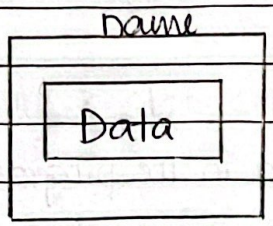
16/8/23

Variable - Variable is a named block of memory which is created in primary area / Execution area used to store the ~~~~ data.



stored in Execution area / Primary area

Datatype - It specify which type of container should be created

Ex : int → Specify Integer Value. ♀ → age = 23

int age;

↘ stored in execution area

Variable Declaration - There are two steps to follow :

1. Declare a Variable (Variable Declaration)
2. Initialize a Variable (Variable Initialization)

1. Variable Declaration -

Syntax - datatype Variablename/Identifier ;

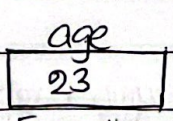
Ex : Integer data

↳ 3, 2, 0, -1, 2 etc

Age : 23 years → Variable → int age ;

↓

Integer

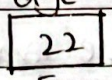


Execution area.

2. Variable Initialization -

Syntax - Variable name = Value;

Ex : int age;
age = 22;



Execution area.

For declaring

Variable declaration and Initialization -

Syntax - datatype Variablename = Value;

Ex : int age = 22;

To create more than one Variable at a time -
declare
- Datatype should be same

Syntax - datatype identifier1, Identifier2 ..... ;

Ex : int a; b;
a = 10;
b = 20;

Characterstics of Variables :

1. Variable has name, by using it we can store the data and we can fetch the data.

2. We can store only one value inside a Variable.

Ex : age [22] ✓ age [22,23] ✗

3. The Variables are temporary.

17/8/23

* How to create a Variable?

By using datatype and Variable name

```
datatype Variablename;
```

* Program for Variable printing

①

```
class P1
{
  Public static void main (String[] args)
  {
    int age ;
    age = 23;
   System.out.Println (age);
  }
}
```

Advantage of a Variable

- Variable can be re-initialized (We can enter the new data in the same container then old data will be erased and new data will be printed)

```
int a = 23;
a = 24;  ⟶ here old data 23 will be erased and new data 24 will be printed.
```

- We can update the Variable Value

```
int a = 23;
a = 24; → Here variable value is updated to 24.
```

Programs for Re-initialization and updating

①

```
class Reinitialize
{
  Public static void main (String[] args)
  {
     int age = 23;
        System.out.println (age);
           age = 24;
        System.out.println (age);
  }
}
```

Output – 23
24

②

```
class Update
{
  Public static void main (String[] args)
  {
    int run = 64;
    System.out.Println (run);
    run = run+1
    System.out.Println (run);
  }
}
```

Output 64
65

Datatypes

It specifies which type of container (variable) should be created.
It specifies which type (or) size (or) which data should the container store.

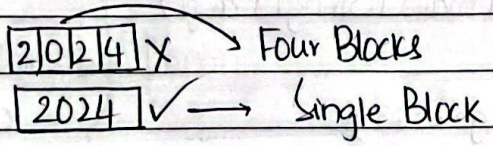
Types of Datatypes

1) Primitive data-type
2) Non-primitive datatype.

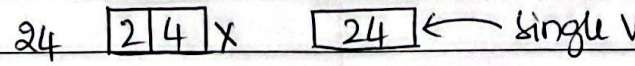
1) **Primitive datatype** - Used to create primitive variable to store primitive data.

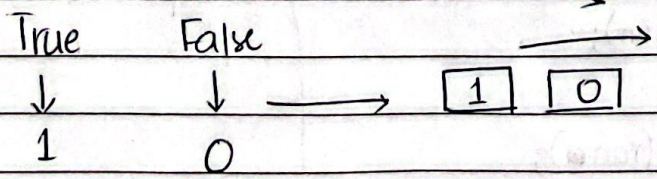
Primitive data → Single value data (Data stored in single block of memory)

Ex: 2024



Number Literal
Character Literal
Boolean Literal
} Primitive Data or value.





