1 : c2 ;
      System.out.println (result) ;   // b
  }
}
```

* 3) To get largest number from three numbers :-

```
class Cond3
{
  public static void main (String [] args)
  {
      int a = 10 ;
      int b = 20 ;
      int c = 30 ;

  ①   int large = a > b ? a : b ;
      int largest = large > c ? large : c ;
      System.out.print ln (largest) ;

  ②   int biggest = (a > b) ? (a > c ? a : c) : (b > c ? b : c);
      System.out.print ln (biggest) ;

  ③   int result = (a > b ? a : b) > c ? (a > c ? a : c) : (b > c ? b : c);
      System.out.print ln (result) ;
```

Siddhant Java

```
(4)  int result1 = c>(a>b? a:b) ? c : (a>b ? a:b);
     System.out.println (result1);

(5)  int result2 = (a>c) ? (a>b ? a:b) : (c>b ? c:b);
     System.out.println(result2);

(6)  int result3 = a>(b>c ? b:c) ? a : (b>c ? b:c);
     System.out.println (result3);

(7)  int result4 = (b>c) ? (b>a ? b:a) : (c>a ? c:a);
     System.out.println (result4);

(8)  int result5 = (b>a) ? (b>c ? b:c) : (a>c ? a:c);
     System.out.println (result5);

(9)  int result6 = b>(c>a ? c:a) ? b : (c>a ? c:a);
     System.out.println (result6);

(10) int result7 = (c>a) ? (c>b ? c:b) : (a>b ? a:b);
     System.out.println (result7);

(11) int result8 = (c>b) ? (c>a ? c:a) : (b>a ? b:a);
     System.out.println (result8);
     }
}
```

Siddhant Java

* 6 Logical Operator :-

- Logical operators are used to make logical decision like true or false.
- This operators help us in decision making.

- Logical operator are :-

1> AND operator. (&&)
2> OR operator. (||)
3> NOT operator. (!)

* 1> AND Operator :- (&&)

- It is an binary operator, which is used to make logical decisions.

* Truth Table :-

I/P1 (OP1)	I/P2 (OP2)	O/P (Result)
true	False	False
False	true	False
False	False	False
true	true	true

- AND operator writtens true only if, both operands are true.
- If any one of the operand is False, the result of AND operator is False.

Siddhant Java

* 2) OR operator :- (||)

- It is a binary operator, which helps us in the decision making.

* Truth Table :-

OP_1	OP_2	Result
true	False	true
false	true	true
False	False	False
true	~~false~~	true.

- The result of OR operator is true, if anyone of operand writtens true.
- OR operator writtens False, if both operands are false.

* 3) NOT operator :- (!)

- NOT operator is a unary operator, which is used for decision making.

* Truth Table :-

OP_1	Result
true	false
False	true

- The result of NOT operator will be true, if operand writtens False.
- The result of NOT operator will be false, if operand is true.

* NOTE :-

- The operands for a logical operator must written boolean values.
- The result type of logical operator is a boolean.

```
class Logic1
{
    public static void main (String[]args)
    {
        AND Truth table
        System.out.println(true && false);     // false
        System.out.println (false && true);    // False
        System.out.println (False && false);   // false
        System.out.println (true && true);     // true
        OR Truth table
        System.out.println (true || false);    // true
        System.out.println (false || true);    // true
        System.out.println (False || false);   // false
        System.out.println (true || true);     // true
        NOT truth table
        System.out.println (!true);            // false
        System.out.println (!false);           // true
    }
}
```

* Examples :-

* a) **AND :-**
```
(10>1 && 1<0.02) ;      // False
(5==5 && 5<5) ;         // False
('x'>'X' && 'a'>'A') ;  // true
(13.52<=13 && 0==0) ;   // false
```

* b) **OR :-**
```
(10>1 || 1>0.04)        // true
(true || 'a'>=97) ;     // true
(5==5 || 5<5) ;         // true
(1>2 || 0<0) ;          // false
```

* c) **NOT :-**
```
(!(9>=9.00002)) ;       // true
(!(true != true)) ;     // true
```

* Program for combination of Logical Operator :-

```
class Logic2
{
  public static void main (String [] args)
  {
   S.o.pln((5>10 && 1.5<2.5) || (4>2));      // true
   S.o.pln(('a'>'b' || 'c'>'d') && ('z'>'Y'));  // false
   S.o.pln((5!=2 && 2==3) && (4!=3));        // false
   S.o.pln((1>5 || 5<6) || (50>50.2));       // true
   S.o.pln((!((10==10) && (40>50)));         // true
   S.o.pln((!(10<25 && 2.5>2.49)) ;          // true
   S.o.pln((!('a'>'c' || 'A'>'c')) ;         // true
   S.o.pln((-2.5>4.5 && -2.5>-2.8) || (4<2));  // False
   S.o.pln((!(10==15 || -2.5>-5.9)) ;        // False
```


* 7] Increment & Decrement Operator :-

- This are unary operators.

I) Increment Operator (++) :-

- Increment operator is classified into :-
  1) Pre-increment operator.
  2) Post-increment operator.

a) Pre-increment operator :-

- If a variable is preffix (before) with increment operator, then it is known as "pre-increment operator.

- Syntax: `++ variable ;`

*** Operation :-**

1) First it updates (adds) one value to the old value by 1.
2) Then, it uses the updated value.

b) Post-increment operator :-

- If a variable is suffix (after) with increment operator, then it is called as "Post-increment operator.

* Syntax :- `Variable ++ ;`

Siddhant Java

- In post increment operator, first the given value is used (substituted).
- After using (substituting) it will then emediatly update the Variable with an increase of one (I) value to the variable.

* II] Decrement Operator :-

- Decrement operator is classified into :-
  1) Pre-decrement operator
  2) Post-decrement operator.

* a) Pre-decrement Operator :-

- If a variable is preffix (before) with decrement operator, then it is "pre-decrement operator."

* Syntax :- `-- Variable ;`

- First it reduces the one (1) value to the old value by 1.
- Then it uses the updated (reduced) value.

* b) Post-decrement Operator :-

- If a variable is suffix (after) with decrement operator, then it is "post-decrement operator".

* Syntax :- `variable -- ;`

- In post decrement operator, first the given value is used (substituted).
- After using (substituting) it will then imediatly update the variable with an ~~in~~ decrease (Reducement) of 1 value to the given variable.

* Programs :-

1)
```
class Increment1
{
    public static void main (String [] args)
    {
        int a = 100
        System.out.println (a++) ;      // 100
        System.out.println (++a) ;      // 102
    }
}
```

2)
```
Class Increment2
{
    public static void main (String [] args)
    {
        int a = 101 ;
        int b = a++ + ++a + a-- ;
        System.out.println (a) ;        // 102
        System.out.println (b) ;        // 307
    }
}
```

Siddhant Java

Siddhant Java

3) class Increment3

```
{
  public static void main (String [] args)
  {
    int a = 100, b = 87, c = 1 ;
    a = b + c-- + a++ - --b + ++c ;
    c = a + b- --a + ++b- --a ;
    c = a + b + c++ ;

    System.out.println (a) ;      // 101
    System.out.println (b) ;      // 87
    System.out.println (c) ;      // 261
  }
}
```

4) class Increment4

```
{
  public static void main (String [] args)
  {
    int a = 26, b = 7, c = 12, d = 4 ;
    a++ ;
    d = ++a - --c ;
    --a ;
    b = d++ - b++ ;
    --b ;

    System.out.println (a) ;      // 27
    System.out.println (b) ;      // 10
    System.out.println (c) ;      // 11
    System.out.println (d) ;      // 19
  }
}
```

Sundaram

* Manually calculation :-

1) a = 100, b = 87, c = 1 ;

a = b + c-- + a++ - --b + ++c ;

c = a + b- --a + ++b- --a ;

c = a + b + c++ ;

Find :- a, b, c

→

a = ~~100~~ ~~101~~ ~~105~~ ~~102~~ 101

b = ~~87~~ ~~86~~ 87

c = ~~1~~ ~~0~~ ~~1~~ ~~75~~ ~~74~~ 261

Answers
a = 101,
b = 87, c = 261

ⓐ a = b + c-- + a++ - --b + ++c

a = 87 + 1 + 100 - 86 + 1

a = 103

ⓑ c = a + b- --a + ++b- --a

c = 103 + 86 - 102 + 87 - 101

c = 73

ⓒ c = a + b + c++

c = 101 + 87 + 73

c = 261

2) a = ~~26~~ ~~27~~ ~~28~~ 27, b = ~~7~~ ~~8~~ ~~10~~ 10, c = ~~12~~ 11, d = ~~4~~ ~~17~~ ~~18~~ 19

ⓐ a++ = 26

ⓑ d = ++a- --c

d = 28 - 11 = 17

ⓒ --a = 26

ⓓ b = d++ - b++

b = ~~18~~ - ~~7~~ = ~~10~~

ⓔ --b = 10

Answers
- a = 27
- b = 10
- c = 11
- d = 19

Sundaram

* Program for Increment & decrement operator in all data types:-

```
class Increment5
{
public void static main (String[] args)
{
    byte b = 25;
    System.out.println(b++); // 25  (b-- = 25)
    System.out.print ln(++b); // 27  (--b = 23)

    short s = 45;
    System.out.println(s++); // 45  (s-- = 45)
    System.out.println(++s); // 47  (--s = 43)

    int i = 50;
    System.out.println(i++); // 50  (i-- = 50)
    System.out.println(++i); // 52  (--i = 48)

    char c = 'a';
    System.out.println(c++); // a  (c-- = a)
    System.out.println(++c); // c  (--c = _)

    long l = 10l;
    System.out.println(l++) // 10  (l-- = 10)
    System.out.print ln(++l) // 12  (--l = 8)

    Float f = 10.5f;
    System.out.println(f++); // 10.5  (f-- = 10.5)
    System.out.println(++f); // 12.5  (--f = 8.5)
```



```
    double d = 20.5;
    System.out.println(d++); // 20.5  (d-- = 20.5)
    System.out.print ln(++d); // 22.5  (--d = 18.5)

    boolean z = true;
    System.out.println(z++); // CTE  (z-- = CTE)
    System.out.print ln(++z); // CTE  (--z = CTE)

    String y = "India";
    System.out.print ln(y++); // CTE  (y-- = CTE)
    System.out.print ln(++y); // CTE  (--y = CTE)

}
}
```

* Dynamically Reading Data from User:-

- To read data dynamically from the user, we use built in class called as "scanner".
- Scanner class is used to read data dynamically during the excution of the program.
- Scanner is a built in class presenting the java.util package.

* Steps to use scanner class:-

* 1) Import scanner class:-

- We can ~~input~~ ~~Scan~~ import scanner class with of help of keyword "import".

Siddhant Java

Siddhant Java

* **Syntax :-** `import java.util.Scanner;`

- import statement must be the first statement of java program.

* 2) Create the object of scanner class, with the help of a scanner constructor.

* **Syntax to create object :-**

- `Scanner a = new Scanner (System.in);`

constructor

- The scanner object gives scanner object refference which should be stored in scanner type variable.

* 3) With the help of scanner object refference use the methods of scanner class to read data dynamically to the user.

* **Methods of Scanner Class :-**

Data Type	Method Name	Return type
byte	nextByte()	byte
short	nextShort()	short
int	next Int ()	int
long	next Long ()	long
Float	next Float()	Float
double	next Double()	double
char	next().CharAt(0)	char
boolean	next Boolean()	boolean
String	→ next()	String
	→ next Line()	

* **Program for Largest Number from three numbers :-**

```
import java.util.Scanner ;          // Step 1
class Scan1
{
  public static void main (String [] args)
  {
      Scanner a = new Scanner (System.in);  //Step 2

      System.out.println ("Enter the first number :");
      int num1 = a.next Int () ;            // Step 3

      System.out.println ("Enter the second number :");
      int num2 = a.next Int () ;

      System.out.println ("Enter the third number :");
      int num3 = a.next Int () ;

      int largest = num1 (num2 > num3 ? num2 : num3) ?
                    num1 : (num2 > num3 ? num2 : num3);
```

Thursday
10/02/2022

```
System.out.println ("The largest of" + num1 +
                    "," + num2 + "&" + num3 +
                    "is :" + largest);
    }
}
```

* Control Flow Statements :-

- The statements which are used to control the execution flow of program, are called "control flow statements".

- The control flow statements are :-
  1) Decision statements.
  2) Loop statements.

* 1) Decision Statements :-

- The decision statements helps us to decide whether to execute a set of instructions or not.

- The execution of the set of instructions is based on the given favourable condition.

- Decision are classified as

  1) Simple if statements
  2) if else statements
  3) if else if ladder statement
  4) Switch statement.

* a) Simple if statements :-

- if statement helps the programmer in the decision making.

* syntax :- `if (condition)`

- The statements inside the if block get executed, if that condition is evaluated to true.

- If the condition is false, the execution of the statements inside the if block are skipped.

* NOTE :-

- The condition for a if statement must be of boolean type.

* Programs :-

```
class If1
{
    public static void main (string [] args)
    {
        System.out.println ("Main Begin");   // printed
        if (False)
        {
            System.out.println ("Hellow World");   // skipped due to false
        }
        System.out.println ("main end");   // printed
    }
```

Siddhant Java

Siddhant Java

* b) if else statements :-

- This statements are help the programmer in decision making.

*** Syntax :-**

```
if (condition)
{
  statement-1 ;
}
else
{
  statements-2 ;
}
```

- The statements inside the if block gets executed, if condition is true and same time, the execution of else block is skipped.

- else block gets executed only when the if condition is false.

*** NOTE :-**

1) The condition must be of boolean type.
2) We cannot execute both the blocks at a time (i.e. only one block can be executed at a time.)

Sundaram

*** Program :- 1) Percentage criteria using if else :-**

```
import java.util.Scanner ;
class If2
{
  Public static void main (String [] args)
  {
    Scanner s = new Scanner (System.in) ;

    System.out.Println ("Enter your percentage") ;
    double percent = s.nextDouble () ;

    if(percent >= 60)
    {
      System.out.println ("You are eligible") ;
    }

    else
    {
      System.out.Println ("You are not eligible") ;
    }
  }
}
```

*** 2) Finding even odd program using if else :-**

```
import java.util.Scanner ;
class If3
{
  Public static void main (String [] args)
  {
```

Sundaram

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

System.out.println ("Enter a number");
int num = s.nextInt ();

if (num % 2 = 0)
{
   System.out.println (num+"is even number");
}
else
{
   System.out.println(num+"is odd number");
}
}
}
```

* 3) Natural number using if else :-

```
import java.util.Scanner;
Class If4
{
 public static void main (string[] args)
 {
   Scanner sc = new Scanner (system.in);

   System.out.println ("Enter a number");

   int num = sc.nextInt ();
   if (num >0)
   {
     System.out.println (num+"is Natural number");
```

Friday
11/02/2022

```
else
{
   System.out.println (num+"is not a natural Num.");
}
}
}
```

* c) if else if Ladder statements :-

- It is used in decision making purpose.
- If a programmer has more than one condition to be checked, we make use of if else if ladder.

* Syntax :-

```
if (condition)
{
    statement 1 ;
}
else if (condition)
{
    statement 2 ;
}
else if (condition)
{
    statement 3 ;
}
:
else
{
    statement ;
}
```

Siddhant Java

* Characteristics of Ladder statements :-

1) Using an else block is not mandatory.

2) Else block gets executed when all the conditions are False.

3) In else if ladder, only one block can get executed, eventhough multiple if conditions are true.
- The first block which evaluates to true, gets executed.

* Programs :- 1) Simple program :-

```
class IfElseIf1
{
  public static void main (string[] args)
  {
    if (false)
    {
     System.out.println ("From if block");
    }
    else if (true)
    {
      System.out.println ("From else if block 1");
    }                                  // executed
    else if (false)
    {
      System.out.println ("From else if block2");
    }
    else
    {
      System.out.println ("From else block");
    }
  }
}
```

* 2) Percentage criteria using if else if ladder :-

```
import java.util.Scanner ;
class Pertage /IfElseIf2
{
  public static void main (string [] args)
  {
    Scanner s = new Scanner (System.in) ;
    System.out.println ("Enter your Percentage");
    double percent = s.nextDouble( ) ;

    if (percent > 100) || percent < 0 )
    {
      System.out.println ("Invalid input");
    }

    else if (percent == 35)
    {
      System.out.println ("Just pass");
    }

    else if ( percent > 35 && percent < 60)
    {
      System.out.println ("Second class");
    }
```

Siddhant Java

Siddhant Java

```
else if(percent>=60 && percent<75)
{
System.out.println("First class");
}

else if (percent>=75 && percent<=100)
{
System.out.println("Distinction");
}

else
{
System.out.println("Fail");
}
}
}
```

* 3) Battery indicator using Ladder Statements:-

```
import java.util.Scanner;
class IfElseIf3
{
public static void main(String[] args)
{
Scanner s= new Scanner(System.in);
System.out.println("Enter your battery %");
int battery = s.nextInt();

if (battery>100 || battery<0)
{
System.out.println("Invalid input");
}
else
if ( battery >20 && battery <=100)
{ System.out.println("Battery is in good state");
}
else
if ( battery <=20 && battery >10)
{
System.out.println("Battery is low");
}
else
if ( battery <=10 && battery >5)
{
System.out.println("Turn on Power Saving mode");
}
else
if (battery <=5 && battery >0);
{
System.out.println("Plug in charger immediatly");
}
}
}
```

* 4) Program on Vaccination using Ladder:-

```
import java.util.Scanner;
Class IfElseIF4
{
public static void main (String[] args)
{
Scanner s = new Scanner (System.in);
```

Assignment

```
System.out.println("Enter your age");
int age = s.nextInt();
if (age<15)
{
  System.out.println("Invalid entry");
}

System.out.println("Enter your number of
                    vaccination done");
int num= s.nextInt();
if (num<0 || num>2)
{
  System.out.println("Invalid entry");
}

if (age>15 && age<=18)
{
  if (num==0)
  {                                  covaxin
    System.out.println("Not vaccinated");
  }
  else if (num==1)
  {                                  covaxin
    System.out.println("Partially vaccinated");
  }
  else if (num==2)
  {                                  covaxin
    System.out.println("fully vaccinated");
  }
}
```

```
else if (age>18 && age<=45)
{
  if (num==0)
  {                                  covishield
    System.out.println("Not vaccinated");
  }
  else if (num==1)
  {                                  covishield
    System.out.println("Partially vaccinated");
  }
  else if (num==2)
  {                                  covishield
    System.out.println("Fully vaccinated");
  }
}

else if (age>45 && age<=60)
{
  if (num==0)
  {                                  covaxin
    System.out.println("Not vaccinated");
  }
  else if (num==1)
  {                                  covaxin
    System.out.println("Partially vaccin...");
  }
  else if (num==2)
  {                                  covaxin
    System.out.println("fully vaccinated");
  }
}
```

