UNIT-1

Program Structure in Java: Introduction, Writing Simple Java Programs, Elements or Tokens in Java Programs, Java Statements, Command Line Arguments, User Input to Programs, Escape Sequences Comments, Programming Style.

Data Types, Variables, and Operators :Introduction, Data Types in Java, Declaration of Variables, Data Types, Type Casting, Scope of Variable Identifier, Literal Constants, Symbolic Constants, Formatted Output with printf() Method, Static Variables and Methods, Attribute Final, Introduction to Operators, Precedence and Associativity of Operators, Assignment Operator (=), Basic Arithmetic Operators, Increment (++) and Decrement (- -) Operators, Ternary Operator, Relational Operators, Boolean Logical Operators, Bitwise Logical Operators.

Control Statements: Introduction, if Expression, Nested if Expressions, if–else Expressions, Ternary Operator ?:, Switch Statement, Iteration Statements, while Expression, do–while Loop, for Loop, Nested for Loop, For–Each for Loop, Break Statement, Continue Statement.

INTRODUCTION TO JAVA:

Java is a popular object-oriented programming language, developed by sun micro systems of USA in 1991(which has since been acquired by oracle). originally it was called as oak. the developers of java are james gosling and his team (patrick naughton, chris warth, ed frank, and mike sheridan). this language was renamed as "java" in 1995. java was publicly announced in 1995. and now more than 3 billion devices run java.

Features of Java:

- Platform Independence: Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.) and Programs developed in JAVA are executed anywhere on any system.
- Simple: They made java simple by eliminating difficult concepts of C and C++. example the concept of pointers and Java is simple because it has the same syntax of C and C++
- Object Oriented: Object oriented throughout - no coding outside of class definitions, including main ().

- Compiler/Interpreter Combo: Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making java a two-stage system.First, java compiler translates source code into bytecode. Bytecodes are not machine instructions and therefore, in the second stage, java interpreter generates machine code that can be directly executed by the machine that is running the java program so we can say that java is both compiled and interpreted language.
- Robust: because of the following concept included in java it is called as robust
    1. Exception handling
    2. strong type checking
    3. local variables must be initialized.
- Automatic Memory Management: Automatic garbage collection - memory management handled by JVM.
- Security: Security problems like eavesdropping, tampering, impersonation and virus threats can be eliminated or maintained by using java on internet.
- Dynamic Binding: The linking of data and methods to where they are located, is done at run-time.
- High Performance: The Java language supports many high-performance features such as multithreading, just-in-time compiling, and native code usage.
- Threading: A thread represents an individual process to execute a group of statements. JVM uses several threads to execute different block of code.
- Built-in Networking: Java was designed with networking in mind and comes with many classes to develop sophisticated Internet communications.
- It is open-source and free
- It is secure, fast and powerful

Java is used for:

- Mobile applications (especially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection and much, much more!

ELEMENTS OR TOKENS IN JAVA PROGRAM:

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

- Keywords
- Identifiers
- Constants
- Special Symbols
- Operators

1. **Keyword:** Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. **Java** language supports following keywords:

abstract	assert	boolean
break	byte	case
catch	char	class
const	continue	default
do	double	else
enum	exports	extends
final	finally	float
for	goto	if
implements	import	instanceof
int	interface	long
module	native	new
open	opens	package
private	protected	provides
public	requires	return
short	static	strictfp
super	switch	synchronized
this	throw	throws
to	transient	transitive
try	uses	void
volatile	while	with

2. Identifiers: Identifiers used for naming classes, methods, variables, objects, labels and interfaces in a program.

   Java identifiers follow the following rules:

   1. Identifier must start with a letter, a currency character ($), or a connecting character such as underscore (_).

2. Identifier cannot start with a number.
3. Uppercase and lowercase letters are distinct.
4. Total, TOTAL and total are different.
5. They can be of any length.
6. It should not be a keyword.
7. Spaces are not allowed.