There are 8 types of Primitive Datatypes :

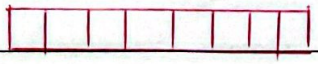
① byte
② short
③ int
④ long
⑤ float
⑥ double
⑦ char
⑧ boolean

Primitive Value	Primitive (8) Datatype	Default Values	Size
Number → Integer	byte, short, int, long	byte – 0 short – 0 int – 0 long – 0l/L	1 byte 2 byte 4 byte 8 byte
Number → Decimal	float, double	float – 0.0f/F double – 0.0d/D (optional)	4 byte 8 byte
Character →	char	char – /u0000 (unicode)	2 byte
Boolean →	boolean	false	1 bit

Note - The Number datatype in increasing order by size (capacity)

byte < Short < int < long < float < double
(1 byte < 2 Byte < 4 byte < 8 byte < 4 byte < 8 byte)

* 1 byte = 8 bits.

byte → 1 byte = 8 bits 

Boolean → 1 Bit → 0/1 0 – positive value
1 – negative value.

Integer { Byte, short, int, long } — Capacity → Based on the capacity of datatype we are going to use the primitive

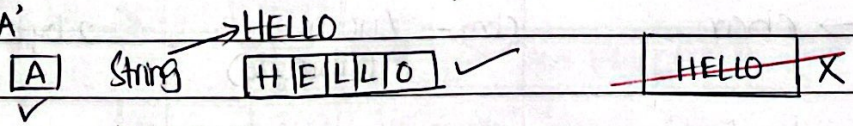
byte → 1 byte → 8 bit
short → 2 byte → 16 bits
int → 4 byte → 32 bits
long → 8 byte → 64 bits

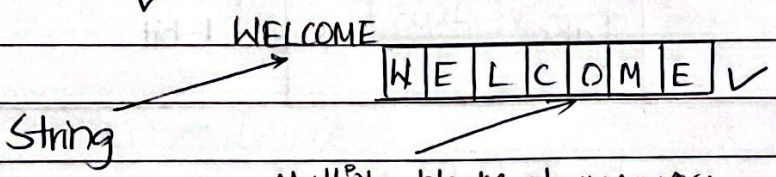
18/8/23

Non-Primitive Datatype - Used to create non-primitive variable to store non-primitive data.

Non-primitive Data - Multi value data stored in multiple blocks of memory.

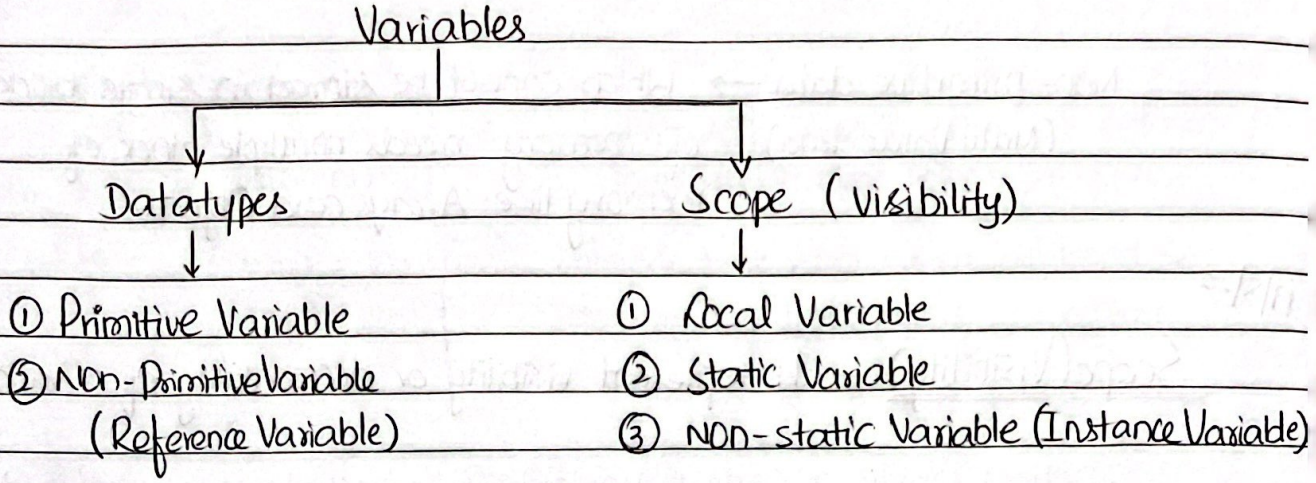
Ex: 'A'