Siddhant Java

```
else if (age>60)
{
    if (num==0)
    {
    System.out.println("Not vaccinated") ;   Booster
    }
    else if (num==1)
    {
    System.out.println("Partially vaccinated");   Booster
    }
    else if (num==2)
    {
    System.out.println("Fully Vaccinated");   Booster
    }
  }
 }
}
```

* 5) Program on Discount using Ladder:-

```
import java.util.Scanner ;
class IfElseIF5
{
  public static void main (String [] args);
  {
    Scanner s= new Scanner (System.in);
    System.out.println ("Enter your total price
                        of shopping") ;
    double price = s.nextDouble();
    if (price < 1000)
    {
      System.out.println ("No discount available");
    }

    else if (price >=1000 && price <3000)
    {
      System.out.println("10% discount available");
    }

    else if (price>=3000 && price< 5000)
    {
      System.out.println("20% discount available");
    }

    else if (price>=5000 && price < 8000)
    {
      System.out.println(30% discount available);
    }

    else if (price >=8000)
    {
      System.out.println(35% discount available);
    }
  }
}
```

* 6) Program on (+)ve, (-)ve OR Neutral using Ladder:-

```
import java.util.Scanner ;
class IfElseIF6
```

Siddhant Java

Siddhant Java

```
{
    public static void main (String [] args)
    {
      Scanner s= new Scanner (System.in);
      System.out.println ("Enter the number");
      int num= s.nextInt ();

      if (num>0)
      {
        System.out.println (num+ "is positive number");
      }

      else if (num<0)
      {
        System.out.println (num+ "is negative no.");
      }

      else
      {
        System.out.println (num+ "is neutral no.");
      }
    }
}
```

7) Tax on income using Ladder :- (Program)

```
import java.util.Scanner;
class IfElseIf7
{
  public static void main (String [] args)
  {
    Scanner s= new Scanner (System.in);
    System.out.println ("Enter your anual income");
    double income = s.nextDouble ();

    if (income < 250000)
      {
        System.out.println ("Tax is 0%");
      }

    else if (income >=250000 && income < 500000)
      {
        System.out.println ("Tax is 10%");
      }

    else if (income >=500000 && income < 1000000)
      {
        System.out.println ("Tax is 20%");
      }

    else if (income > 1000000)
      {
        System.out.println ("Tax is 30%");
      }
    }
}
```

* 8) Product sorting program using Ladder :-

```
import java.util.Scanner
class IfElseIf8
{
  public static void main (String [] args)
  {
```

```
Scanner s= new Scanner (System.in);
System.out.println (" Enter your product serial
                    number");
int num= s.nextInt();

if (num>=1 && num < 25)
{
  System.out.println ("Your product is in Rack1");
}

else if (num >=25 && num <50)
{
  System.out.println ("Your Product is in Rack2");
}

else if (num >=50 && num < 100)
{
  System.out.println ("Your product is in Rack3");
}

else
{ System.out.println ("Your product is not
                    available");
}
}
}
```

Saturday
12/02/2022

* d) Switch statements :-

- Switch is a decision statement.

*** Syntax :-**

```
switch (Lit/exp/var)
{
  case Lit/exp :
  {
    task 1 ;
    break ;
  }

  case Lit/exp :
  {
    task 2 ;
    break ;
  }

  case Lit/exp :
  {
    task 3 ;
    break ;
  }
  ⋮
  default :
  {
    task 4 ;
    [break] ;   ← Not required OR NOT mandatory
  }
}
```

Siddhant Java

1) • We can pass following type data only inside switch.
   1) Byte
   2) short
   3) int
   4) char
   5) string

2) • We cannot pass following inside switch & case:-
   1) Long
   2) Float
   3) double
   4) boolean

3) • Inside switch block, we can have any number of case blocks, but only one default block.

4) • Using a default block is not mandatory.

5) • We cannot pass a variable inside case.

6) • We can pass expressions inside switch & case.

7) • We can pass variable inside switch.

* Programs :- 1) Simple Program:-

* 1)
```
class Switch1
{
  public static void main (String[] args);
  {
    switch (I)
    {
      case (2) :
      {
        System.out.println("From First block");
        break;
      }

      case (1) :
      {
        System.out.println("From second block");
        break;          // executes it
      }

      case (5) :
      {
        System.out.println("From third block");
        break;
      }

      default :
      {
        System.out.println("From default block");
      }
    }
  }
}
```

* 2) Program for Days & Subjects using switch:-

Siddhant Java

Sundaram

Siddhant Java

```
import java.util.Scanner;
class Switch2
{
  public static void main (String[] args)
  {
      Scanner s = new Scanner (System.in);
    System.out.println ("****Welcome****");
    System.out.println (1. Marathi) ;
    System.out.println (2. English) ;
    System.out.println (3. Science) ;
    System.out.println (4. Maths ) ;
    System.out.println (5. History) ;
    System.out.println (6. Art ) ;
    System.out.println (7. Go play) ;

    System.out.println ("Enter your option :");
    int option = s.nextInt () ;

    switch (option)
      {
        case 1:
        { System.out.println ("Read Marathi");
          break ;
        }
        case 2:
        { System.out.println ("Practice English");
          break ;
        }
        case 3:
        { System.out.println ("study science");
          break ;
        }
        case 4:
        { System.out.println ("Practice Maths");
          break ;
        }
        case 5:
        { System.out.println (" Read History");
          break ;
        }
        case 6:
        { System.out.println (" Do Art");
          break ;
        }
        case 7:
        { System.out.println (" Go play... enjoy");
          break ;
        }
        default :
        { System.out.println (" Invalid entry");
          break ; ← (Not neccessary)
        }
      }
  }
}
```

* 3) Program on Hotel order using switch statement:-

```
import java.util.Scanner ;
class Switch3
{
  public static void main (String [] args)
  {
    Scanner s = new Scanner (System.in);
```

Assignment

```
System.out.println ("*** Menu ****");
System. out. println (" 1. Veg");
System.out. print ln ("2. Non veg");

System.out.print ln ("Enter your option : ");
int opt = s.next Int ();

Switch (opt)
{ Case 1:
System.out.print ln ("1. Mix Veg Thali" ) ;
System.out. print ln ("2. Kolhapuri special");
System.out.print ln ("3. Malvani Special");

System.out.print ln ( "Select your option");
int ord = s.next Int ();

Switch (ord)
  { case 1:
    { Syst.out.print ln ("You have to pay Rs.150");
      break ;
    } ~~System.out.prin~~
    case2:
    { syst.out.pln("You have to pay 220 Rs.");
      break ;
    }
    case3:
    { syst. out. pln. (" You have to pay Rs.200");
      break ;
    }
  }
  break ;
```

```
case 2:
System.out. print ln ("1. Kolhapuri special");
System.out.print ln (" 2. Mutton thali") ;
System.out.print ln (" 3. Eggs thali" ) ;

System.out.print ln ("Select your order");
int order = s.next Int ();

switch (order)
  { case 1:
    { syst.out.pln("You have to pay Rs.250");
      break;
    }
    case 2:
    { syst.out.pln("You have to pay Rs.300);
      break;
    }
    case3
    { syst.out.pln ("You have to pay Rs.175);
      break;
    }
    break;
  }

default:
  {
   System.out.print ln ("Invalid Entry... Retry!!");
  }
}
}
}
```

Siddhant Java

* 4) Program on Loan & its interest using Switch :-

```
import java.util.Scanner;
class Switch4
{
 public static void main (String[] args);
 {
   Scanner s = new Scanner (System.in);

   System.out.println ("*** Welcome in Loan Section **");
   System.out.println ("1. Home Loan");
   System.out.println ("2. Vehicle Loan");
   System.out.println ("3. Business Loan");
   System.out.println ("Enter your option :");
   int opt = s.nextInt ();

   switch (opt)
   {
   Case 1:
      System.out.println ("1. BOI Bank");
      System.out.println ("2. SBI Bank");
      System.out.println ("3. HDFC Bank");

      System.out.println ("Select your option");
      int loan1 = s.nextInt ();

      switch (loan1)
      { Case 1:
         { Syst.out.pln ("Interest is 6.75% P.A.");
           break;
         }
```

```
         case 2:
         { Syst.out.pln ("Interest is 6.25% P.A.");
           break;
         }
         Case 3:
         { Syst.out.pln ("Interest is 7.05% P.A.");
           break;
         }
      }
      break;

      Case 2:
         System.out.println ("1. BOI bank");
         System.out.println ("2. SBI bank");
         System.out.println ("3. HDFC bank");

         System.out.println ("Select your option");
         int loan2 = s.nextInt ();

         switch (loan2)
         {
           case 1:
           { Syst.out.pln ("Interest is 10.05% P.A.");
             break;
           }
           case 2:
           { Syst.out.pln ("Interest is 9.75% P.A.");
             break;
           }
           case 3:
           { Syst.out.pln ("Interest is 10.25% P.A.");
             break;
           } } break;
```

```
Case3 :
    System.out.println("1. BOI bank");
    System.out.println("2. SBI bank");
    System.out.println("3. HDFC bank");

    System.out.println("Enter your option");
    int loan3= s.nextInt();

  switch (loan3)
  {
    case1 :
    { syst.out.pln("Interest is 11.50% P.A.");
      break;
    }
    case2 :
    { Syst.out.pln(" Interest is 10.75% P.A.");
      break;
    }
    case3:
    { Syst.out.pln("Interest is 12.25% P.A.");
      break;
    }
  }
  break;

  default :
  { System.out.println("Invalid entry .... Retry!!");
  }
}
}
```

Monday
14/02/2022

* 5) Program for Vowel identification :-

```
class Switch5
{
  public static void main (string [] args)
  {
      char ch = 'b';

    if (ch>='a' && ch<='z' || ch>='A' && ch<='Z')
    {
      Switch (ch)
      {
        case'a' :
        case 'e' :
        case 'i' :
        case 'o' :
        case 'u' :
        case 'A' :
        case 'E' :
        case 'I' :
        case 'O' :
        case 'U) :
        { System.out.println("It is a vowel");
          break;
        }

        default:
        { System.out.println("It is consonant");
        }
      }
    }
  }
```

Siddhant Java

```
    else
        {System.out.println("Not an alphabate");
        }
    }
}
```

* Control Flow in switch Statements :-

- The value given to the switch is compared to the value given to the case using equality operator.

2) If a case is evaluated to false, the control goes to the next case block.

3) If all the cases are evaluated to false, then default block gets executed.

4) If a case evaluated to true, then :-
   a) The statement inside the case block gets executed.
   b) If break statement is not used inside case block, the statements of below case blocks also gets executed.

* Break :-

- Break is a control transfer statement.
- It is used only inside switch and loops statements.

* Using break inside switch :-

- Break will transfer the control outside the switch block.
- We can use a break statement in all the case blocks and it is optional.

* 2] Loop Statements :-

- Loop statements are control flow statements, which helps the programmer to execute the set of instructions repeatedly.
- Different types of loop statements are :-
  1) while loop
  2) do-while loop
  3) for loop
  4) For each loop OR Advanced For loop. OR Enhansed For loop.

* ~~While Loop~~

- Every loop statements has 3 parts :-
  1) Initilization.
  2) condition.
  3) Update.

* 1) Initialization :-

Siddhant Java

Siddhant Java

- It has to be done before the starting of the loop.

* 2) Condition :-

- It is used to control the loop.

a) If condition is true, the loop continueous.
b) If condition is false, the loop stops or terminates.

* 3) Update :-

- An update statement should be executed in every cycle OR iteration of the loop.
- The update statement must help the condition to become false after desired number of cycles.

* 1) while Loop :-

* Syntax :-

```
while (condition)
{
    statements ;
    update ;
}
```

* Program :- 1) Printing "Hellow World" 5 times :-

```
class While1
{
  public static void main (String [] args)
  {
    int start =0

    while (start <5)
    {
      System.out.println ("Hellow World!!");
      start ++ ;
    }
  }
}
```

// Executes "Hellow World" 5 times.

* 2) Program for starting Number to end number :-

```
import java.util.scanner;
class While2
{
  public static void main(String [] args)
  {
    Scanner s = new Scanner (System.in) ;

    System.out.println ("Enter starting number:");
    int start = s.nextInt ();
    System.out.print ln ("Enter ending number:");
    int end = s.next Int ();

    while (start <= end)
    {
      System.out.println (start) ;
      start ++ ;
    }
  }
}
```

// can use ++start also (i.e. pre or post increment both are acceptable).

* 3) Program for a to z :-

```
class While3
{
  public static void main (String [] args)
  {
      char start = 'a' ;
      char end = 'z' ;

    while (start <= end)
      {
        System.out.println (start);
        start ++ ;
      }
  }
}
```

*** NOTE:-**

- For printing characters with its ASCII value, we have to one change is to :- (In while loop)

* System.out.println (start + " ---> " + (int) start)

* 4) Program for even numbers :- (Betn 2 numbers) :-

```
import java.util.Scanner;
class While4
{
  Public static void main (String [] args);
  {
      Scanner s = new Scanner (System.in);

      System.out.println ("Enter starting number");
      int start = s.nextInt () ;
      System.out.println ("Enter ending number");
      int end = s.nextInt ();

      while (start <= end)
      {
        if (start % 2 == 0)
        {
          System.out.println (start);
        }
        start ++ ;
      }
  }
}
```

* 5) Program for odd numbers betn two numbers :-

```
import java.util.Scanner;
class While5
{
  public static void main (String [] args)
  {
    Scanner s = new Scanner (System.in);

    System.out.println ("Enter starting number");
    int start = s.nextInt ();
    System.out.println ("Enter ending number");
    int end = s.nextInt ();
```

Siddhant Java

```
while (start<=end)
  {
     if ( start%2 ==1)
     {
       System.out.println (start);
     }
     start++;
  }
}
}
```

* 6) Odd/Even Alphabets list program :-

```
import java.util.Scanner;
class While6
{
  public static void main (String [] args)
  {
    Scanner s= new Scanner (System.in);

    System.out.println("Enter starting character");
    Char start= s.next().charAt(0);
    System.out.println ("Enter ending character");
    char end = s.next().charAt (0);

    while (start <= end)
    { if (start %2 ==1)
        { System.out.println (start);
        }
      start++ ;
    }
  }
}
```

odd alphabets
(For even :- start%2==0)

* 7) Program for Reverse ordered number :-

```
import java.util.Scanner;
class While7
{
  public static void main (String [] args)
  {
    Scanner s= new Scanner (System.in);

    System.out.println ("Enter starting number");
    int start = s.nextInt ();
    System.out.println ("Enter ending number");
    int end= s.nextInt ();

    while (start >=end)
    {
      System.out.println (start);
      start -- ;
    }
  }
}
```

* 8) Program for Reverse Ordered Characters or Alphabets :-

```
import java.util.Scanner;
class While8
{
  Public static void main (String [] args)
  {
    Scanner s= new Scanner(System.in);
```


Siddhant Java

```
System.out.println("Enter starting alphabet");
char start = s.next().charAt(0);
System.out.println("Enter ending alphabet");
char end = s.next().charAt(0);

while (start >= end)
{
  System.out.println(start);
  start --;
}
}
}
```

9) Program on Table formation using while loop :-

```
import java.util.Scanner;
class While9
{
  public static void main (String [] args)
  {
    Scanner s = new Scanner (System.in);

    System.out.println ("Enter the number");
    int num = s.nextInt();

    int a = 1;
    while (a<=10)
    { System.out.pln (num + "*" + a + "=" + num*a);
      a++;
    }
  }
}
```

Tuesday
15/02/2022

* 2) do-while loop :-

* Syntax :-

```
do
{
  statements;
}
while (condition);
```

- do-while is a loop statement, where the work is done first and then the condition is checked.
- i.e. The control first goes to the do block and executes the statements.
- After the execution of statements, the control goes to while to check for the condition.
- If the condition is true, the loop continuous.
- If the condition is false, the loop terminates.

* The minimum number of execution of **do-while** loop is 1.
* The minimum number of execution of **while** loop is 0.

* **Programs :-** 1) Printing HW for 5 times using do-while :-

```
class DoWhile1
{ public static void main (String [] args)
  { int start = 0;
    do
    { System.out.pln("Hello World");
      start ++;
    }
    while (start<5);
  }
}
```

// ~~HW executed for~~ 5 times

Siddhant Java

Siddhant Java

* 2) Program for Password Function (Mobile lock) :-

```
import java.util.scanner;
class DoWhile2
{
 public static void main (String [] args)
 {
    Scanner s = new Scanner (System.in);

    int times = 0;
    int pin = 1234;

   do
    {
    System.out.println("Enter your pin : ");
    int pass = s.nextInt();

    if (pin == pass)
     {
      Syst.out.pln("Login Successfully");
      break;
     }
    else                    "pin incorrect... Retry"
     { Syst.out.pln(" Try after some time");
    }
 }
while (times < 5);

if (times == 5)
{ sy.out.pln ("Try after 30 sec.");
}
}
}
```

Wednesday
16/02/22

* 3) For loop :-

- It is a loop statement, used to executes set of instructions repeatedly.

* Syntax :-

```
For (initialization ; condition; update)
{
    statements ;
}
```

- For loop has three parts :-

* 1) Initialization :-

- Here we can declare the variable directly.

* 2) Condition :-

- Here, programmer can define any condition, whose result is boolean type.

* 3) Update :-

- Once all the instructions of loop block are executed, control goes to update part of the loop.

- This keeps on repeating until the condition becomes falls.

