JAVA NOTES 

Java is a **High-Level Language**(easy for humans to read and write).

***HLL*:**High-Level Language , Human understandable code.

***LLL*:**Low-level language(machine language), contains only 1's and 0's and is directly understood by a computer.

***JDK* :**JAVA DEVELOPMENT KIT

- It is a collection of software tools, libraries, java compiler JRE etc.
- It enables developers to write, compile, and run Java programs.

Compiler

- It translates the entire source code of HLL into LLL or an intermediate code(closer to machine code) in a single step.
- It scans syntax errors.

Interpreter

- The interpreter translates HLL line by line.

WHOLE PROCESS OF RUNNING A JAVA CODE

Java compiler translates source code into byte code.The JVM (DTL) loads and executes the byte code. Optionally, some JVMs may choose to interpret the byte code directly for certain use cases.This combination

of compilation and interpretation allows Java programs to be both platform-independent (byte code run on any machine).

|| DAY 1 ||

JDK :https://www.oracle.com/java/technologies/downloads/#jdk21-windows

IDE :https://www.jetbrains.com/idea/download/?section=windows

Boiler-plate || Default Code

```
public class Demo1 {
public static void main(String[] args) {

    }
}
```

Class- Collection of methods and variables. It is concept of OOPs(DTL).

Method - A method is a block of code which performs certain operations and returns output(DTL).

Entry Point

The **main** method is the entry point for executing a Java application.

When you run a Java program, The JVM or java compiler looks for a public static void main(String args[]) method in that class, and the signature of the main method must be in a specified format for the JVM to recognize it as its entry point.

If we update the method's signature, the program will throw the error NoSuchMethodError:main and terminate.

|| DAY 2 ||

Just like we have some rules that we follow to speak English(the grammar), we have some rules to follow while writing a Java program. The set of these rules is called Syntax.

Variables

A variable is a container that holds data. This value can be changed during the execution of the program.

Before use, you need to declare and define it.

1. Variable Declaration:

```
int age;
String name; //int and string is data types
```

2. Variable Initialization:

```
age = 69;
name = "the boys";
```

3. Combined Declaration and Initialization:

```
int age = 69;
String name = "the boys";
```

4. Final Variables (Constants):

final int a = 7;[DTL]

// DAY 3 //

Identifiers- Identifiers are used to uniquely identify the variables.

Identifier is a name given to a variable, class, method, package, or other program elements.

Rules for Identifiers in Java:

1. Start: Must start with an alphabet or _ or $ NOT with a digit.

2. End: Can end with an alphabet or _ or $ or numeric digit.

3. No Reserved Words : You cannot use Java's reserved words (also known as keywords) as identifiers.

4. No Special Symbols: Identifiers cannot contain special symbols like @, #, %, etc. except for underscores (_) and dollar signs ($).

5. No Space : Spaces are not allowed.

6. Length – No Limit

Java is CASE SENSITIVE : Shery and shery is different for java

camelCase	Used to name methods and variable eg- main(), lastName.
PascalCase	Used to name classes, interfaces(yet to come)
snake_case	Can be used in place of camel case(not recomm)
kebab-case	Unsupported in java

Keyword and word : Keywords are reserved(built-in) words which has specific meanings and cannot be used as identifiers.e.g.public, class, static, if, else, while etc.

|| DAY 4 ||

Literal or Constant:

Any constant value which can be assigned to the variable.