* String — Non-primitive Datatype.

Variables are classified by based on

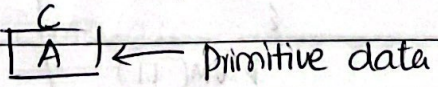


Primitive Variable - A variable created to store primitive data.

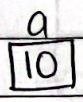
Primitive data → Single value data → Stored in single block of memory.

Number Literal
Boolean Literal
character Literal
} → Primitive Data or Value → Single Value data

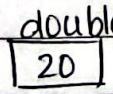
Ex :1) char c = 'A';



2) int a = 10;



3) double money = 20;



Non-primitive Variable — The Variable created to store non-primitive data.

Non-primitive data → Which cannot be stored in single block (Multi Value data) of memory, needs multiple block of memory like Arrays and Objects.

21/8/23

Scope (Visibility) — It represent visibility or accessability of a Variable.

(Defines where a certain variable is accessible in a program)

Local Variable - A Variable declare in a method block or Initializers block or any block except class block is known as a Local Variable.

Rules of Local Variable :

1. We can not use the local variable without assigning the data.

Ex:

```
class Demo
{
  PSVM( [] )
  {
    int a;
  }
}
```

✗ error - missing method body

```
class Demo
{
  PSVM( [] )
  {
    int a = 10;
  }
}
```

✓

2. We can use the variable inside the block where it is declared, we cannot use it out of the block.

Ex:

```
Class Demo {
  Psvm( [] ) {
    int a = 10;
  }
  S.o.p (a);
}
```

✗ Compile time error — <identifier> expepted

Rule 3 - We cannot declare two local Variable with the same name inside the same scope (block)

Ex:
```
Class A
{
  PSVM(){
  int a = 10;
  int a = 20;
  }
}
```

Error - Variable is already defined in instance initializer of class Demo3 int a=20;

X compile time error as we declared same variable two times in the same block.

Program ①

```
Class Demo4
{
  Public static void main (String[] args)
  {
    int a;
    a = 10;
    System.out.println(a);
  }
  System.out.println(a);
}
```

↓

Error - <identifier> expected

- Should not give print statement in class block, we should give the print statement only where int the variable is declared.

Program ②

```
class Demo5
{
  public static void main(String[] args)
  {
    int a = 10;
    int b = 20;
    System.out.println(a);
    System.out.println(b);
  }
}
```

compile - Successful
Execution - Successful.

o/p - 10
20

Program ③

```
Class Demo6
{
  public static void main(String[] args)
  {
    int a;
    System.out.println(a);
  }
}
```

Error - Variable a might not have been initialized

→ Didn't initialized 'a'. should initialize variable a. then print 'a'

Program ④

```
Class Demo7
{
  public static void main(String[] args)
  {
    int b = 10;
    int c = 20;
    System.out.println(b);
  }
  System.out.println(c);
}
```

Error - <identifier> expected

System.out.println(c);

Use the print statement between in the block where the variable is declared.

21/8/23

Operators : These are pre-defined symbol used to perform some task. based on operands. by returning some value.

Ex : 10 + 20 = 30
operator
operand

Operand - It is a data on which operator is performing some action.

Ex :

```
Class Demo
{
  Public static void main(String[] args)
  {
    10+20;   erro – not a statement
  }
}
```

This is not a statement, it is an expression.