* Programs :-

Siddhant Java

* 1) Program for printing HW using for loop:-

```
class For1
{
  public static void main (String [] args)
  {
    for( int i=0 ; i<5 ; i++)
      {
        System.out. println("Hellow World");
      }        //executes 5times HW.
  }
}
```

* Practice Programs Of Loops *

* 1) Program to Reverse a given number:-(Using while loop)

```
class Pr1
{
  public static void main (String [] args)
  {
    int num= 1234 ;
    int ans= 0 ;

    while (num>0)
    {
      int rem = num%10 ;
      ans= ans*10 + num ;
      num= num/10 ;
    }
    System.out.pln(ans);      // (4321)
  }
}
```

* 2) Reverse a given number using do-while loop:-

```
import java.util.Scanner
class Pr2
{
  public static void main (String [] args)
  {
    Scanner s = new Scanner (System.in);

    System.out.Print ln ("Enter a number");
    int num= s.nextInt ();
    int ans= 0;

    do
    {
      int rem = num%10 ;
      ans= ans*10 + num ;
      num= num/10 ;
    }

    while (num>0);

    System.out.pln(ans);
  }
}
```

* 3) Program for Palindrome checking:-

```
import java.util.Scanner
Class Pr3
{ public static void main (String [] args)
```

Thursday
17/02/2022

```
{
    Scanner s = new Scanner (System.in);
    Sy.out.pln ("Enter a number");
    int num = s.nextInt ();
    int ans = 0;

    for (int i = num ; i > 0 ; i = i/10)
    {
        int rem = i % 10 ;
        ans = ans * 10 + rem ;
    }
    if (num == ans)
    {
        sy.out.pln ("It is a Palindrome");
    }
    else
    {
        sy.out.pln ("It is not a Palindrome");
    }
}
}
```

* In while loop, for checking Palindrome, there must be use of an duplicate variable as int temp = num ; , Such that, we can check palindrom with "temp" variable value, by compaining ans == temp.
(similar for do-while loop).

* Nested Loops :-

- A loop inside another loop, is known as "Nested Loops".
- Here, the complete execution of 1 iteration of outer loop results in complete execution of inner loop.

* Syntax :-

```
for (int i=0 ; i<5 ; i++)
{
    for (int k=0 ; k<5 ; k++)
    {
        System.out.pln ("Good Morning");
    }
}
```

- Here, for loop is considered :-

* Programs :-

*1)

```
class Nested1
{
    public static void main (String [] args)
    {
        int count = 0
        for (int i=0 ; i<5 ; i++)
        {
            for (int k=0 ; k<5 ; k++)
            {
                Sy.out.Pln ("Good morning");
                count ++ ;
            }
        }
        System.out.pln (count);   // Printed "G.m" 25 times.
    }
}
```

Siddhant Java

Siddhant Java

* Pattern Programs :-

* 1) Square Pattern :-

```
****
****
****
****
```

```
j →
        *       *       *       *
(i,j) = (1,1)   (1,2)   (1,3)   (1,4)
i ↓
        *       *       *       *
        (2,1)   (2,2)   (2,3)   (2,4)
        *       *       *       *
        (3,1)   (3,2)   (3,3)   (3,4)
        *       *       *       *
        (4,1)   (4,2)   (4,3)   (4,4)
```

```
class Pattern1
{
  public static void main (string [] args)
  {
    For (int i=1, i<=4, i++)
      {
        For (int j=1, j<=4, j++)
          { System.out.print ("*") ;
          }
        System.out.println ();
      }
  }
}
```

* 2) Square Pattern having border only :-

```
j →
i ↓
* * * *
*     *
*     *
* * * *
```

```
Class Pattern2
{
  public static void main (string [] args)
  {
    For(int i = 1 ; i<=4 ; i++)
    {
      For (int j=1; j<=4 ; j++)
      { if (i==1 || j==1 || i==4 || j==4)
        { System.out.print ("*"+" ");
        }
        else
        {
          System.out.print ("  "+"  ");
        }
      } System.out.println ();
    }
  }
}
```

(space just for good design) only

* 3)

```
*
  *
    *
      *
```

=> • Only change condition as:-

∴ if (i==j)

* 4)

```
*       *
  *   *
  *   *
*       *
```

=> • Condition

∴ if (i=j || (i+j==5)

Friday
18/02/2022

* Methods :-

- Methods are the block of instructions design to perform specific task.
- Methods are used to achieve code modularity.

* Code Modularity :-

- It is design of breaking the entire task into smaller modules and submodules & creating specific method for each and every module and submodule, is known as "code modularity".

* Advantages of Methods :-

1) Code Reusability :-

- The method can be called any number of times.

2) Easy code modification.

3) Easy bug fixing.

4) Readability of code increases.

* NOTE :-

- A programmer can use methods which are already designed by someone. (Built in methods).
- A programmer can design and create his own method. That method can be used by himself or someone else. (User defined methods).

* Syntax of Method :-

Access Modifier modifier return type method Name { Formal Arguments }

{
  Method block or Method body, it contains statements;
}

* 1) Formal Arguments :-

- Variables created in the method declaration statement, are known as "Formal arguments".
- Formal arguments are used, if a method has to accept any input data.

* 2) Method Name :-

- Method name is an identifire, used to, ~~ind~~ identify or call a method.
- Follow the conventions given to name a method.

Siddhant Java

Siddhant Java

* 3> Return type :-

- Return type specifies either the type of value is return or nothing is returned after the execution of the method.
- The return type can be either "void" or a "specific data type".
- If a method returns nothing, the return type of method, should be "void".
- If a method want to return something, the return type must be a "data type". (It might be premitive or non-premitive or void)

* 4> Modifier :-

- Modifiers are the keywords, which modifies a behaviour of java member or component.
- Modifiers are of two types :-
  - a> Access modifiers.
  - b> Non-access modifiers

* a> Access modifiers :-

- This are the keywords which alter the behavior of java member according to its accesibility (visibility).

Sundaram

- There are four access modifiers as :-
  - 1> public
  - 2> private
  - 3> protected
  - 4> default.

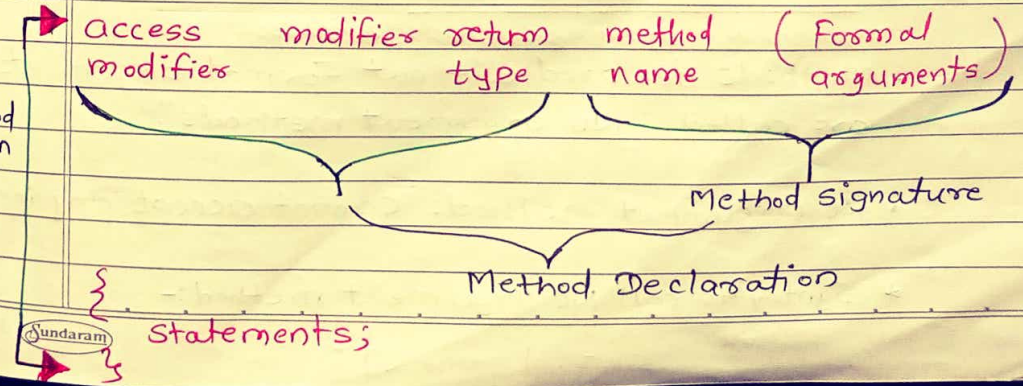
* b> Non-access modifiers :-

- Non-access modifiers are :-
  - 1> static
  - 2> abstract
  - 3> final
  - 4> Synchronize
  - 5> volatile, etc.

* 5> Method block / body / implementation :-

- Here, we write the java statements step by step to perform the specific task.

* Terminologies of Method :-



Sundaram

Tuesday
22/02/2022

• 1) **Method signature :-**

• It is combination of method name and formal arguments.

• 2) **Method Declaration :-**

• It is combination of access modifier, modifier, return type and method signature.

• 3) **Method Definition :-**

• It is combination of method declaration and method block.

** **Types of methods :-**

• Methods are of two types based on formal arguments :-

1) No argument method.
2) Parameterized method.

* 1) **No argument method :-**

• Methods declared without formal arguments, are called "No argument methods".

• No argument methods cannot accept any input

* Syntax to call No argument method :-

```
method name ();
```

* **Program :-**

1)
```
class M1
{
    public static void main (String [] args)
    {
        System.out.println("Hellow World");

        sid ();          // Method call statement
    }

    public static void sid ()    // No argument method
    {
        System.out.println("I am sid method");
    }
}
```

// Hellow World
I am sid method. (O/P)

* **NOTE :-**

1) • A method gets executed only when it is called.
• When we didnt called method in main method, the sid method is only compiled, but not get executed.
• i.e. only "Hellow world" gets executed.

2) • A method must be called using a method call statement.

* ** **Method call statement :-**

• The statement which is used to ~~execute~~ invoke a method for execution, is called "mcs".

Siddhant Java

3) A method can be called any number of times.

• * **Work of main method :-**

1) Execution starts from main method.

2) Main method is called by JVM.

3) Execution Flow is controlled by the main method.

4) Execution ends at main method.

5) Main method is an Parameterized method.

* **Program :-**

```
* 1) Class M2
   {
     public static void main (String [] args)
     {
       Syst.out.pln("Main begin");
       Syst.out.pln (" Hellow world");
       abc();
       Syst.out.pln (" Main end");
     }
     Public static void abc()
     { Sy.out.pln ("From method abc()"); }
   }
```

// Main begin
Hellow world
From method abc()
Main end.

* **NOTE :-**

1) • A variable declared inside the method block, stays local to that method and we cannot use it, in any other methods.

* **e.g. :-** **Program :-**

```
1) class M3
   {
     public static void xyz ()
     {
       system.out.println (a);      // * error :- Not found "a" because it is local to main method only
     }

     Public static void main (String [] args)
     {
       int a = 10;
       System.out.println (a);      // 10
       xyz ();
     }
   }
```

2) We can design a class with any number of method

3) We can have one or more methods which has local variables declared with the same name.

* **e.g. :-**

```
public static void demo ()
{ double a = 10.50;          → // 10.50
  S.O.pln(a); }

Public static void main (String [] args)
{ int a=10; S.O.pln(a); }    → // 10
```

Siddhant Java

Thursday
24/02/22

* 2) Parameterized Method :-

- Methods with formal arguments, are known as "Parameterized methods".
- It is used to accept data when called.

* Formal arguments :-

- The variable declared in the method declaration, is called as "formal arguments".
- Formal arguments are the local variables to that method.
- Formal arguments will be initialized with the data, when the method is called.

* Actual argument :-

- The arguments passed (data or value) is called in the method call statement, is known as "actual arguments".

* Program :-

- 1)

```
class P1
{
  public static void main (string [] args)
  {
    add (10,20) ;
  }
```

Actual argument

```
  public static void add (int a, int b)
  {
    int res = a+b ;
    System.out.println(res);
  }
}
```

Formal argument

- 2) Parameterized method for sub, mut & division :-

```
class P2
{
  public static void main (string [] args)
  {
    sub (20,10) ;
    multi (5, 4) ;
    divide (5.0, 2.0) ;
  }
  public static void sub (int a, int b) //F.A
  {
    int res = a-b ;
    System.out.print ln (res) ;   // 10
  }
  Public static void multi (int a, int b) //F.A
  {
    int res = a*b ;
    System.out.println (res) ;   //20
  }
  public static void divide (double a, double b)
  {                                          //F.A.
    double res = a/b ;
    System.out.print ln (res) ;  // 2.5
  }
}
```

Actual arguments

Siddhant Java

Siddhant Java

* 3) Program for largest value :-

```
class P3
{
  public static void main (string [] args)
  {
    largest (20, 10) ;
  }

  public static void largest (int a, int b)
  { int res = a>b ? a : b ;

    System.out.println (res) ;
  }
}
```

* 4) Design a method which can accept a character and prints its ASCII value :-

```
class P4
{
  public static void main (string [] args)
  {
      chart ('a', ) ;
  }
  public static void chart (char a, )
  {
      int result = a ;
      System.out.println (result); // 97
  }
}
```

* 5) Design a method which accept ASCII value and prints its character :-

```
class P5
{
  public static void chart (int a)
  { if ( a >= 0 && a <= 127 )
    { char ans
        = (char) a ;
      System.out.println (ans); // a
    }
  else
    { System.out.println ("Invalid I/P") ;
    }
  }

  Public static void main (String [] args)
  {
      chart (120) ;
  }
}
```

* What value we can pass to :- add (int a, int b) ≡

1	add (1,2) ✓	5	add (121,21) ✗
2	add ('a',2) ✓	6	add ("hi",3) ✗
3	add ('s','d') ✓	7	add (true, false) ✗
4	add (1.23, 21) ✗	8	add (25f, 30) ✗

Siddhant Java

* Practice Programs *

Saturday 26/02/2022

1) * Method for Reversing a number:-

```
* class P71
{
  public static void main (string [] args)
  {
     reverse (12) ;
  }

  Public static void reverse (int a)
  {
     int ans=0
    do
    {
      int rem = a %10 ;
      ans = ans* 10 + a ;
       a =  a/10 ;
    }
    while (a 70)

    System.out.println( ans)
  }
}
```

* For Loop:-

```
      int ans=0.
* for (int i=0 ; i>0 , i/=10)
  { int rem= i%.10
      ans = ans * 10 + rem
  }
```

* 2) Method for square & cube of number:-

```
* class P12
{
  public static void square (int a)
  {
      int ans = (a*a) ;
      System.out.println (ans) ;
  }

  Public static void cube (int b)
  {
      int res =(b*b*b) ;
      System.out.println (res) ;
  }

  public static void main (string [] args)
  {
      square (15) ;
      cube (20) ;
  }
```

* 3) Rounding up a decimal number :-

```
* Class P13
{
  public static void main (string [] args)
  {
     double num = 14.25
     double rem = num %1 ;
```

```
if (rem < 0.5)
  { int ans = (int)num ;
    System.out.println (ans);      // 14
  }

else
  {
    int ans = (int)num+1;
    System.out.println (ans);
  }
}
}
```

. Here, if input num=14.75
then O/P = 15

- Previous value → Floor
- Next value → Ceiling

* Assignment :-

1) Write a java program to find the sum of given digits of a number.
2) Write a java program which will print the count of a given number's digit.
3) Design a method which can accept a number and prints its multi. table of a given number.

*)

* 1) Program of sum of digits of given number :-

```
class DigitSum
{
  public static void sum (int a)
  {
      int rem=0 ;
      int sum=0;

    while (a>0)        // Also (a!=0)
      {
        rem = a % 10;
        sum = sum + rem;
        a = a/10;
      }
    System.out.println ("The sum of digits of given
                          number is " + sum);
  }

  public static void main (string [] args)
  {
    sum (45);
  }
}                          // sum is 9.
```

* 2) Program for count of digits of given number :-

```
class DigitCount
{
  public static void count (int a)
  {
    int count = 0;
```

Siddhant Java

```
while (a>0)          // Also, (a!=0)
  {
    a = a/10 ;       // ∵ a/=10
    count ++ ;
  }

System.out.println("The sum of digits of given
                    number is" + count);
}

public static void main (String [] args)
  {
    count (45238) ;
  }
}                                  // count is 5
```

* 3) Multiplication Table for given number (Program):-

```
class Table
{
  public static void main (String [] args)
  {
    System.out.println("The multiplication table
                        for given number is");
    table (7) ;
  }

  public static void table ( int num)
  {
    int a=1 ;
    while (a<=10)
    {
      System.out.println (num * a);
      a++;
    }
  }
}
```

Tuesday
01/03/2022

* Practice Program *

* 1) Program for Leap Year :-

- Every 1 year has 365.2425
- But, they use 1 year as 365.25
- Here, they assigns leap year as

  365.25 + 365.25 + 365.25 + 365.25 = 1 day

- So that, error is

```
  365.2500
- 365.2425
  --------
   0.0075
```

- As, they considered,

  0.0075 × 100 = 0.75
  0.0075 × 200 = 1.5
  0.0075 × 300 = 2.25
  0.0075 × 400 = 3 ← 3 days error.

- Hence, they reduces 1 day from each year, to remove that error as:-

100 - 1	500 - 1	900 - 1
200 - 1	600 - 1	1000 - 1
300 - 1	700 - 1	1100 - 1
400 LY	800 LY	1200 LY

Siddhant Java

Siddhant Java

* Program :- (Leap Year)

```
import java.util.Scanner;
class LeapYear
{ public static void main (string [] args)
  { Scanner s = new Scanner (System.in);
    System.out.println ("Enter a year :");

    int year = s.next int ();

    if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0)
      {
        System.out.println ("It is a Leap Year");
      }
    else
      {
        System.out.println ("It is not a Leap Year");
      }
  }
}
```

* 2) Program For Range of Leap year :- (Printing Leap years betn 2 years range)

```
class LeapRange
{
  public static void main (string [] args)
  {
      leap (2000, 3000);
  }
```

```
public static void leap (int a, int b)
{
    For (int i = a, i <= b, i++)
    {
        if (i % 4 == 0 && year % 100 != 0 || year % 400 == 0)
        {
            System.out.println (i);
        }
    }
}
}
}
```

// prints all leap years betn 2000 & 3000

* 3) (A) Program for factorial of a number :- Using For Loop

```
class Factorial
{
  public static void main (string [] args)
  {
      int num = 5;
      int a = 1;
```

// For large values fact. take double.

```
    For (int i = num ; i > 0 ; i--)
      {  a = a * i ;
      }
    System.out.println (a) ;
```

// 120

```
}
```

3) (B) Factorial of a number :- Using while loop

```
class Fact
{
 public static void main (string [] args)
 {
      int num = 3;
      int a  = 1;

      int Fact = 1;

     while (a<=num)
     {
       Fact = Fact * a ;
       a++ ;
     }
     System.out.println (Fact) ;   // 6
 }
}
```

```
int num = 3 ;
int Fact = 1 ;

while (num>0)
{
  Fact = Fact * num;
  num-- ;
}
```

*4) (A) Program on Power of a number:- Using For Loop :-

```
class Power
{
 public static void main (string [] args)
{
      int num = 4 ;
      int power = 3 ;

      int ans = 1;
  for (int i = 1 ; i<=power ; i++)
  {
      ans = ans * num ;
  }
  System.out.println (ans) ;   // 64
 }
}
```

*4) (B) Power of a number :- Using While Loop :-

```
class Pow
{
 public static void main (string [] args)
 {
      int power = 3 ;
      int num = 5 ;

      int ans = 1 ;

     while ( power>0)
      {
        ans = ans * num ;
        power-- ;
      }
     System.out.println (ans) ;   // 125
 }
}
```

Siddhant Java

Sundaram

Thursday
03/03/22

* Practice Programs *

* 1) Program for Fibonacci Series :- (Fibonacci series)

```
class Fib
{
  public static void main (String [] args)
  {
      int num1 = 0 ;
      int num2 = 1 ;
      System.out.println (num1);
      int sum;   —"— (num2);

    for (int i=0; i<20 ; i++)
      {
        sum = num1 + num2 ;
        system.out.println (sum);

        num1 = num2 ;
        num2 = sum ;
      }
  }
}
```

* Fibonacci using method :-

```
Class Fib1
{
  public static void fib (int num1, int num2)
  {
      System.out.println (num1);
      System.out.println (num2);
      int sum;
    for (int i=0 ; i<20 ; i++)
    {
        sum = num1 + num2 ;
        System.out.println (sum);

        num1 = num2 ;
        num2 = sum ;
    }
    public static void main( String [] args)

    {
        fib (0,1) ;
    }
}
```

* 2) Program on Prime number :-

```
class Prime
{
  public static void main (String [] args)
  {
      int num = 7 ;
      int i ;

    for ( i=2 ; i< num ; i++)
      {
        if (num%i==0)
        {
          break ;
        }
      }
```

Siddhant Java

```
        if ( i==num)
        {
         System.out.println("It is a prime no.");
        }
        else
        {
         System.out.println("It is not prime no.");
        }
    }
}
```

* Prime number using Method :-

```
class Prime1
{
 public static void prime(int num)
 {
     int i;
    for ( i=2; i<num; i++)
    {
     if (num % i == 0)
     {
       break;
     }
    }
    if (i==num)
    {
     System.out.println ("It is prime no.");
    }
    else
    {
     System.out.println ("It is not prime no.");
    }
    public static void main (String [] args)
    {
        prime (7);
    }
}
```

* Assignment:-

- 1) Check if the given number is belongs to Fibonacci series.
- 7) Java program to print prime numbers until given range.