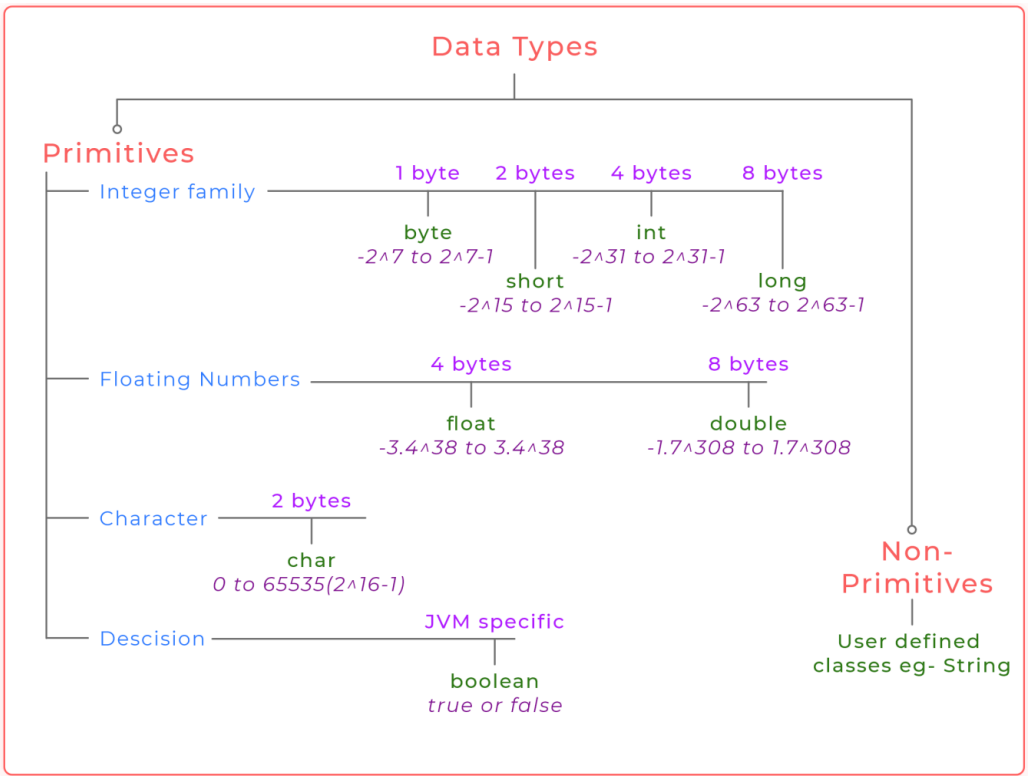
DATA TYPES

Data types are used to classify and define the type of data that a variable can hold.

There are 2 types of Data Types:

1.Primitive Data types: pre-defined, fixed size.

2.Non-Primitive Data types: Customize and no fixed size.



Default Values

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false

the compiler never assigns a default value to an uninitialized local variable(DTL).

+ operator between two char values

- It performs addition between their Unicode code points.
- For example :

```
char a = 'a';
char b = 'b';
System.out.println(a+b);//97+98=195
```

Output : 195

// DAY 5 //

Scanner

To take input from users we use Scanner class.

Scanner class is a built-in class in the java.util package(DTL). Before using the Scanner class you have to import the Scanner class using the import statement as shown below:

To use the **Scanner** class, you need to create an object of it, and then you can use that object to interact with the input data.

```
import java.util.Scanner;

Scanner sc = new Scanner(System.in);//object
int n = sc.nextInt();
```

The nextInt() method parses the token from the input and returns the integer value.

Use methods to read respective data

nextByte(), nextShort(), nextInt(),nextLong(),nextFloat(), nextDouble(), nextBoolean()

Reading String Data

nextLine() - Reads the whole line

next() - Reads the first word

Reading char data–next().charAt(0)

Problem with nextLine() method:

Ifwe try to read String after reading in an Integer, Double or Float etc.

Java does not give us a chance to input anything for the name variable.

When the method input.nextLine() is called Scanner object will wait for us, to hit enter and the enter key is a character("\n").

Let's understand this with the example

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

System.out.print("Enter an integer: ");
int age = scanner.nextInt();

System.out.print("Enter a string: ");
String name = scanner.nextLine();

System.out.println(name + " age is = " + age);
```

Console :

Enter an integer: 69 // 69\n(enter)

Enter a string: age is = 69

1. We first prompt the user to enter an integer age using **nextInt()**.
2. After reading the integer, we immediately hit enter and enter is also a character represented by “\n” – 69\n
3. The int value 69 is assigned in age but not the \n still left in the memory or buffer.
4. In next line when we Call **nextLine()** to consume the name it first check in buffer is there any thing as we have \n in buffer it take \n (for nextLine() method \n is the stopping point it will consider we stop giving input and return) and skip the line.

Solution :

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

System.out.print("Enter an integer: ");
int age = scanner.nextInt();

// Consume the newline character left in the input buffer
scanner.nextLine();

System.out.print("Enter a string: ");
String name = scanner.nextLine();

System.out.println(name + " age is = " + age);
```

1. After taking an integer input we Call nextLine() to consume the name character left in the input buffer.In next line when we Call nextLine() to consume the name character left in the input buffer.

2. Then, we prompt the user to enter a string using nextLine().

\ is a special symbol

\n (next Line), \b (backspace), \t (tab), \" (double quote), \' (single quote), and \\ (backslash).