Expression : It is a combination of operator and Operand.

data + data
operand
operator

Ex :

```
Class Program1
{
  Public static void main(String[] args)
  {
    int res = 10+20;
    System.out.println(res);
    System.out.println(10+20);
  }
}
```

O/p – 30
30

Operators will return some value

(int) (int)
10 + 20 = 30
- 30 ← value
- + ← operator
- 10, 20 ← type of data

(int) (double)
10 + 20.5 = 30.5

depends on the (2 byte) capacity it will return the value.

Characterstics of an Operator :

1. Operators are going to return something
   1. Type of Value (or) data
   2. Value

2. Precedence (Priority) :

It is a priority given to every operator and this precedence is used when the expression contains more than one operator.

* Expression having more than one operator is called as Complex Expression.

Ex: 1+2x3 = 7 ← Here 'x' is having the highest priority according to BODMAS method

Complex Expression

Ex: 10x3+20/2 = 40 ← Complex Expression.

Highest precedence (priority) in java program. (BODMAS)

10x3+10 → 30+10 = 40

3. Associativity :→ Direction of order of Execution.

→ Two possibilities in the direction

① Right to left

② Left to Right.

Note : Whenever an expression having the same Operator then Associativity will take place.

Ex: 10 + 20 + 20 (Same Operator in this complex Expression)

= 50

Complex Expression

Types of Operators

- Operands
  - Unary Operator
  - Binary Operator
  - Ternary Operator
- Task
  - Arithmetic Operator
  - Relational Operator
  - Logical Operator
  - Assignment Operator
  - bitwise Operator
  - ⋮

Ex: 1 + 2 + 3

Int 6 (left to Right / Right to left)

(Associativity)

1 + 2 × 3

Int 7

(precedence)

Unary Operator : It can perform task with only one data a time.
(one)

Ex : int a = 10; → output 10
a++; → a = a+1
↘ o/p = 10+1 = 11

Here only data ie - 10.

Binary Operator : It can perform task with two data at a time.

Ex : 10+20, 10x20, 30-20 (Here 10, 20 are the data)

Ternary Operator : It can perform task with 3 or more data at a time.

Ex : 10+20x3 (Here 10, 20 and 3 are the data)

22/8/23

Arithmetic Operator - In Java we have some pre-defined Symbol. to perform arithmetic operations. Arithmetic operator also known as binary operator as we need 2 datas to perform an arithmetic operation.

1. (+) → Addition → It is an arithmatic operator
→ It is also known as Binary operator

This operator have two different behaviour

① Addition
② Concatination (Merging (or) Combine two things)

- This '+' operator is polymorphic in nature

Polymorphic (many forms)

↓

'+' - Some times (+) operator acts as addition

'+' - Sometimes its acts as concatination

Ex : Add → 10+20 = 30

concatination → 10 + 'HI' → 10HI

integer → String

For concatination any one of the operands should be a 'String'

Ex : 5+3+"HI" → 8+"HI" = 8HI

Here first addition is the priority

Ex : 1

```
class op4
{
  Public static Void main(String[] args)
  {
   int a = 10;
   int b = 20;
   int c = a+b;
   System.out.println(c);
  }
}
```

→ output is 30, here addition is the arithmatic operation took place.

Ex:4 :

```
class {
public static void main (String[] args)
{
int a = 10;
boolean b = true;
System.out.println(a+b);
}
}
```

→ o/p - bad operand types for binary operator '+' because we cannot concatenate integer value and boolean value

Ex:5

```
class Demo5 {
Public static void main (String[] args)
{
String a = "10";
boolean b = true;
System.out.println(a+b);
}
}
```

o/p → 10true (compile and execution successful)

Ex:6

```
class Demo6 {
public static void main(String[] args)
{
int a = 10;
int b = 20;
String s = "HI";
System.out.println(a+b+s);
}
}
```

→ o/p → 30HI