* 1) Program for number belongs to Fibonacci series or not :-

* (A) Using for loop :-

```
import java.util.Scanner;
class Fibonacci
{
  public static void main (String [] args)
  {
     Scanner s = new Scanner (System.in);

     System.out.println ("Enter a number:");
     int num = s.nextInt ();

      int num1 = 0;
      int num2 = 1;
      int sum = 0;
```

```
for (int i=0 ; sum<num ; i++)
  {
    sum= num1 + num2;
    num1 = num2;
    num2 = sum;
  }

if (sum==num)
  {
   System.out.println("It is an Fibonacci
                        series number");
  }
else
  {
   System.out.println("It is not an Fibonacci
                        series number");
  }
 }
}
```

* 1) B] Checking is it fibonacci Series number or not :-

* Using while loop :-

```
import java.util.Scanner;
class Fibonacci1
{
 public static void main (String [] args)
 {
    Scanner s= new Scanner (System.in);
    System.out.println("Enter a number:");
    int num = s.nextInt();
    int num1 = 0;
    int num2 = 1;
    int sum = 0;

  while (sum< num)
   {
      sum= num1 + num2;
      num1 = num2;
      num2 = sum;
   }

  if (sum==num)
   {
    System.out.println("It is an Fibonacci
                        series number");
   }
  else
   {
    System.out.println("It is not an Fibonacci
                        series number");
   }
 }
}
```

* 2) * Prime numbers betn two numbers :- (Using for loop)

```
import java.util.Scanner;
class Prime2
{
 public static void main (String [] args)
 {
    Scanner s = new Scanner (System.in);
```

Siddhant Java

```
System.out.println ("Enter first number :");   "starting"
int start = s.nextInt();

System.out.println ("Enter last number :");   "end"
int end = s.nextInt();

  int i;
  int j;

For (i=start; i<=end; i++)
{
   for (j=2; j<=end; j++)
   {
      if (i%j==0)
      {                                  // Nested Loops
         break;
      }
   }

   if (i==j)
   {
     System.out.println(i);
   }
}
}
}
```

Monday 07/03/2022 & Mon. 14/3/22

* Methods with Return Type :-

* Return statement :-

- A return is a control transfer statement.
- Return statement will stop the execution of the current method. The frame gets destroyed and control is transffered back to the method. (i.e. to calling method).

* NOTE :-

* Memory Perspective of JVM while execution :-

- The memory prespective contains 4 types of area as :-
  1) Method area :- All methods declared. (loaded)
  2) Class static area :- All static member decla. (loaded)
  3) Stack area / frame :- Execution frame decl. i.e. all operations. (loade
  4) Heap area

1) In method area, all methods are declared and (loaded) Frame for them is created.
2) In class static area, all static members i.e. methods declared (loaded) with giving some refference numbers to them. (i.e. 01, 02, 03, ... etc)
3) In static area, the execution of program Frame is declared (loaded) and here all operations are done, Also, execution flow done by main method to other members, with reff. numbers assigned to it. (i.e. 0x1, 0x2, 0x3, ... etc)

Siddhant Java

Siddhant Java

* **Program :-** Add & divide with methods by Return type:

```
Class Ret1
{
 public static void main (string [] args)
 {
     int ans = add (10,20);          // 30
     System.out.println(ans);        // 32.0

     System.out.println (division (4,2) + ans);
 }

 public static int add ( int a, int b)
 {
     int result = a+b;
     return result;
 }

 public static double division (double a, double b)
 {
     double result = a/b;
     return result;
 }
}
```

* **Program :-** Print ascii and again print character with even or odd Ascii with Return type:-

```
class Ret2
{
 public static int ascii (char ch)
 {  int ascii = ch;
    return ascii;
 }

 public static void main (string [] args)
 {
     int ans = ascii ('b');          // 98
     System.out.println (ans);

     if (ans % 2 == 0)
      {
       System.out.Pln("It is even ascii value");
      }                              // executes
     else
      {
       System.out.pln("It is odd ascii value");
      }

     char result = asciiToChar (ans);
     System.out.pln( result);        // b
 }

 public static char asciiToChar (int num)
  {
    char ch = (char) num;
    return ch;
  }
}
```

Wednesday
16/03/2022

* Practice Programs on Return type :-

1) Power of a given number :-

```
class Pow1
{
 public static void main (string [] args)
 {
   int ans= pow(2,4);
   system.out.print ln(ans);
 }

 public static int pow(int a,int b)
  {
     int pow= 1;

   for (int i=0, i<b, i++)
    {
      pow= pow*a;
    }

   return pow;
  }
}
```

number
power
number
power

2) **Armstrong Number:-** (1, 153, 370, 371, 407)

- Count to the power of sum of individual digits.
- ie. if num = 123, then power = 3 because digit count is 3.

$\therefore$ ans = sum = $1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36$ ////////

- Also, if given number is equals to executed result i.e. answer or sum, then that number is an "Armstrong number".

* Program :-

```
import java.util.Scanner;
class ArmStrong
{
  public static void main (string [] args)
  {
    Scanner s= new Scanner (System.in);

   System.out.println("Enter the number");
    int num= s.nextInt ();

    int temp= num;
    int count= 0;

   while (temp>0)
   {
      temp /= 10;
      count ++;
   }

   int sum= 0;

   while (num>0)
    {
      int rem= num%10;
        sum= sum + power (rem, count);
        num /= 10;
    }
```

Step Number
1
2
4

Siddhant Java

Siddhant Java

```
System.out.println(sum);
 if (num == sum)                    5
  {
   System.out.println("It is Armstrong Num.");
  }
 else
  {
   System.out.println("It is not Armstrong no.");
  }

public static int power(int base, int exp)
 {
    int pow = 1;                    3
   while ( exp>0)
    {  pow= pow * base;
       exp--;
    }
   return pow;
 }
}
```

4) Write java program to print all the armstrong number betn a range.

3) Write java program to find if given number is strong or not.

5) Java program for print all strong numbers between given range.

*3) Strong Number:- (Program) (1, 2, 145)

- If given number is 123, then $1!+2!+3! = 9$, i.e. here given number and the sum of factorials of each digit is not same.

- Let, a=145, here, Result = $1!+4!+5! = 145$. i.e. a = result, hense, the given number and its sum of factorial of each digit is same, Hence, "145" is an strong number.

→ * Program:-

```
import java.util.Scanner;
class Strong
{
 public static void main(String [] args)
 {
   Scanner s = new Scanner(System.in);
   System.out.println("Enter the number");
   int num= s.nextInt();

   int sum=0
   for (int i=num; i>0; i/=10)
    {
      int rem = i%10;
      sum= sum + Factor(rem);
    }
   System.out.println(sum);

   if (num==sum)
    { System.out.pln("It is an strong Number");
    }
```

```
        else
        { System.out.println("It is not strong no.");
        }
    }
    public static int Factor (int a)
    {
        int fact = 1;

        while (a>0)
        {
            fact = fact * a;
            a--;
        }
        return fact;
    }
}
```

4) Program for All Armstrong numbers between given Range:-

```
import java.util.Scanner;
class Armstrong1
{
 public static void main (String [] args)
 {
   Scanner s = new Scanner (System.in);
   System.out.pln("Enter first number");
   int num1 = s.nextInt();
   System.out.println("Enter second number");
   int num2 = s.nextInt();
   for (int i = num1, i<=num2 ; i++)
   {
       int temp = i ;
       int count = 0;

       while (temp>0)
       {
           temp /=10;
           count ++ ;
       }

       int sum = 0;
       for (int j = i ; j>0 ; j/=10)
       {
           int rem = j % 10;
           sum = sum + power (rem, count);
       }

       if (i==sum)
       {
           System.out.println(sum);
       }
   }
}

public static int power (int rem, int count)
{
    int pow = 1 ;
    while (count >0)
    { pow = pow * rem ;
      count --
    }
    return pow;
}
}
```

// 1,2,3,4,5,6,7,8,9,
153, 370, 371, 407

Siddhant Java

5) Program for Strong numbers between given Range :-

```
import java.util.Scanner;
class Strong1
{
  public static void main(string [] args)
  {
    Scanner s= new Scanner (System.in);

    System.out.pln("Enter First number");
    int num1 = s.next Int ();
    System.out.pln("Enter second number");
    int num2= s.next Int();

    For (int i= num1 ;i<=num2 ; i++)
     {
       int sum=0 ;

      For (int j=i ; j>0 ; j/=10)
       {
         int rem= j % 10 ;
         sum= sum+ factor (rem);
       }

      if (i==sum)
      {
        System.out.println (sum);
      }
    }
  }
```

```
  public static int factor(int rem)
  {
      int fact=1;
      while (rem>0)
      {
        fact= fact * rem;
        rem--;
      }
      return fact;
  }
}
```

// 1, 2, 145

* 6) Neon Number :- (1, 9)

- Let, num=9, then $9^2=81$ & $8+1=9$ i.e. num = sum.
- This is neon number.

- i.e. The sum of digits of square of a given number will equal to num, then it is Neon No.

* Program :-

```
class Neon
{
  public static void main (string [] args)
  {
      neon (9);
  }
  public static void Neon (int num)
  {
```

Siddhant Java

```
int square = num * num ;
int sum=0
while ( square >0)
{
    int rem = square % 10;
    sum = sum + rem;

    square /=10;
}

if (sum == num)
{
  System.out.println ("It is a Neon number");
}

else
{
  System. out. pln("It is not a Neon number");
}
}
}
```

7) Perfect Number :- (6, 28, 496)

- Let, num = 6, its factors are 6 = 1,2,3 & 1+2+3=6.
- Hence, num & sum is same, it is Perfect no.

- When given number and the sum of the factors of that number is same, then it is an "perfect number".

```
* class Perfect
{
  public static void main( string [] args)
  {
      perfect (6);
  }

  public static void perfect (int num)
  {
      int sum=0 ;

      int factor=1 ;

    while (factor < num)
      {
        if ( num % factor ==0)
        {
          sum = sum + factor;
        }

        factor++ ;
      }

    if ( num == sum)
    {
      System.out. pln ("It is an Perfect number");
    }

  else
  {
    System.out. pln ("It is not perfect number");
  }
}
}
```

Siddhant Java

Siddhant Java

* 8) Spy Number :- (1,2,3,4,5,6,7,8,9 and 22,123,132,213,221,312,321)

- Let, num = 123, then 1+2+3=6 & 1×2×3=6.
- Here, digits sum & product is same. i.e. it is an Spy number.
- Hence, if the sum of digits and sum of products of each digit is same, then that number is "Spy number".

```
*  Class Spy
   {
    public static void main (String [] args)
    {
        int num = 123;

        int sum = 0;
        int product = 1;

      while (num > 0)
       {
         int rem = num % 10;
         sum = sum + rem;
         product = product * rem;

         num /= 10;
       }
      if ( sum == product)
      {
       System.out.pln ("It is a Spy number");
      }
      else
      { System.out.pln ("It is not Spy number");
      } } }
```

Assignment

* 9) Program for Neon numbers in given range :-

```
import java.util.Scanner;
class Neon1
{
  public static void main (String [] args)
  {
    Scanner s = new Scanner(System.in);

  System.out.pln("Enter starting number:");
  int num1 = s.next Int ();
  System.out.pln("Enter ending number:);
  int num2 = s.next Int ();

 for ( int i = num1 ; i<=num2 ; i++)
  {
     int square = i * i ;
     int sum = 0;

   while (Square > 0)
    {
      int rem = square % 10 ;
        sum = sum + rem;
      square /= 10 ;
    }

    if (sum == i)
    {
      System.out.println(i);
    }
  }
}
}
}
// 1 and 9 only betn (1-1000)
```

10) Program for Perfect numbers in given range:-

```
import java.util.Scanner;
class Perfect1
{
 Public static void main (string [] args)
{
  Scanner s= new Scanner (System.in)

 System.Out.pln("Enter starting number:");
  int num1= s.next Int();
 System.out.pln("Enter ending number :");
  int num2 =  s.next Int();

 for (int i=num1; i<=num2 ; i++)
  {
     int sum= 0;
     int Factor = 1;

   while (Factor < i)
    {
      if (i % factor ==0)
       {
         sum= sum+ factor;
       }
     factor ++;
    }

   if ( sum==i)
   {
    System.out.println(i);
   }
}
}
}
// 6,28,496 ----- (1-1000)
```

* 11) Program for Spy numbers in given range:-

```
import java.util.Scanner;
class Spy1
{
 public static void main (String [] args)
 {
   Scanner s = new Scanner(System.in);

   System.out.pln("Enter starting number:");
   int num1 = s.next Int();
   System.out.pln("Enter ending number:");
   int num2 = s.next Int();

   for (int i= num1 ; i<=num2 ; i++)
    {
      int sum=0;
      int product = 1;

      for (int j=i ; j>0 ; j/=10)
       {
        int rem = j % 10;
        sum= sum+ rem;
        product = product * rem;
       }

      if (sum== product)
       {
        System.out.println(i);
       }
    }
 }
}
// (1-9, 22, 123, 132, 213, 221)
312, 321
```

Siddhant Java

Thursday
24/03/2022

* Methods with Return Type :-

* Conditions to be Followed while using Return Type:-

- 1) If a return type of method is "void", then use of return statement is optional.

- 2) If a return type is anything otherthan "void", then :-

  a) It means method return some value to the calling method.
  b) Use of return statement is mandatory.
  c) The return statement must have a value or exp. to be returned back to calling method.

- 3) *) Return statement can do widening.

* Program :-

```
1) Class M1
{
  public static void test ()
  {
    Sy.out.println("From void test method"); //exe acts.
    return ;  ------ // Its optional   ②
  }
  public static void main (string [] args)
  {
    System.out.pln("Hellow world"); // executes
    test ();  . . . . . . . . . . . . . . . . ①
  }
}
```

```
2) Class M2
{
  public static int test ()
  {
    System.out.pln("From test () method"); //ex. ②
    return 10+20*5 ;   //  110 .... executed. ③
  }
  public static void main (string [] args)
  {
    System.out.pln("Hellow World"); // executed
    System.out.pln (test ());  ①
```

```
3) Class M3
{
  Public static double test ()
  {
    return 10+20*5 ;  ------ widening
  }
  public static void main (string [] args)
  {
    System.out.println(test ()) ;   // 110.0
    System.out.print-ln (test1 ()); // n
  }
  public static char test1 ();
  {
    return (char) (10+20*5) ;   narrowing.
  }
}
```

* NOTE:- 1) A value returned by a method :-
  a) can be stored inside a variable
  b) can be used directly inside an expression
  c) OR do nothing.

Siddhant Java

Siddhant Java

* Part :- B *

* Static :-

- Static is a keyword and modifier.
- We can prefix static modifier to a java members, hence they are called as the "static members".
- static members are loaded in or take memory in "class static area".
- We can use static members inside any member of a same class or it can also be used in any member which belongs to different class, with the help of class name as a reference.
- We can use a static member without creating an object of a class.
- static members are considered as the "class members".
- static modifier can be prefix to a method or variable or block, created inside a class, they are known as "static members".

1] * Static Method :-

- A method prefixed with static modifier, is known as "static method".

* NOTE :-

1) The method body will be stored inside the method area.

2) The method signature and its reference is stored in class static area.

3) A static method can be used inside the same class, either directly with the help of method name OR with the help of class name as reference.

* Simple Programs :-

1)
```
Class S1
{
 public static void demo ( ) .
 {
   Sy.out.pln ( " demo () From class S1 ");
 }
 public static void main ( string [] args) ;
 {
   Sy.out.pln ( "Hellow world" ) ;
 }
}
```

Siddhant Java

2) class S2
```
{
    public static void main (string [] args)
    {
        S1.demo();
    }
}
```

* Example for calling static method from another class :-

* 1)
```
import java.util.Scanner
class Return
{
  public static void main (string [] args)
  {
    Scanner s= new Scanner(System.in);
    Sy.out.pln ("Enter base number");
    int base = s.next Int ();
    Sy.out.pln ("Enter power value");
    int pow = s.next Int ();

    int ans = power (base, pow);
    Sy.out.pln (ans);
  }
  public static int power (int base, int pow)
    { int ans = 1;
      for (int i = 1, i<= pow, i++)
        { ans = ans * base;
        }
      return ans;
    }
```

* 2)
```
class S3
{
    public static void main (string [] args)
    {
      Sy.out.pln (Return.power (2,4)); // 16
    }
}
```

* NOTE :-

1) • Calling a static method inside a same class ~~with~~ directly or with the help of class name as reference, both are correct.

2) • When we have to call a static method, from one class to another class, using class name as reference is mandatory.

• A static method block is also called as "**static context**"

• i.e. static context = method block of static method

* NOTE :-

1) Inside static context, we can use static members of same class, with the help of class name.

2) We cannot use a non-static member inside a static context directly without creating an object of that class.

• i.e. We can use a non-static member from a static context, with help of object refference.

Saturday 26/03/22

2] * **Static Variables :-**

- A variable declared in the class block and prefixed with static modifier, is known as "static variable".

* **Scope of static variable (Visibility):-**

* 1] A static variable can be used inside any member of the same class or different or any other class.

* e.g :- * ① SV is used in same class member.
* ② SV is called in other class.