// DAY 6 //

Operators

Operators can be easily defined as characters that represent an operation. These symbols perform different operations on several variables and values.

Example : 5 + 6 = 11. Here, 5 and 6 are the **operands**, and **+** is called the **operator**.

Categories of Operators

Unary operators : perform an action with a **single operand**.

Binary operators: perform actions with two **operands.**

Types of Operator

1. Arithmetic Operator :

Arithmetic operator can be divided into two categories -

- **Binary Operators :** +, -, *, /(int/int will always yield int) , % (Return remainder after dividing two numbers & with int (works perfectly) but with float (produces ambiguity)).

Special powers of / & % by powers of 10

/ : to reduce the number by 1 digit

% : to get last digit(s) of number

- **Unary Operators :**

  Increment Operator **(++)** : Increase the value by 1.
  Decrement Operator**(- -)** : Decrease the value by 1.

  +, - and **!**(DTL) is also a unary operator(-5, 5 (is same as +5)).

RULES for Increment and Decrement :

- Cannot applied to constant
  Example : int c = ++10; // compile-time error

- Nesting of both operators is not allowed
  Example :int a = 10;
  int b = ++(++a); // compile-time error [++11]
- They are not operated over final variables
  Example : final int a = 10;
  int b = ++a; // compile-time error
- Increment and Decrement Operators can not be applied to booleans.
  Example : boolean a= false;
  a++;// compile-time error

Quiz On Increment And Decrement Operators :
https://sheryians.com/whizz/test/651ce22ebf489c724c896167

2. Relational Operators : Used to check the relations between two operands. They return a boolean value (true or false) by comparing the two operands.

Greater Than (>) ,Less Than (<), <=, >=, ==, !=

- **Equal To (==)** - Checks if two operands are equal
- **Not Equal To (!=)** - Checks if two operands are not equal
- **Greater Than or Equal To (>=)** - Checks if one operand is either greater than or equal to the other.
- **Less Than or Equal To (<=)** - Checks if one operand is either less than or equal to the other.

3. Logical operators : Combine multiple conditional statements. There are three types of logical operators in Java: **AND(&&), OR (||) and NOT(!)** operators.

- **Logical AND Operator(&&)**

  Returns **true** when both conditions under evaluation are true, otherwise it returns **false**.
  e.g : if(a>b && a<c) System.out.println("Maximum: " + a);

- **Logical OR Operator(||)**

  Returns **true** if any one of the given conditions is true, otherwise it returns **false**. It returns **false** if and only if **both conditions** under evaluation are false.

  e.g : if(a>b || a<c) System.out.println("Max: " + a);

- **Logical Not Operator(!)**

  It accepts a single value as an input and returns the inverse of the same. This is a unary operator unlike the AND and OR operators.
  e.g : if(!(a<b)) System.out.println("Max: " + a);

4. ShortHand operators

The assignment operator can be combined with other operators to build a shorter version of the statement. **+=, -=, *=, /=, %=**

Example : a = a+5, we can write a **+=** 5.

do not use **=+ & -=** [(=) followed by a unary plus (+)]

|| DAY 7 ||

Package

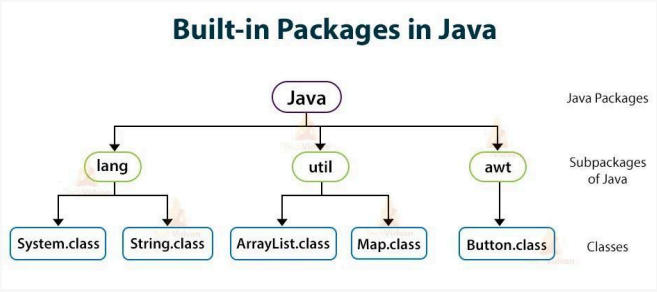
A Java package is a collection of similar types of sub-packages, interfaces, and classes. They help you manage and group related classes, interfaces, and sub-packages to avoid naming conflicts and create a more organized and maintainable codebase.

Example:

Directories or folders on your computer's file system(manage files).

In Java, there are two types of packages: built-in packages and user-defined packages.

in-built packages : They are available in Java, including util, lang, awt etc. We can import all members of a package using package name.* statement



java.lang package is a special package that is automatically **imported by default** in every Java class.

Commonly used classes and types from the java.lang package include: String, System, Math etc.

User-defined packages : User-defined packages are those that the users define. Inside a package, you can have Java files like classes, interfaces, and a package as well (called a sub-package).