* ①
```
Class statVar1
{
static int a = 10; //-------- static variable

public static void main (string [] args)
 {
   System.out.println (a);    // 10
   {
     System.out.println (a);  // 10
   }

  demo ();
 }

Public static void demo()
 {
   System.out.println (a);  // 10
 }
}
```

* ②
```
class statVar2
{
  public static void main (string [] args)
  {
    System.out.println( statVar1.a);  // 10
  }
}
```

* 2] A static variable will be loaded inside the class static area.

* 3] A static variable is given with default values during the loading process. Hence, we can use static variable without initialization.

* e.g.:- static variable uses defaultt values :-
(If they not initialized)

```
class StatVar3
{
    static int a ;
    static double d ;
    static char ch ;
    static boolean b ;
    static String s ;

  public static void main (string [] args)
  {
    System.out.println (a) ;   // 0
    System.out.println (d) ;   // 0.0
    System.out.println (ch) ;  //  Blank space /U0000
    System.out.println (b) ;   // False
    System.out.println (s) ;   // null
  }
}
```

Siddhant Java

Tuesday
29/03/22

* 4] We can have a local variable and globlal variable of same ~~use~~ name. (we can use).

- In this situation, if I use variable name directly, Firstly local variable is reffered but not static variable.

- To use the static variable, when we have a local variable of same name, we need to call a static variable with its refference of the class name.

- e.g.:- Global var. & local variable having same variable name :-

```
class StatVar4
{
    static int a = 10 ;

public static void main (string [] args)
  {
      int a = 20 ;
  System.out.println (a) ;        // 20
  System.out.println (StatVar4.a) ; // 10

    demo () ;
  }

Public static void demo()
  {
    System.out.Print ln (a) ;      // 10
  }
```

** Static Initializers :-

```
Class S1
{
    static int a = 123 ;

public static void main (String [] args)
{
  System.out.pln (a) ;          //123

  int a = 234 ;
  System.out.pln (a) ;          //234

  S1.a = 456 ;  ---------- Ref. required due to local a.
  System.out.pln (S1.a) ;       // 456

    test () ;
}

Public static void test ()
{
  System.out.pln (a)            // 456

  a = 0 ;  --------- Ref. not required because no any local a.
  System.out.pln (a) ;          //0
          or (S1.a) ;
}
```

* NOTE:-
- 1> Static declaration :- static int a ;
- 2> Static Initialization :- ~~static int~~ a=10 ;

Siddhant Java

Siddhant Java

* Static Initializers :-

→ Static variables
→ Static Block

- Static initializers are the members which get executed during the loading process of a class. (i.e. before actual execution of program)

- They get executed before the execution of main method.

- We can use static initializers for :-

  1) To asign the value to the static data members. (Static variables)

  2) It can also be used to perform some activity such as setting up the environment, etc before actual execution is start.

- Static initializer are of two as :-

  1) Static declare & initialization statement.
  2) Static Block.

1) Static Declare & Initialization Statement

- It is used to create a static variable and store value directly into it, before the actual execution.

*** Syntax :-**

[access modifier] static data type var. = value/exp/var.

e.g.:-
1) static int a = 20;
2) static char c = 'b';
3) static string s = "India";

* 2) Static Block :-

- It is a block created in the class block and prefixed with static modifier is known as a "static block"

*** Syntax :-**

```
Class ClassName
{
    static
    {
        statements ;
    }
}
```

static block

*** NOTE :-**

1) A static block does not have name.

2) Static block does not have any return type.

3) We cannot prefix any access modifier to a static block.

4) Static block does not accept any values. (i.e. No formal arguments).

5) Therefore, the programmer cannot call the static block explicitly for execution.

Que] When does a static block gets executed?
(Also static initializers)
→
① • Before the actual execution starts.
② • During loading process.
③ • Before the main method execution start.

* * Simple program :-

```
class S2
{
    static string s= "India";
    static int i;

  public static void main (String [] args)
  {
    System.out.pln ("Hellow World"); ⑦

    int i= 10;
    System.out.pln (i);          ⑧
    System.out.pln (S2.i);       ⑨

    test ();
  }

  public static void test ()
  {
    System.out.pln (s);   ⑤ & ⑩
    System.out.pln (i);   ⑥ & ⑪
  }
```

execution number (sequence)

```
  static
  {
    System.out.pln ("From static block 1"); ①
    System.out.pln (s+ "is my country"); ②
  }

  static
  {
    System.out.pln ("From static block 2"); ③
    i= 20;
  }

  static
  {
    System.out.pln ("From static block 3"); ④
    test ();
  }
}
```

* Output :-

1) From SB1
2) India is my country.
3) From SB2
4) From SB3
5) India
6) 20
7) Hellow world
8) 10
9) 20
10) India
11) 20

Siddhant Java

** Class Loading Process :-

1) A block will be created for a class in class static area.

2) All methods blocks will be loaded inside method area.

- If there are static members, the reference of static methods is stored in the table with signature in the class static area.

- If there are static variable, all static variable will be allocated in the class static area and will be given or assigned with default values.

3) • If there are static initializers, all the static initializers get executed in top to bottom order, during the loading process.

4) This completes the loading process of a class.

Wednesday
30/03/22

* Multiple Classes :-

- It is a design of having more than one class inside the single java file, is known as "multiple classes".

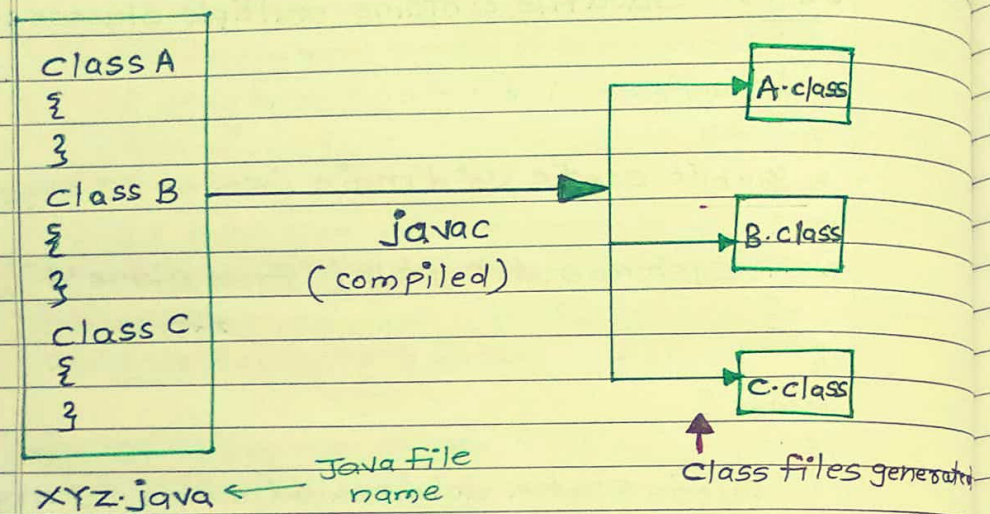
- e.g. :- Java file contains multiple classes :-

```
class A
{
   public static void main (String [] args)
   {
      System.out.println("From class A");
   }
}
class B
{
   public static void main (String [] args)
   {
      System.out.println("From class B");
   }
}
class C
{
   public static void main (String [] args)
   {
      System.out.println("From class C");
   }
}
```

* NOTE :- • Take java file name as XYZ.java.
• When it compiled it generates :- class files of A, B and C. (i.e. 3 files class)

Siddhant Java

- A java file can have more than one java class.
- Compiler will create one class file for every class present in the java file.



* NOTE :-

- In this case, name of java file can be anything.
- But, at execution time we have put or use respective class name given in the that java file. (i.e. here, A, B or C).

- e.g. :- Consider above example :-
  1) For compile :- javac XYZ.java
  2) For run :- java A OR java B OR java C

* Public class :-

- A class which is prefixed with public access modifier, is known as "public class".

* NOTE :-

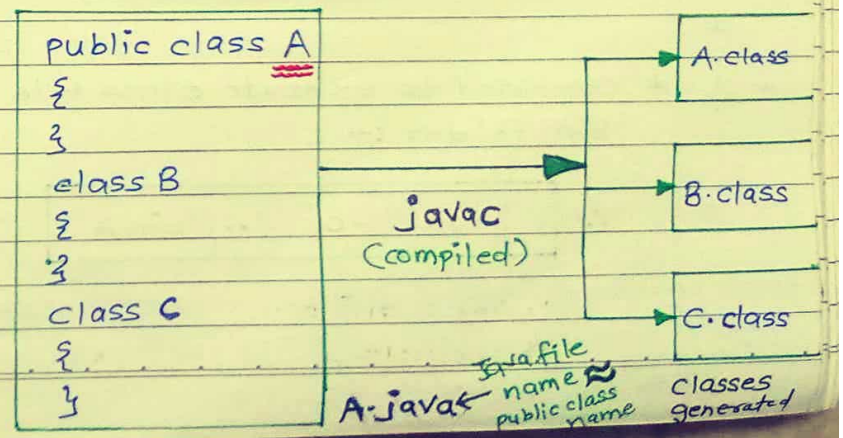
- We cannot use more than one public class inside the single java file.

* Rule :-

1) When java class is public, java file name should be same as class name, else we get compile time error.

2) To achieve multiple classes using public class, we can use any no. of default classes, but only one public class inside a single java file, which provided class name (public) and java file name must be same.

- e.g. :-



Siddhant Java

Siddhant Java

- Here, in this example, a java file contains two class (default class) blocks with only one public class.
- Here, the name of java file should be same as per public class name. i.e. A.
- After compilation it generates three class files as class A, class B and class C.
  1) for compile :- javac A.java
  2) for run :- java A / java B / java C.

* Segregating class files & source files :-

- All the source files are stored in folder named as "**src**".
- All the executable are stored in folder named as "**bin**".

* 1) * Command to compile a java file inside src and creating the class file inside bin folder :-

∴ `javac -d ../bin A.java`

2) * Command to execute class file present in bin folder :-

∴ `java -cp ../bin A`

Friday
01/04/22

* Object Oriented Programming :- (OOP's)

* Object :-

- An object is a block of memory which is created at the run time in the heap area, which represents a real world object.

- An object consist of two things :-
  1) Attributes or properties.
  2) Actions or Behaviours.

* • Attributes are used to define the object.
- Actions or behaviours specify how the object is used.

* • Non-static variables represent the attributes or properties of an objects.
- Non-static methods represents the actions or behaviours.

* Class :-

- In real world, before creating an object a blueprint is created.
- This blueprint provides all specifications of the object.

• e.g.:-

1) • Before manufacturing a car, a blueprint of car is created.

2) • Before building a house, a blueprint of house is created.

• In java, before creating an object a blueprint must be created, which provides specification of object.

• This design is done with the help of a class.

* Define class :-

• A class is user defined, non-primitive data type, which represents the blueprint of an object and provides specifications of the object.

• A class is used to create objects.

* To create an object :-

• We can create object with the help of a keyword "new" (operator) and constructor of class.

* new :-

• It is keyword, also an operator, used to create new objects in a class, with help of constructor.

Sundaram

Siddhant Java

Monday
04/04/22

* e.g.:- • Let us consider a object as "Book".

* Book :- (object)

Attributes	Behaviours
• Type	• Reading.
• size	• writing.
• pages	• drawing.
• colour	• study.
• Publications	
• author	
• title	

* Blue Print of object (Book) :-

```
class Book
{

}
```

* Driver of object (Book) :-

```
class BookDriver
{
    public static void main (String [] args) •
    {
        Book b1 = new Book();
                      ← constructor
        System.out.println (b1); // Book@15db9742
                                    (i.e. Ref.no.)
    }
```

• For compile :- javac -d ../bin BookDriver.java
• For Run :- java -cp ../bin BookDriver

Sundaram

* Memory perspective or Loading process of Object :-

1) Stack Area :-

- Here main method frame is created with its ref. no. as (0x1).
- It contains all the objects followed by the constructors.
  - i.e. Book b1 = new Book ();
    Book b2 = new Book ();
    Book b3 = new Book () ; constructor
- The refference number of memory block or frame of heap area is given back here for further process.

2) Heap Area :-

- Here, constructor will loads all the attributes and behaviours of the objects.
- For every object a memory block is created in the heap area.
- The memory block contains or loaded with its unique refference numbers.
- e.g.:- 1) For Book b1 ≈ Book@15d04324.
  2) For Book b2 ≈ Book@24Fd2892.
- This ref. no. is given to stake area.

* Example for creation of objects using class and new :-

1) Object :- Fruit

- Blue print :-
```
class Fruit
{

}
```
- Driver :-
```
class FruitDriver
{
public static void main (string[]args)
{
    Fruit f1 = new Fruit ();

    System.out.println (f1); // Fruit@15d8742
}
```
(Ref. No.)

- For compile :- javac -d ../bin FruitDriver.java
- For run :- java -cp ../bin FruitDriver.

Siddhant Java

Tuesday
05/04/22

* Non-Primitive Data Types :-

- Every class name is an non-primitive datatype.
- e.g. :- class Book, class Fruit, class Mobile, etc.

- Here, Book, Fruit, Mobile, etc are non-primitive data types.

- with the help of Non-primitive D.T., I can create reference variables, which can stores the ref. of an object created.

* Syntax to create non-primitive variable :-

Non-primitive data type variable ;

[Creation of Object Reference]

- OR

class name variable ;

- e.g. :- 1) Book b1
  2) Fruit f1
  3) Mobile m1.

* NOTE :-

- Default value of a non-primitive variable is "null."

- e.g. :- 1) Book b1 = null ;
  2) Fruit f1 = null ;

A) * Non-Static Variables :-

- The variable declared in the class block without prefixing any static modifier, is known as "Non-static variables".

- For a non-static variable, a memory is allocated inside the object of a class.
- i.e. non-static variables are loaded in the object in the heap area.

- Non-static variable will be created inside a every object created for a class.

- A non-static variable will be assigned with default values, during the loading process.

* Example of non-static variables :-

1) ① Requirements :- 3 Laptops as Brand, Price, Hard disk, RAM

a) Lenove	b) HP	c) Dell	← Brand
40000	60000	50000	← Price
1TB	2TB	1TB	← HD
4GB	8GB	8GB	← RAM

② Blueprint :-

```
class Laptop
{
    string brand;
    double price;
    int hd;
    int ram;
}
```

Siddhant Java

Siddhant Java

③ Driver :-

```
class LaptopDriver
{
public static void main (String [] args)
{
    Laptop l1 = new Laptop ();

    l1.brand = "Lenovo";
    l1.price = 40000.00;
    l1.hd    = 1
    l1.ram   = 4

System.out.println ("Brand:"+ l1.brand);
System.out.println ("Price:" + l1.price);
System.out.println ("Hard disk:"+ l1.hd + "TB");
System.out.println (" RAM:"+ l1.ram + "GB");
System.out.println ("=========");

    Laptop l2 = new Laptop();

    l2.brand = "HP";
    l2.hd    = 2;
    l2.price = 60000.00;
    l2.ram   = 8;

System.out.println ("Brand:"+ l2.brand);
System.out.println ("Price:"+ l2.price);
System.out.println ("HD:"+ l2.hd + "TB");
System.out.println ("RAM:"+ l2.ram + "GB");
System.out.println ("=========");
}
}
```

B) * Non-static Methods :-

- Let us consider, the same example of 3 Laptops.
- Here, we dont need print all the laptop statement in driver file, instead of this we can create the non-static method, which will represent the printing of objects variable as required.

- This non-static method will be / should be create in the Blue-print of given examples object.

① Requirement :- 3 Laptops as :-

	ⓐ	ⓑ	ⓒ
Brand →	Lenovo	HP	Dell
Price →	40000	60000	50000
HD →	1TB	2TB	1TB
RAM →	4GB	8GB	8GB

② Blue Print :-

```
class Laptop
{
    String brand;
    double price;
    int hd;
    int ram;
  public void displayLaptop()  ..... Non-static Method
  {
    System.out.println ("Brand:"+ brand);
    System.out.println ("Price:" + price);
    System.out.println ("RAM:" + ram + "GB");
    System.out.println ("HD:" + hd + "TB");
    System.out.println ("=========");
  }
}
```

Wednesday
06/04/2022

③ Driver :-

```
class LaptopDriver
{
  public static void main (String [] args)
  {
    Laptop l1 = new Laptop ();

      l1.brand = "Lenovo";
      l1.price = 40000.00;
      l1.ram = 4;
      l1.hd = 1;

      l1.displayLaptop ();  ------- Method call stat. with Ref. to l1

    Laptop l2 = new Laptop ();
      l2.brand = "HP";
      l2.price = 60000.00;
      l2.ram = 8;
      l2.hd = 2;

      l2.displayLaptop ();  ------ MCS with Ref. to l2

    Laptop l3 = new Laptop ();
      l3 brand = "Dell";
      l3 price = 50000.00;
      l3 ram = 8;
      l3 hd = 1;

      l3.displayLaptop ();  ------ MCS with Ref. to l3
  }
```

* The methods which are declared in the class block and are not prefixed with static modifier, is called "non-static method".

* The non-static method defination is stored in the method area.

* The reference of the non-static method is stored inside the object of a class.

* A non-static method block is also called as "non-static context".

* This non-static context will hold the reference of the object.
* We can use the non-static members of same class without any refference.

* Calling a non-static method :-

* We can call a non-static method with help of object reference.

* Calling a non-static method from one class to another class :-

* We can call non-static method of one class inside class with help of object reference.

Siddhant Java

Siddhant Java

* Non-static context :-

- A block which belongs to non-static method is called as a "non-static methods".

- In non-static context, we can use :-

1> Static members of class directly.
2> Non-static members of class directly.

* this :-

- It is a keyword.
- It is a non-static reference variable.
- It will hold the reference of the object which is currently under use.

* Purpose of this :- (Use)

- To refer the object members (non-static members) from non-static context.

* e.g.:-

1> If there are global and local nonstatic variables are of same name, we use "this" keyword, in order refer non-static global variable.

- Let us consider a non-static context having global & local variable of name "a", then :-

```
class B                                   ← Blue Print *
{
    int a = 10
    public void display B()
    {
        double a = 3.14 ;

        System.out.println(a) ;          // 3.14
        System.out.println (this.a) ;    // 10
    }
}
```

* Driver:-

```
class BDriver
{
    public static void main (String [] args)
    {
        B b1 = new B() ;
        System.out.println (b1) ;        // 1s@b4297
                                            (Ref No.)
        b1.display B () ;                // 3.14
                                         // 10
    }
}
```

Sundaram
Sundaram

Siddhant Java

* Non-static Initializers :-

- It gets executed during the loading process of an object. (i.e. during creation of object)

- Non-static Initializers are of two types :-
  1) Non-static declare & initialization statement
  2) Non-static block.

* 1) Non-static declare & initialization statement :-

```
data type var = Value / exp / var ;
```

* 2) Non-static Block :-

- A block which is declared in class block without prefixing any modifiers, is called as "non-static block".

```
class ClassName
{
    {
        Statements ;
    }
}
```

Non-static block

1) A non-static block does not have name.
2) It does not have any Return type.
3) It does not have any formal arguments.
4) It cannot be prefixed with any modifiers, hence programmer cannot call non-static block Explicitly, for execution.

Sundaram

Thursday 07/04/2022

* NOTE :-

- Non-static initializers are gets executed during object creation process.

* Method Chaining :-

- The process of calling methods continuosly one after the another, is called "Method chaining".

- Syntax :-

```
ref. m1().m2().m3().------;
```

* NOTE :-

- We can call methods in this fashion, only if the method returns some reference.

- e.g. :- * Blueprint :-

```
class Demo
{
    public Demo m1()
    {
        System.out.println("From Demo class m1()");
        System.out.println(this);
        return this;
    }
    public Demo m2()
    {   system.out.println("From Demo class m2()");
        system.out.println(this);
        return this;
    }
```

Sundaram

```
    public Demo m3 ()
    {
      System.out.println("From Demo Class m3()");
      System.out.println(this);
      return this;
    }
}
```

* Driver :-

```
class DemoDriver
{
  public static void main (String [] args)
  {
      Demo d1 = new Demo ();

      d1.m1().m2().m3();
  }
}
```

```
// • From Demo class m1 ()
  • Demo @ 15db9742
  • From Demo class m2 ()
  • Demo @ 15db9742
  • From Demo class m3 ()
  • Demo @ 15db9742
```

* Constructor :-

- A constructor is a special member of a class whose name is same as class name and which does not have any return type.

- It is also called as a "special method".

* NOTE :-

- A programmer cannot cannot call constructor, but he can only use along with new operator.

* Purpose of constructor :- (Use) :-

1) It is used to load all the non-static members of class into its object.

2) Every class must have a constructor. & it is mandatory.

- If a programmer fails to create any constructor, at compile time, the java compiler will add a no argument constructor into the class, it is called as "default constructor".

```
Class Book                                  Class Book
{                  ------>                  {
                    javac                     Book ()
                                              {
                                              }
}                                           }
```

Blueprint

Blueprint with default constructor

Siddhant Java

Siddhant Java

* Inside the constructor Body :-

1) Instructions to load non-static members will be there.

2) All the non-static initializers of class will be inserted into constructor body.

3) User written instruction of constructor will be there.

* User defined Constructors :-

- The constructors added by the programmers, are called "User defined constructors".
- There are two types of constructors :-
  1) No argument constructor.
  2) Parameterized constructor.

* 1) No Argument Constructor :-

- A constructor without any Formal arguments, are called "No argument constructor".

* 2) Parameterized constructor :-

- Constructors with Formal arguments in it, are called as "Parameterized constructors".

* Examples :-

* 1

* Blueprint :-

```
class Laptop
{
        String brand ;
        double price ;
        int hd ;
        int ram ;

    Laptop ()                                   } No argument constructor
    {                                             (User defined constructor)
    }

    Laptop (String brand, double price, int hd,
            int ram)                            ------// parameterized constructor
    {
        this. brand = brand ;
        this. price = price ;
        this. hd = hd ;
        this. ram = ram ;
    }

    public void displayLaptop ()
    {
      System. out. println ("Brand:" + brand);
      System. out. println ("Price:" + price);
      System. out. println ("HD:" + hd + "TB");
      System. out. println ("RAM:" + ram + "GB");
      System. out. println ("==========");
    }
```

Siddhant Java

* Driver :-

```
class LaptopDriver
{
  public static void main(string [] args)
  {
      Laptop l1 = new Laptop();
      l1.displayLaptop("Lenovo", 40000.00, 1, 4);

      Laptop l2 = new Laptop();
      l2.displayLaptop("HP", 60000.00, 2, 8);

      Laptop l3 = new Laptop();
      l3.displayLaptop("Dell", 50000.00, 1, 8);
  }
}
```

Alter it (values are at first state)

Alter (Put values Here)

Alter

IMP → // Values or data must be given at constructor, not in method call statement.

(2) Blueprint :-

```
class House
{
      string city;
      string location;
      double price;
      int bhk;

  House()
  {
  }

  House(string city, string location, int bhk, double price)
  {
      this.city = city;
      this.location = location;
      this.bhk = bhk;
      this.price = price;
  }

  public void displayHouse()
  {
    System.out.pln("city:"+city);
    System.out.pln("Location:"+ location);
    System.out.pln("BHK:"+ bhk);
    System.out.pln("Price:"+ price);
    System.out.pln("==========");
  }
}
```

* Driver :-

```
class HouseDriver
{ public static void main(string [] args)
  {
    House h1 = new House();
    h1.displayHouse(Pune, Baner, 3, 350000.00);

    House h2 = new House();
    h2.displayHouse(Mumbai, Thane, 2, 320000.00);

    House h3 = new House();
    h3.displayHouse(Pune, Nigdi, 3, 280000.00);
  }
}
```

Alter it

Changes (Put values Here)

Alter it

(Put values in constructor, not in method call state)

Friday
08/04/2022

* Method Overloading :-

- It is a design of having more than one method with same name inside the same class, is known as "method overloading".

- Here, method name can be same, but the methods must differ in its length of formal argument or it must be differ with its type of formal argument.

- Method overloading is an example of the compile time polymorphism (compile time binding) i.e, the compiler decides which method should be executed according to the actual arguments, passed by the programmer.

* Compile time binding :- (Polymorphism)

- The binding done between the method call statement to the method defination at the compile time, by considering the length and type of actual argument to the length and type of formal argument, is known as "compile time binding".

- Here, compiler decides the method to be executed.

* Example :-

```
class Addition
{
    public static void main (String [] args)
    {
        add (10,20) ;        ..... 1st add executed
        add (10,20, 30) ;    ..... 2nd add executed
        add (5.0 , 10 ) ;    ..... 3rd add executed
                                   (method)
    }

    public static void add (int a ,int b)
    {
        System.out.println.(a+b) ;      // 30
    }

    Public static void add (int a, int b, int c)
    {
        System.out. println (a+b+c) ;   // 60
    }

    public static void add (double d , int a)
    {
        System.out. println ( d+a) ;    // 10.5
    }
}
```

Siddhant Java

Siddhant Java

* Constructor Overloading :-

- A class having more than one constructor is known as "constructor overloading".

* NOTE :- (Rules) :-

1> Name of the constructor must be same.

2> The formal argument of constructor, must change either in its type or no. of formal arguments.

3> It also called as "compile time binding or polymorphism".

* Example :-

* Blue print :-

* • Let us consider a example of Form which will includes Name, phone, email, date of birth, telephone, blood group and address fields.

- Here name, phone, email and dob are the mandatory fields.
- Consider :-
  1. User enters all mandatory fields only.
  2. User enters mandatory fields with telephone only.
  3. User enters all the fields of forms.

```
class Forms
{
        String name;
        long phone;
        String email;
        String dob;
        long tel;
        String blood;
        String address;

    Forms ()
    {
    }

    Forms (String name, long phone, String email,
            String dob)
    {
        this.name = name;
        this.phone = phone;
        this.email = email;
        this.dob = dob;
    }

    Forms (String name, long phone, String email,
            String dob, long tel)
    {
        this.name = name;
        this.phone = phone;
        this.email = email;
        this.dob = dob;
        this.tel = tel;
```

```
Forms (string name, long phone, string email,
        string dob, long tel, string blood)
{
    this.name = name ;
    this.phone = phone ;
    this.email = email ;
    this.dob = dob ;
    this.tel = tel ;
    this.blood = blood ;
}

Forms (string name, long phone, string email,
        string dob, long tel, string blood,
        string address)
{
    this.name = name ;
    this.phone = phone ;
    this.email = email ;
    this.dob = dob ;
    this.tel = tel ;
    this.blood = blood ;
    this.address = address;
}
public void displayForms ()
{   Sy.out.pln ("Name:" + name);
    Sy.out.pln ("Phone No.:" + phone);
    Sy.out.pln ("Email ID:" + email);
    Sy.out.pln ("DOB:" + dob);
    Sy.out.pln ("Telephone:" + tel);
    Sy.out.pln ("Blood group" + blood);
    Sy.out.pln ("Address" + address);
    Sy.out.pln ("==========");
}}
```

* Driver :-

```
class FormsDriver
{
    public static void main (string [] args)
    {
    • Forms f1 = new Forms ("Rahul", 828729, "abc@xyz",
                            "14/06/1998");

      f1.displayForms ();

    • Forms f2 = new Forms ("Anant", 23456, "xyz@abc",
                            "19/08/92", 852823, "At")

      f2.displayForms ();
    }
}
```

* • Here, the execution takes place as :-

① f1 = when user enters only mandatory fields, f1 will executes.

② f2 = when user enters mandatory fields with telephone and blood group, then f2 get executed.

③ f3 = Also, when user enters all the fields of form, then f3 which contains all the containts, so that, f3 get executed.

Siddhant Java

Monday
11/04/2022

* Loading process of an Object :-

1) "new" operator creates a block a memory for the object inside the heap area and returns the reference of the block.

2) The constructor is called for execution

a) It loads the non-static members of class.
b) Non-static variables are assigned with default values.
c) Non-static method reference is stored in table with its method signature.
d) All non-static initializers get executed in top to bottom order.

3) User written statements of constructor gets executed.

4) Execution of constructor is completed. This ends the loading process of the object.

* Constructor Chaining :-

- The data inside constructor is :-

a) Load instruction
b) Load user defined instructions.

} work of an constructor.

- Here, we calls the constructor inside another constructor.

- It is a design or way of calling one constructor inside another constructor. is called as the "constructor chaining".

- We can achieve constructor chaining in

1) this () statement (call)
2) super () statement (call)

* 1) * this call statement [this()] :-

- This statements are used to achieve constructor chaining within same class.

* Rules :-

1) this call statement can be used only inside a constructor.

2) It should be the first statement inside the constructor body.

3) Recursive call of constructor is not allowed. (A constructor calling itself is not allowed).

4) If a class has "n" numbers of constructors, we can use this call statement inside "n-1" constructors.

- Let us consider the example of form as :-

Siddhant Java

Siddhant Java

* Example :-

* Blue Print :-

```
class Form
{
        String name ;
        long phone ;
        String email ;
        String dob ;
        long tel ;
        String blood ;
        String address ;

    Form () { }

    Form (String name, long phone, String email,
            String dob)
    {
        this () ;
        this . name = name ;
        this . phone = phone ;
        this . email = email ;
        this . dob = dob ;
    }

    form (String name, long phone, String email,
            String dob , long tel )
    {
        this( name, phone, email , dob) ;
        this . tel = tel ;
    }

    Form (String name, long phone, String email,
            String dob, long tel , String blood )
    {
        this (name, phone, email , dob, tel) ;
        this. blood = blood ;
    }

    Form (String name, long phone, String email, String
            dob, long tel, String blood, String address)
    {
        this (name, phone, email, dob, tel, blood) ;
        this. address = address ;
    }

    public void displayForm ( )
    {
        System. out. println (name) ;
        System. out. println ( phone) ;
        System. out. println (email) ;
        System. out. println (dob) ;
        System. out. print ln (tel) ;
        System. out. print ln (blood) ;
        System . out. print ln (address) ;
    }
}
```

• Here, the Constructor chaining is done with the help of this call statement [this()].

Siddhant Java

* Driver :-

```
class FormDriver
{
  public static void main (String [] args)
  {
    Form f1 = new Form("Sagar", 2853291, "abc@.com",
                       "15/08/2003");

      f1.displayForm();

    Form f2 = new Form("Sagar", 85242141, "x@.com",
                       "10/10/22", 234561, "xt",
                       "Pune");

      f2.displayForm();
  }
}
```

- Here, for Form f1, only the first block of constructor is executed with the default constructor block.
- For Form f2, all the Form blocks of constructors are get executed, due to all are called with respect to its formal arguments type and length.
- Also, the default constructor block also get executed.

Tuesday
12/04/2022

* Packages :-

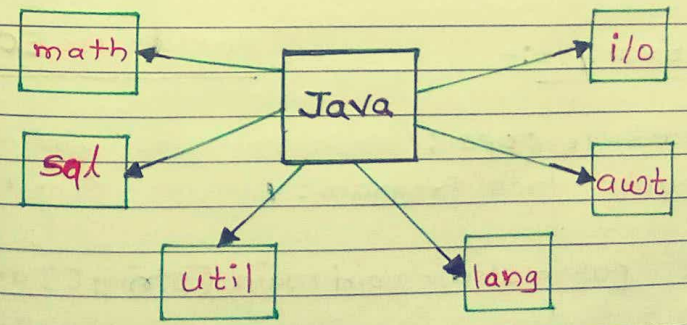
- It is a mechanism to organise and group the similar java files or A group of related classes and interfaces grouped together to form a single entity are known as the "packages".

- Packages are of two types :-

  1) User defined packages.
  2) Built-in packages.

* 1) Built-in packages :-

- Packages created by the developers of the language, which are available to use directly, are known as "built in packages".



- Here, java is main package, which contains numbers of sub-packages like util, lang, awt, io, math, sql, and many other functional packages.

Siddhant Java

*2) User defined packages :-

- We can create a package with the help of a keyword called "package".
- The packages created by the user, are called "user defined packages".

* Syntax to create Package :-

∴ `package packagename ;`

- The instructions of creating a package, must be the first instruction in java program.
- i.e. The first statement of a program is an package statement & then import statement.

* Example :- [Now, we use **Eclipse**]

```
package oops ;
public class Program1
{
    public static void main (String[] args)
    {
        System.out.println ("Hellow World");
    }
}
```


Wednesday 13/04/2022

* Advantages of Packages :-

1) Modularity of the software is increased.
2) Maintainability of software is increased.
3) Code reusability.
4) Security is increased.
5) Easy bug fixing.

* Principles of Object Oriented Programming :-

- Object Oriented programming has four principles, which help the programmer to design softwares easily.
- The principles are :-

    1) Encapsulation
    2) Inheritance.
    3) Polymorphism.
    4) Abstraction.

- This principles will help the programmer to design software according with privacy, security, importance, data hiding, multiple usage, code reduction, code reusability, protection and many more.

* 1) Encapsulation :-

- The process of binding or wrapping the attributes and behaviours of an object, is known as "Encapsulation".

* NOTE :-

1) We should not give direct access to the attributes of class (object) to the user.

2) But, the user should be able to read or modify the attributes of a class (object) through behaviours or actions.

3) Therefore, it is a good practice of a programmer to design a class such that, we can have methods to read the attributes, as well as update the attributes.

* Advantages :-

1) Since, we provide indirect access to it is secure and we can have validation steps before reading or updating the attribute.

2) Here, data hiding can be achieved.

3) Protection & Security of an object will be increased.

* Data hiding :-

- It is the process of restricting direct access to the attributes of a class (object) and providing secure and indirect access to the attributes of class (object) is called as the "data hiding".

* Step to achieve Data Hiding :-

1) Make the attributes private.

2) Design "getter" and "setter" methods for read and write operation respectively.

3) A programmer gets **flexibility** to design the following :-

   a) Make any attribute only readable & not modifyable.

   b) Make an attribute modifiable but not readable.

   c) Make an attribute both readable and modifiable.

   d) Make an attribute neither readable nor modifiable.

- Let us consider an example on encapsulation as :-

Siddhant Java

* Example :-

- Let us consider an ATM machines example, which consist its name, location, money and ID as its attributes.
- Here, all attributes are only for read only for the user, i.e. user cant interchange in value or data inside it.
- Here, we use getter method to name, location and balance check. For withdraw we use setter method.

* Blueprint :-

```
package encapsulation;
public class ATM
{
    private string name ;
    private string location;
    private double money ;
    private int id ;

  ATM () { }

  ATM (string name, string location , double money,
        int id)
  {
      this.name = name ;
      this. location = location ;
      this. money = money ;
      this. id = id ;
  }

  public string getName ()
  {
      return name ;
  }

  public string getLocation ()
  {
      return location;
  }

  public string getBalance (long acno , int pin)
  {
      long acno1 = 12345678 1 ;
      int pin1 = 1234 ;
      string bal = "5000.00" ;

    if (acno == acno1 && pin == pin1)
    {
        return bal ;
    }
    return "Invalid entry .... Retry" ;
  }

  public void setMoney (long acno , int pin,
                        double amount)
  {
      long acno1= 12345678 1 ;
      int pin1 = 1234 ;

  if( acno == acno1 && pin == pin1) {

      System. out. println ("Login Successfully");
```

Siddhant Java

```
if (amount <= money)
{
    money = money - amount ;

  System.out.println ("Withdraw Successfully");
}
else
{
  System.out.pln ("No cash");
}
}
else
{
  System.out.pln ("Invalid entry...Retry");
}
}
```

* Driver :-

```
Package encapsulation;
public class ATMDriver {
 public static void main (String [] args)
 {
   ATM a1 = new ATM ("SBI", "Deccan", 5000000.00, 18924);

   System.out.pln (a1.getName ()) ; ①

   System.out.pln (a1.getBalance (12345678, 1234)); ②

   a1.setMoney (123456789, 1234, 1000.0); ②④
 }
}
```

// ① SBI ③ Login Successfully!!
② 5000.00 ④ Withdraw Successfully!!

Thursday
14/04/2022

* * Relationships :-

- It is nothing but association between two objects.
- There are two types of relationship :-
  1> Has-A relationship.
  2> Is-A relationship.

* * 1> Has-A relationship :-

- It is the association betn two or more objects, such that, they are dependent on each other, is known as "Has-A relationship".
- It is classified into two types based on the levels of dependancy as :-
  a> Composition.
  b> Aggregation.

* a> Composition :-

- It is the association betn two objects, such that one object cannot exist without another object.
- They behave like part & whole relationship.

* e.g. :- Body & Heart, Engine & car, TV and Remote, Mobile & charger, tree & leaves, etc.

Siddhant Java

*b) Aggregation :-

- It is the association between two object, there is dependancy but one can stay without another object.

*e.g. :- Phone & sim, Fish & water, bottle & water, Tractor & trolley, Bike & petrol, etc.

** How to achieve **Has-A relationship** :-

1) One object should have refference of another object.

2) Here, we can easily achieve this design by creating ref. variable of one object inside another class.

3) We can achieve Has-A relationship using two

a) Early instanciation
OR Early binding.

b) Lazy instanciation
OR Lazy binding.