Math Class

java.lang.Math class is a built-in class. It provides mathematical functions and constants for mathematical operations.

Commonly used methods and constants:

Math.abs(a)	Returns the absolute value of a value.
Math.sqrt(a)	Returns the sqrt root of a double value.

Math.ceil(a)	Returns the closest value that is greater than or equal to the argument
Math.max(a, b)	Returns the greater of two values.
Math.min(a, b)	Returns the smaller of two values.
Math.pow(a, b)	Returns a raised to the power of b.
Math.random()	Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

Math.PI, Math.floor(a)(closest value that is greater than or equal to the argument) etc.

// DAY 8 //

CONTROL-FLOW STATEMENTS

Control Flow statements in programming control the order of execution of statements within a program. They allow you to make decisions, repeat actions, and control the flow of your code based on conditions.

Types of control flow statements

1. Conditional or Decision Making statements (if-else and switch)
2. Looping statements (for, while, and do-while)

3. Branching statements (break and continue)

1.Conditional statements **If-else** :

The **if-else** statement allows you to execute a block of code conditionally.

If the condition inside the **if** statement is true, the code inside the **if** block is executed;

otherwise, the code inside the **else** block is executed.

Syntax of if-else :

```
int age = 30;
if(age >18) {
    System.out.println("Adult");//executes if condition is true
}else {
    System.out.println("Abhi chote ho"); //condition false
}
```

If-Else-If Ladder :

"If-Else-If" ladder consists of an if statement followed by multiple else-if statements. It is used to evaluate a condition using multiple statements. The chain of if statements are executed from the top-down.

It checks each if condition, and as soon as one of the if condition yields true, it executes the statement inside that if block and **skip the rest of the ladder**. If none of the conditions evaluates to be true, then the program executes the statement of the final **else** block.

For example :

```
int number = 10;
if (number % 2 ==  0) {
```

```
    System.out.println("Number is even.");
}
else if (number % 2 != 0) {
    System.out.println("Number is odd.");
}
else {
    System.out.println("Invalid input.");
}
```

Output : Number is even

If Ladder :

"If" ladder consists of an multiple if statements. It is used to evaluate a condition using multiple statements. The chain of if statements are executed from the top-down.

The program checks each if condition, and as soon as one of the if condition yields true, it executes the statement inside that if block and **still check further conditions**. If none of the conditions evaluates to be true, then the program executes the statement of the final **else** block.

```
int number = 10;
if (number >0) {
    System.out.println("Number is positive.");
}
if (number <20) {
    System.out.println("Number is less than 20.");
}
if (number % 2 == 0) {
    System.out.println("Number is even.");
}
```

Output : Number is positive

Number is less than 20.";

Number is even.

// DAY 11 //

Ternary Operator

The ternary operator, also known as the conditional operator, is a shorthand way of writing an **if-else** statement with a single expression.

Syntax : **condition ? expression1 : expression2**

- If the **condition** is true, the expression before the **:** (i.e., **expression1**) is evaluated and returned.
- If the **condition** is false, the expression after the **:** (i.e., **expression2**) is evaluated and returned.

Example:

```
int num = ;
String result = (num % 2 == 0) ? "Even" :"Odd";
System.out.println("The number is " + result);
```

Output : Even

Type Conversion

Type casting in Java is the process of converting one data type to another. It can be done automatically or manually.

Type Casting in Java is mainly of two types.

1. Widening or Implicit Type Casting
2. Narrow or Explicit Type Casting

1.Widening or Implicit Conversion:

- Java allows automatic type conversion when a smaller data type is promoted to a larger data type..
- It is secure since there is no possibility of data loss.
- **Both the data types must be compatible with each other :** converting a string to an integer is not possible as the string may contain alphabets that cannot be converted to digits.

Order :byte->short->int->long->float->double

char->int

Example :

```
int intValue = 42;
double doubleValue = intValue; // Implicit conversion
```

2.Explicit or Narrowing Conversion:

- Sometimes, we need to convert a larger data type to a smaller one explicitly and it requires a cast operator.
- **Narrowing Type Casting in Java is not secure** as loss of data can occur due to shorter range of supported values in lower data type.

Example :

```
double doubleValue = 42.0;
int intValue = (int) doubleValue; // Explicit
conversion (casting)
```

Note : *Shorthand operators do implicit conversion.*

Byte b = 1;

b=b+2; // error , 2 is int(all non-float by default int) so can't store in byte

b += 2; // works perfectly as += did implicit conversion