* a) Early Instanciation (Early Binding) :-

- In early instanciation, the instance of dependent object implicitly created when the instance of a class is created.

- This design is achieved with the help of non-static initializers.

- In above design we have achieved Has-A relationship, where a car object have reference of engine object.

- We can achieve either composition or aggregation by creating reference of one object into the another object.

- When service user of car class, create an object of car, automatically an engine object gets created. & reference is stored inside car object.

* Blue Print of Engine :-

```
package encapsulation;
public class Engine
{
    string fuel ;
    int cylinders ;
    double hp ;
    double cc ;
    string type ;

  Engine ()
  { }
```

Siddhant Java

```
Engine (String fuel, int cylinders, double hp,
        double cc, string type)
{
    this.fuel     = fuel ;
    this.cylinders = cylinders ;
    this.hp       = hp ;
    this.cc       = cc ;
    this.type     = type ;
}

public void displayEngine ()
{
    S.O.println ("Fuel :" + fuel) ;
    S.O.println ("Cylinders :" + cylinders) ;
    S.O.println ("HP :" + hp) ;
    S.O.println ("cc :" + cc) ;
    S.O.println ("Type :" + type) ;
}
}
```

* Blue print of Car :-

```
package encapsulation;
public class Car
{
    String brand ;
    string name ;
    string colour ;
    double price ;
    string type ;
    int airbag ;
    int seat ;
    string gear ;
```

```
Engine e1 = new Engine ("Petrol", 4, 200, 1000,
                        "Sport") ;

public Car () { }

public Car (String brand, string name, string colour,
            double price, string type, int airbag,
            int seat, string gear)
{
    this.brand = brand ;
    this.name = name ;
    this.colour = colour ;
    this.price = price ;
    this.type = type ;
    this.airbag = airbag ;
    this.seat = seat ;
    this.gear = gear ;
}

public void displayCar ()
{
    S.O.println ("Brand :" + brand) ;
    S.O.println ("Name :" + name) ;
    S.O.println ("Colour :" + colour) ;
    S.O.println ("Price :" + price) ;
    S.O.println ("Type :" + type) ;
    S.O.Println ("Airbag :" + airbag) ;
    S.O.println ("Seats :" + seat) ;
    S.O.println ("Gear :" + gear) ;
}
```

Siddhant Java

* Driver for Car object :-

```
package encapsulation
public class CarDriver
{
  public static void main (String [] args)
  {
    Car c1 = new Car ("TATA", "Nexon", "Black",
                      140000.0, "SUV", 5, 5, "Auto");

    c1.displayCar ();

    System.out.println ("==========");

    c1.e1.displayEngine ();

    System.out.println ("==========");

    Car c2 = new Car ("TATA", "Punch", "Blue",
                      150000.0, "SUV", 5, 5, "Auto");

    c2.displayCar ();

    System.out.println ("==========");

    c2.e1.displayEngine ();

    System.out.println ("==========");
  }
}
```

Monday
18/04/2022

* b) Lazy Binding (Lazy Instanciation) :-

- The instance (object) of dependent object is not created implicitly with the help of non-static initializers.
- Instead, a method is created and reference gets created, when that method is called for execution.
- This method is called as "Helper method".

* Example :- * Blue print for sim :-

```
package encapsulation ;
public class Sim
{
    String service ;
    String type ;
    String size ;
    long no. ;

  Sim () { }

  Sim ( String service, String type, String size, long no.)
  {
    this . service = service ;
    this . type = type ;
    this . size = size ;
    this . no = no ;
  }

  public void display sim ()
  {
```

Siddhant Java

```
        Syso ("Service Provider :" + service);
        Syso ("Type :" + type);
        Syso ("size :" + size);
        Syso ("Number :" + number);
        Syso ("==========");
    }
}
```

* Blue print of Mobile :-

```
package encapsulation;
public class Mobile
{
    string brand;
    double price;
    string colour;
    int ram;
    int hd;
    int battery;

    Sim s1 = null;

    public Mobile () { }

    public Mobile (string brand, double price, string
                   colour, int ram, int hd, int
                   battery)
    {
        this.brand = brand;
        this.price = price;
        this.colour = colour;
        this.ram = ram;
        this.hd = hd;
        this.battery = battery;
    }

    public void insertSim (String service, string type,
                           string size, long no.)
    {
        s1 = new Sim(service, type, size, no);

        System.out.pln("Sim inserted Successfully!!");
    }

    public void displayMobile ()
    {
        Syso ("Brand :" + brand);
        Syso ("Price :" + price);
        Syso ("Colour :" + colour);
        Syso ("RAM :" + ram);
        Syso ("HD :" + hd);
        Syso ("Battery :" + battery);
    }
}
```

* Driver for Mobile :-

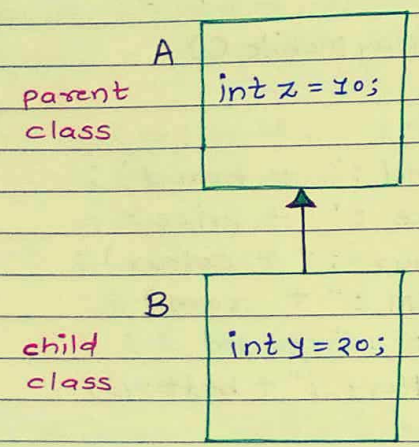
```
Package encapsulation
public class MobileDriver
{   public static void main (string [] args)
    {
    ① Mobile m1 = new Mobile ("Redmi", 14000.0, "Black", 4,
                              64, 4000)
      m1.displayMobile ()
    ② m1.insertSim ("Jio", "4G", "Nano", 123456781)
      m1.s1.displaySim ().
    }
}
```

Tuesday
19/04/22

* * 2] Is-A Relationship :-

- It is the association between two objects, which is similar to parent and child relation, is called as " Is-A relationship".

- In this, the child object aquires all properties and behaviours of parent object along with child object, as it can have its own properties and behaviour.



- declared = 1
- inheritant = 0
- Total = 1

- declared = 1
- inheritant = 1
- Total = 2

- Hence, with the help of Is-a relationship, we can achieve generalization and specialization.

* NOTE :-

1) Parent class is called "generalized".

2) Child class is called as "Specialized".

3) With the help of child class object reference, we can use all the members of parent class, as well as child class.

4) With the help of parent class reference of object, we can only use member of only parent class, not child class.

* Terminologies :-

1) Parent class is also called as "Super class" or "Base class".

2) Child class is also called as "Sub-class" or "Derived class".

3) With the help of "Inheritance", we can achieve Is-a relationship.

* 2] Inheritance :-

- The process of one class aquiring all properties and behaviours from another class, is known as "Inheritance".

- In java, we can achieve inheritance in 2 ways :-

  1) with "extends" keyword.
  2) with "implements" Keyword.

Siddhant Java

* Example :- (Using "extends" keyword) :-

* Blueprint of A (Parent)

```
package inheritance
public class A
{
    int z = 10;
}
```

* Blueprint of B (child)

```
package inheritance
public class B extends A
{
    int x = 20;
}
```

* Driver class :-

```
package inheritance
public class ABDriver
{
    public static void main(String [] args)
    {
        B b1 = new B();
        Sy.out.println(b1.x);    // 20
        Sy.out.println(b1.z);    // 10

        A a1 = new A();
        Sy.out.println(a1.z);    // 10
    // Sy.out.println(a1.x);    // CTE (Parent class (A) ∵ cant access child class (B) members)
    }
}
```

Wednesday
20/04/2022

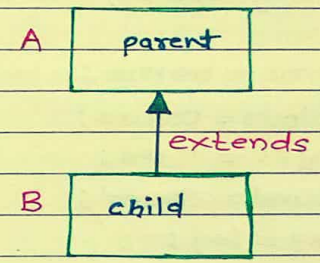
* * Levels of Inheritance :-

- In inheritance there are some levels as :-

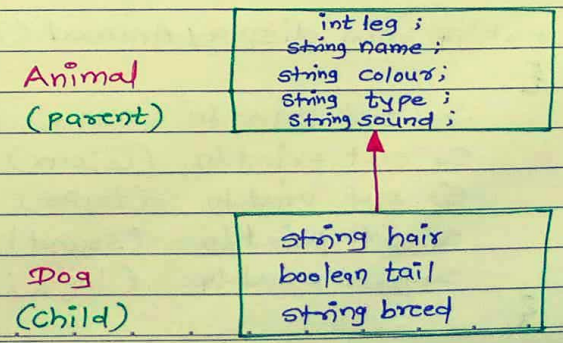
1> Single level inheritance.
2> Multi level inheritance.
3> Hierarchical inheritance.
4> Multiple inheritance.
5> Hybrid inheritance.

* 1> Single Level inheritance :-

- Inheritance achieved in only one level, is known as "single level inheritance".



* Example :-



Siddhant Java

Siddhant Java

* Blueprint of Animal :-

```
package inheritance ;
public class Animal
{
    int leg ;
    string name ;
    string colour ;
    string type ;
    string sound ;

  Animal () { }

  Animal ( string name, string colour, string type
           string sound, int leg)
  {
      this. name= name;
      this · colour= colour;
      this. type = type;
      this· sound= sound;
      this. leg= leg ;
  }

  public void display Animal ()
  {
      Sy. out. println (name) ;
      Sy. out. println (colour) ;
      Sy. out. println (type) ;
      Sy. out. println (sound) ;
      Sy. out. println (leg) ;
  }
}
```

* Blueprint of Dog :-

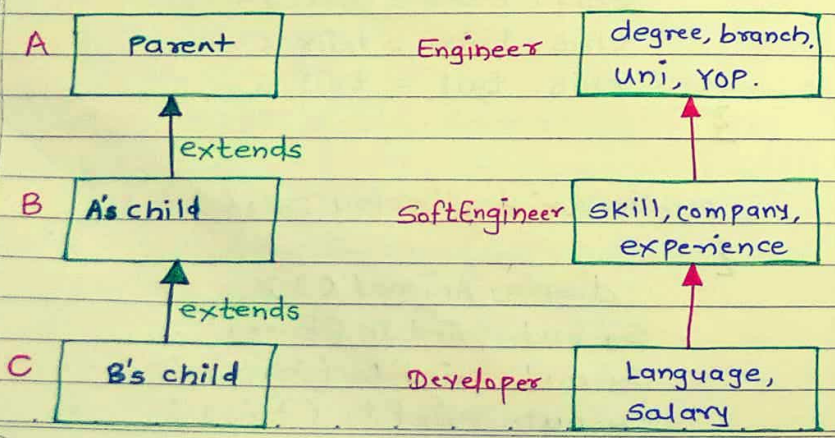
```
package inheritance ;
public class Dog extends Animal
{
    string hair ;
    boolean tail ;
    string breed ;

  Dog ( ) { }

  Dog (string name, string colour, string type,
       string sound, int leg, string breed,
       string hair, boolean tail)
  {
      this. name = name ;
      this. colour = colour ;
      this · type = type ;
      this. sound = sound ;
      this. leg = leg ;
      this. breed = breed ;
      this. hair = hair ;
      this. tail = tail ;
  }

  public void displayDog ( )
  {
      display Animal ( ) ;
      Sy. out. print ln (breed) ;
      Sy. out. print ln (hair) ;
      Sy. out. print ln (tail) ;
  }
}
```

* Driver class :-

```
package inheritance;
public class DogDriver
{
    public static void main (String [] args)
    {
        Dog d1 = new Dog ("Shery", "Black", "Omnivorous",
                          "Bark", 4, "German.shef",
                          "long coat", true) ;

        d1. displayDog () ;
        d1. displayAnimal () ;
    }
}
```

* 2) Multi Level inheritance :-

- Inheritance achieved in more than one level, is known as " multilevel inheritance".



(Example)

* Example :- * Blueprint of Engineer :-

```
package inheritance ;
public class Engineer
{
        String degree ;
        String branch ;
        String university ;
        int    yop ;

    public Engineer () { }

    Engineer (String degree, String branch, String
              university , int yop)
    {
        this. degree = degree ;
        this. branch = branch ;
        this. university = University ;
        this. yop = yop ;
    }

    public void displayEngineer ()
    {
        Sy.out. println (degree) ;
        Sy.out. println (branch) ;
        Sy.out. println (university) ;
        Sy.out. println (yop) ;
    }
}
```

Siddhant Java

* Blueprint of SoftEngineer :-

```
package inheritance;
public class SoftEngineer extends Engineer
{
        string skill ;
        string company ;
        double experience ;

    SoftEngineer () { }

    SoftEngineer (string degree, string branch, string
                  university, int yop, string skill,
                  string company, double experience)

    {
        this. degree = degree ;
        this. branch = branch ;
        this. university = university ;
        this. yop = yop ;
        this. skill = skill ;
        this. company = company ;
        this. experience = experience ;
    }

public void displaySoftEngineer ()
{
        displayEngineer () ;
        Sy.out. println (skill) ;
        Sy.out. println (company) ;
        Sy.out. println (experience) ;
    }
}
```

* Blueprint of Developer :-

```
package Inheritance ;
public class Developer extends SoftEngineer
{
        string langauge ;
        double salary ;

    Developer () { }

    Developer (String degree, string branch, string
               university, int yop, string skill,
               string company, double experience,
               string language, double salary)
    {
        this. degree = degree ;
        this. branch = branch ;
        this. university = university ;
        this. yop = yop ;
        this. skill = skill ;
        this. company = company ;
        this. experience = experience ;
        this. langauge = language ;
        this. salary = salary ;
    }

Public void displayDeveloper ()
{
        displaySoftEngineer () ;
        Sy.out.println (langauge) ;
        Sy.out.println (salary) ;
    }
}
```

Siddhant Java

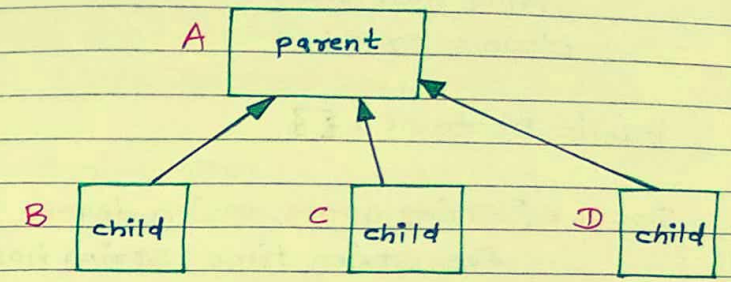
Siddhant Java

* Driver class :-

```
package inheritance;
public class DeveloperDriver
{
    public static void main (String [] args)
    {
        Developer d1 = new Developer ("BE", "CS", "Pune",
                                      2020, "Programming",
                                      "TCS", 2.5, "Java",
                                      30000);

        d1.displayDeveloper();
        d1.displaySoftEngineer();
        d1.displayEngineer();

        Engineer e1 = new Engineer ("BE", "Mech", "Pune",
                                    2021);

        e1.displayEngineer();
    // e1.displaySoftEngineer(); //CTE [Cant access child class objects]
    // e1.displayDeveloper(); //CTE
    }
}
```

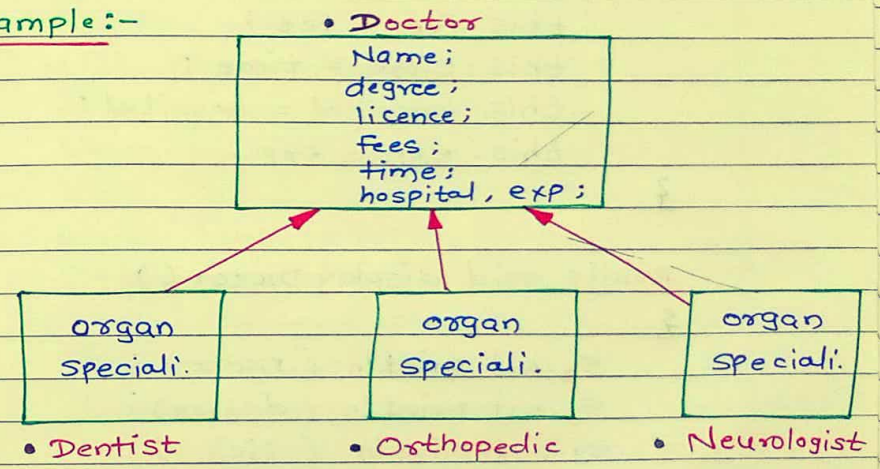
Thursday
21/04/2022

* * 3) Heirarchical Inheritance :-



- It is a design of one parent class having more than one child class of same level, is known as " Heirarchical inheritance".

* Example :-



* Blueprint for Doctor :-

```
package inheritance;
public class Doctor
{   String name;
    string degree;
    long lic;
    double fees;
```

```
    string time;
    string hospital;
    double exp;

Public Doctor() { }

Doctor(string name, string degree, long lic, double
        fee, string time, string hospital, double
        exp)
{
    this.name = name;
    this.degree = degree;
    this.lic = lic;
    this.fee = fee;
    this.time = time;
    this.hospital = hospital;
    this.exp = exp;
}

public void displayDoctor()
{
    Sy.out.println (name);
    Sy.out.println (degree);
    Sy.out.println (lic);
    Sy.out.println (fee);
    Sy.out.println (time);
    Sy.out.println (hospital);
    Sy.out.println (exp);
}
}
```

* Blueprint for Dentist :-

```
package inheritance;
public class Dentist extends Doctor
{
    String spcl;
    String organ;

  Dentist() { }

  Dentist (string name, string degree, long lic, double
            fee, string time, string hospital,
            double exp, string spcl, string organ)
  {
    this.name = name;
    this.degree = degree;
    this.lic = lic;
    this.fee = fee;
    this.time = time;
    this.hospital = hospital;
    this.exp = exp;
    this.spcl = spcl;
    this.organ = organ;
  }

  public void displayDentist()
  {
    displayDoctor();
    Sy.out.println (spcl);
    Sy.out.print ln (organ);
  }
```

Siddhant Java

* Blueprint for Orthopedic :-

```
package inheritance;
public class Orthopedic extends Doctor
{
    string spcl;
    string organ;

  Orthopedic () { }

  Orthopedic (string name, string degree, long lic, double
              fee, string time, string hospital, double
              exp, string spcl, string organ).
  {
    this.name = name;
    this.degree = degree;
    this. lic = lic;
    this. fee = fee;
    this. time = time;
    this. hospital = hospital;
    this. exp = exp;
    this. spcl = spcl;
    this. organ = organ;
  }

  public void displayOrthopedic()
  {
    displayDoctor();
    Sy.out.println(spcl);
    Sy.out.println(organ);
  }
}
```

* Blueprint for Neurologist :-

```
package inheritance;
public class Neurologist extends Doctor
{
    string spcl;
    string organ;

  Neurologist () { }

  Neurologist (string name, string degree, long lic,
               double fee, string time, string hospital,
               double exp, string spcl, string organ)
  {
    this.name = name;
    this.degree = degree;
    this. lic = lic;
    this. fee = fee;
    this. time = time;
    this. hospital = hospital;
    this. exp = exp;
    this. spcl = spcl;
    this. organ = organ;
  }

  public void displayNeurologist()
  {
    displayDoctor();
    Sy.out.println(spcl);
    Sy.out.println(organ);
  }
}
```

Siddhant Java

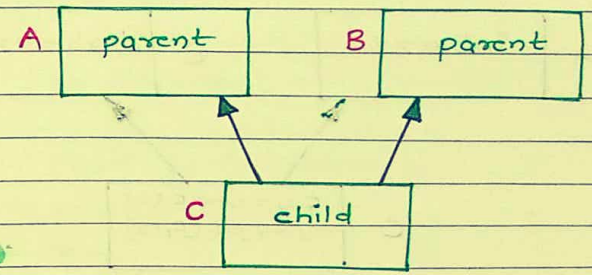
Friday
22/04/2022

* Driver class :-

```
package inheritance;
public class DoctorDriver
{
    public static void main (String [] args)
    {
        Doctor d1 = new Doctor ("XYZ", "MBBS", 3243721,
                                100, "8-5", "Sancheti",
                                1) ;

        d1.displayDoctor () ;
    // d1.display Dentist () ;  // CTE
    Sy.out.println ("=================") ;

        Dentist d2 = new Dentist ("ABC", "MBBS", 428251,
                                  500, "9-6", "PQR", 3,
                                  "MD", "Teeth") ;

        d2.displayDentist () ;
        d2.displayDoctor () ;
    }
}
```

* 4) Multiple Inheritance :-

- A child derived from more than one parents of same level, is known as " Multiple inheritance".
- i.e. A sub-class derived from more than one super class of same level is the " Multiple inheritance".



- Multiple inheritance has a limitation known as the "diamond problem".

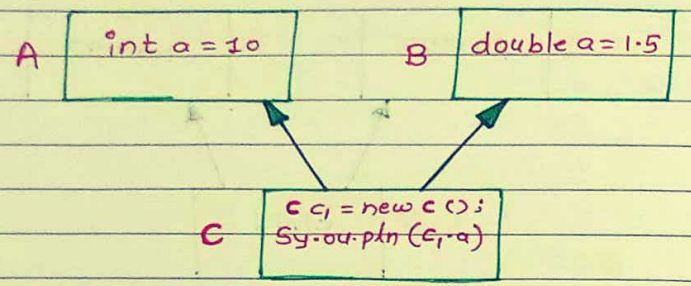
* Diamond Problem :-

- Diamond problem is an ambiguity or confusion that arrive while calling a method (variable) from sub-class (child) refference.

- e.g.:- Let us consider, class A and class B having common child C. In class A & B, we are having variables of same name as "a". Then there is confusion whether to inherit the variable or member from class A or class B.
  - This confusion is termed as the diamond problem.

Siddhant Java

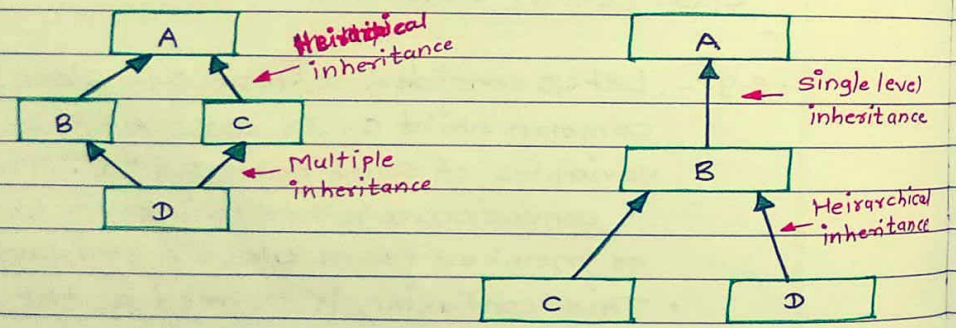
- Therefore, creaters of java instead of resolving this issue they restricted inheritance of class with only one class.
- Hence, in java a class can extend only one class.
- So that, it does not support multiple inheritance.



- Here, class C have two parents as A & B. For execution of variable 'a' from C, compiler gets confused which 'a' will take from A or B.

* 5) Hybrid Inheritance :-

- Combination of different types of inheritance, is known as "Hybrid inheritance".



- Here, classes A, B, C and D are inherited with each other, with various types of inheritance.

** Super :-

- It is the keyword which is used to reffer parent class member inside child class, when we have both parent class member and child class member of same.

- e.g. :-

* Blueprint of X :-

```
package inheritance ;
public class X
{
    int x = 20 ;

    public void abc()
    {
        Sy.out.println (x);
    }
}
```

* Blueprint of Y :-

```
package inheritance ;
class Y extends X
{
    int x = 10;
    public void abc()
    {
        Sy.out.println(x);              // 10
        Sy.out.println (super.x);       // 20
        super.abc();                    // 20 ... X-class
    }
}
```

Siddhant Java

* Driver class :-

```
package inheritance ;
Public class XYDriver
{
  public static void main (String [] args)
  {
     Y y1 = new Y();

     y1.abc();
  }
}
```

* Super call statement [super()] :-

- Super call statement is used to achieve the constructor chaining.
- Super call statement is used to call the super-class constructor during the object creation of sub-class.
- It helps to load all the non-static members of super class into the sub-class object.

* NOTE :-

1) Super call statement can only we use inside the constructor.

2) It should be the first instruction inside the constructor body.

3) A constructor cannot have both this call and super call statement.

- If a programmer fails to add the super call statement, implicitly the compiler will add no argument super call statement.
- If this call statement is used inside the constructor, the compiler will not add the super call statement.

* Purpose of super call statement :-

1) It helps to load non-static members of super-class into sub-class.

2) We can pass data from sub-class to super-class with the help of super call statement.

Siddhant Java

Saturday
23/04/22

* Derived Type Casting :- (Non-Primitive Type Casting)

- The process of converting one type of object reference to another type, is known as "Non-primitive type casting".

*** NOTE :-**

- We can acheive non-primitive type casting in cases as :-

1) The object must be in Is-A relationship.

2) If the both reference has common child.

- Non-primitive type casting i.e. derived type casting as of two types :-

  1) Upcasting
  2) Downcasting.

* * 1) Upcasting :-

- The process of converting child type object reference to parent type, is known as the "Upcasting".

- When the subclass reference is converted to superclass type, we can use only members of super class but not sub-class.

- Upcasting is implicitly done by the compiler.

- We do upcasting in order to achieve the generalization.

* * 2) Downcasting :-

- The process of converting super class reference type back to sub-class type, is known as "downcasting".

- Downcasting is not done implicitly by compiler. i.e. It has to be done explicitly by programer with the help of type cast operator.

*** Purpose of Downcasting :-**

- When sub-class reference is upcasted to the super class type, we cannot use the members of sub-class.

- Hence, in order to use members of sub-class we need to downcast the reference to the sub-class type.

Siddhant Java

Siddhant Java

* Example :-

* Blueprint of Doctor :- Ref. e.g. of Heirarchical Int.

* Blue print of Dentist :- Ref. e.g. of Heirarchical Int.

* Driver class :-

```
package inheritance ;
public class DoctorDriver
{
  public static void main (String [] args)
  {
    Doctor d2 = new Dentist ("ABC", "MBBS", 3242524,
                500, "8-6", "ABC smile care", 6,
                "MD", "Teeth"); // Upcasting

    d2.displayDoctor ();

    d2.displayDentist ();   // CTE

    Dentist d3 = (Dentist) d2 ; // Downcasting

    d3.displayDentist ();

    d3.displayDoctor ();
  }
}
```

* Class Cast Exception :-

- When we try to downcast the parent type reference directly to child type, we get class cast exception.

* Example :-

```
package inheritance ;
public class DoctorDriver
{
  public static void main (String [] args)
  {
    Doctor d1 = new Doctor ();

    Dentist d2 = (Dentist) d1 ; // Class Cast Exception (During Run Time)

    d2.displayDentist ();
    d2.displayDoctor ();
  }
}
```

Monday
25/04/2022

* 3) Polymorphism :-

- Poly means "many" and morph means "forms".
- The ability to have one name and many forms, is known as "polymorphism".
- It is of two types :-

1) Compile time polymorphism.
2) Run time polymorphism.

* 1) Compile Time Polymorphism :- (Compile Time Binding)

- If a member with one name shows many forms.
- Here, compiler decides which form to be executed at the compile time.
- This is known as "compile time polymorphism".

* Examples of compile time polymorphism :-

1) Method overloading.
2) Constructor overloading.
3) Method shadowing
4) Variable shadowing.
5) Operator overloading. ..... It is not possible in java.

* 2) Run Time Polymorphism :- (Run Time Binding)

- If a member has one name and exebits many forms.
- Here, compiler will not decide which form to be executed, instead the prefference is given (of execution) on the basis of type of object created.
- It is decided by the JVM, at the run time of a program.

* How to acheive Run time polymorphism ? :-

1) Derived type casting. (Upcasting)
2) Overriding.

* Method Overriding :-

- It is the process of giving new design to the super class method from sub-class, is known as "method overriding".
- Method overriding can be done only with non-static methods. (static methods cannot be overriden).

* To give new implementation or new design to previous designed methods, we use method overriding.

Siddhant Java

* Blueprint of Parent :-

```
package polymorphism ;
public class Parent
{
    public void m1 ()
    {
      Sy.out.println("old design") ;
    }
}
```

* Blueprint of child :-

```
package polymorphism ;
public class Child extends Parent
{
    public void m1 ()
    {
      Sy.out.println("New design") ;
    }
}
```

* Driver class :-

```
Package polymorphism;
public class ChildDriver {
  public static void main (String [] args)
  {
      Parent P1 = new Child ();      (method overriding)
      P1.m1() ;  // New design      child class method executed.
  }
}
```

[Here object is child type & ref. is Parent type. For P1.m1(), the object have more prefference. i.e. method in child executed]

Tuesday
26/4/2022

* Examples of Compile time Polymorphism :-

1) * Method Shadowing :-

- If a super class and sub class are having static method of same method signature, then it is called as "Method shadowing".
- Method shadowing is compile time polymorphism.
- Execution of the method defination does not depend on the type of object created, instead it depends on type of refference used to invoke the method.

* Blueprint of Parent :-

```
package polymorphism;
Public class Parent
{
    Public static void m1()
    {
      Sy.out.println ("Static method from Parent");
    }
}
```

* Blueprint of child :-

```
Package polymorphism;
Public class Child extends Parent
{
```

Siddhant Java

```
    public static void m1()
    {
        Sy.out.pln("From child's static method");
    }
}
```

* Driver class :-

```
package Polymorphism;
public class ChildDriver
{
    public static void main (string [] args)
    {
        Parent P1 = new Child();

        P1.m1(); // Static method From Parent
                 //..... Method shadowing
    }
}
```

* 2> Variable Shadowing :-

- If a super class and sub class variables of same name (static or non-static), the preffence is given to the type of reffence, but not to the type of object created.
- Variable shadowing is compile time polymorphism.

* Blueprint of Parent :-

```
package polymorphism;
public class Parent
{
    Static int a = 10;
    String s = "Parent";
}
```

* Blueprint of child :-

```
package polymorphism;
Public class Child extends Parent
{
    static int a = 20;
    string s = "Child";
}
```

* Driver class:-

```
package Polymorphism;
public class ChildDriver
{
    public static void main (string [] args)
    {

        Parent P1 = new child();

        Sy.out.print-ln (P1.a);   // 10      (Variable
        Sy.out.print-ln (P1.s);   // Parent   shadowing)
    }
```

Siddhant Java

* NOTE :-

- Overriding happens only when the visibility of overriding method (child) is greater or equal to the overridden method of parent class.

- i.e. ✓ 1) public child & public Parent

  ✓ 2) Public child & default Parent

  ✗ 3) private child & public Parent

  ✗ 4) default child & public Parent

* Example on Method Overriding :-

- Let us consider an ABC company's job application registration page.
- Initially the registration page contains the attributes or entries like student name, their qualification, branch, percentage, year of passout, etc.

- Now, there is new requirement in this registration page, that it also contains the year gap of student and if they applied for this company in passed 6 month or not?

- Hence, to give new design, we follow the method overriding as :-

* Blueprint of Older Registration page :-

```
package polymorphism;
public class Registration
{
        string name;
        string qualification;
        string branch;
        double percent;
        int yop;

  Registration () { }

  public Registration (String name, string qualification,
                       string branch, double percent,
                       int yop)
  {
        this.name = name;
        this.qualification = qualification;
        this.branch = branch;
        this.percent = percent;
        this.yop = yop;
  }

  public void displayPage ()
  {
        Sy.out.println ("Student name:"+ name);
        Sy.out.println ("Qualification:"+ qualification);
        Sy.out.println ("Branch:"+ branch;
        Sy.out.println ("Percentage:"+ percent;
        Sy.out.println ("Year of passout:"+ yop;
  }
}
```

Siddhant Java

* Blueprint of new Registration Page :-

```
package polymorphism;
public class Registration1 extends Registration
{
    int gap;
    boolean appliedBefore;

  Registration1() { }

  Registration (String name, String qualification, String branch,
                double percent, int yop, int gap, boolean
                appliedBefore)
  {
    super (name, qualification, branch, percent, yop);
    this.gap = gap;
    this.appliedBefore = appliedBefore;
  }

  Public void displayPage ()
  {
    Sy.out.pln ("Student name :" + name);
    Sy.out.pln ("Qualification :" + qualification);
    Sy.out.pln (" Branch :" + branch);
    Sy.out.pln (" Percent :" + pencent);
    Sy.out.pln (" Year of passout :" + yop);
    Sy.out.pln (" Year gaps :" + gap);
    Sy.out.pln ("Are you applied before in last
                 6 months :" + appliedBefore);
  }
}
```

* Driver class :-

```
package polymorphism;
public class RegistrationDriver
{
  public static void main (String [] args)
  {
    Sy.out.pln ("** Student Registration Page **");

    Registration R1 = new Registration1 ("Vishal",
                        "B.E.", "Mech", 73.85, 2019, 0,
                        False); // Upcasting

    R1.displayPage ();   // Method Overriding
  }
}

// Here, displayPage method of Registration1
   class (i.e. new) is executed.
```

Siddhant Java

Wednesday
27/04/2022

*4] Abstraction :-

- Feature without implementation details.

- It is a design process, which help the service specifier to provide the most essential features of a project, without providing the implementation detail, is known as the "abstraction".

- Also, it is a design of hiding the implementation details from service specifier or service user, is known as "abstraction".

- i.e. The process of hiding the implementation details and showing only the features, is called "abstraction".

* How to achieve Abstraction :-

1) With the help of classes.

2) With the help of interfaces.

- Methods are classified into :-

  1) Abstract method.
  2) Concrete methods.

*1) Concrete Methods :-

- Methods which has implementation details (body) is called as "concrete methods".

*2) Abstract Methods :-

- Method without any implementation details (body) and are prefixed with "abstract" key-word, is called as "Abstract methods".

  - Abstract methods used by Service Specifier.
  - Concrete methods used by Service Provider.

* Syntax :-

Access Modifier	Abstract	Return type	Method name (F.A.) ;

* • NOTE :-

- If a class is having atleast one abstract method either declared or inherited, it is mandatory to make the class "abstract", else we get compile time error.

- Classes are classified as :-

  1) Abstract class.
  2) Concrete class.

Siddhant Java

Sundaram
Sundaram

Siddhant Java

* 1) Concrete class :-

- A class which is not prefixed with "abstract" modifier, is known as " concrete class".
- We can create objects only inside the concrete class, but not inside abstract class.

* 2) Abstract class :-

- A class which is prefixed with abstract modifier, is called as " Abstract class".
- It is mandatory to make a class abstract, if the class is having atleast one abstract method, either declared or inherited, but not overriden.
- We cannot create instance (object) for an abstract class.

* Example :-

- Let us consider an calculator software or app, which is to be developed.
- For making / developing calculator two providers as windows and Linux develop there own design.
- Here, calculator is abstracted class, whereas, linux and windows are concreted / completed class.

Sundaram

* Blueprint of Calculator :- // Service specifier.

```
package abstraction ;
public abstract class Calculator
{
    abstract public void add (int a, int b) ;

    abstract public void sub (int a, int b) ;

    abstract public void div (double d, double c);

    abstract public void mult (int s, int r) ;
}
```

* Blueprint of Windows :- // Service Provider

```
package abstraction ;
public class Windows extends Calculator
{
    public void add (int a, int b)
    {
        int res = a+b ;
        Sy.out.pln (res) ;
    }

    public void sub (int a, int b)
    {
        int res = a-b ;
        Sy.out.pln (res) ;
    }
```

Sundaram

```
    public void div (double s, double r)
    {
      double res = s/r ;
        Sy.out.pln(res);
    }

    public void mult (int a, int b)
    {
        int res = a*b ;
        Sy.out.print ln(res);
    }
}
```

* Blueprint of Linux :- // Service Provider

```
package abstraction
public class Linux extends Calculator
{
    public void add (int a, int b)
    {
      Sy.out.pln(a+b) ;
    }

    public void sub (int p, int q)
    {
      Sy.out.pln (a-b);
    }

    public void div (double d , double c)
    {
      Sy.out.pln (d/c);
    }
```

```
    public void mult (int a, int b)
    {
      Sy.out.pln(a*b);
    }
}
```

* Driver class :-

```
package abstraction ;
public class CalDriver
{
  public static void main (String [] args)
  {

    Windows w1= new Windows ();
    w1.add (10,20) ;    // 30     } widow's method
    w1.div (20,10) ;    // 2.0    } executed

    Linux l1 = new Linux ();
    l1.sub (20,10) ;    // 10     } Linux Method
    l1.add (30,40) ;    // 70     } executed

    Calculator c1= new Window () ; //Upcasting
     c1.add (15,20) ;   // 35     --> window method
     c1.mult (10,2) ;   // 20     --> executed

    Calculator c2 = new Linux () ; //Upcasting
     c2.sub (5,4) ;     // 1      --> Linux method
     c2.div (10,2) ;    // 5.0    --> executed.
  }
}
```

Siddhant Java