<u>Examples of legal and illegal identifiers follow, first some legal identifiers:</u>

- int _a;
- int $c;
- int ______2_w;
- int _$;
- int this_is_a_very_detailed_name_for_an_identifier;

<u>The following are illegal (Recognize why):</u>

- int :b;
- int -d;
- int e#;
- int .f;

<u>Conventions for identifier names:</u>

1. Names of all public methods and instance variables start with lowercase, for more than one word second word's first character shold be capital.
   Ex:- total, display(), totalMarks, netSalary, getData()

2. All classes and interfaces should start with capital letter, for more then one word the second word's first character should be capital.
   Ex:- Student, HelloJava

3. Constant variables should be in capital letters and for more than one word underscore is used.
   Ex:- MAX, TOTAL_VALUE

4. Package name should be in lowercase only.
   Ex:- mypack

3. Constants: Constants are also like normal variables. But the only difference is, their values cannot be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals. Constants may belong to any of the data type.

Syntax: final data_type variable_name;

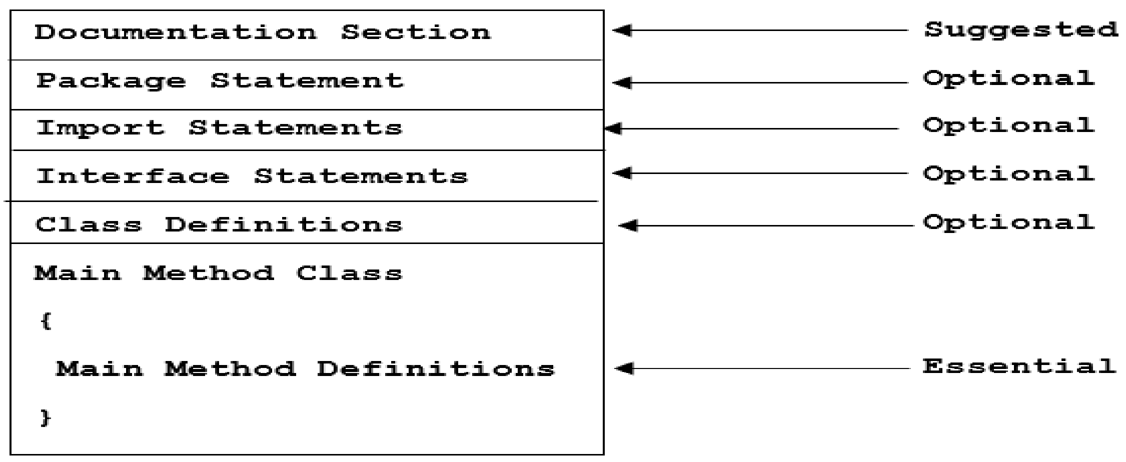
4. **Special Symbols:** The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.

   [] () {}, ; * =

   - Brackets[]: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
   - Parentheses(): These special symbols are used to indicate function calls and function parameters.
   - Braces{}: These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
   - comma (,): It is used to separate more than one statements like for separating parameters in function calls.
   - semi colon : It is an operator that essentially invokes something called an initialization list.
   - asterick (*): It is used to create pointer variable.
   - assignment operator: It is used to assign values.
5. Operators: Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-
   - Arithmetic Operators
   - Unary Operators
   - Assignment Operator
   - Relational Operators
   - Logical Operators
   - Ternary Operator(conditional operator)
   - Bitwise Operators and bit-shift operators
   - instance of operator

WRITING SIMPLE JAVA PROGRAM:

A java program may contain many classes, of which only one class should contains main() method.



Documentation section:

It consists of comment lines(program name, author, date,…….),

// - for single line comment.

/*----*/ - multiple line comments.

/**---*/ - known as documentation comment, which generated documentation automatically.

Package statement:

This is the first statement in java file. This statement declares a *package name* and informs the compiler that the class defined here belong to this package.

Ex: - package student

Import statements:

This is the statement after the package statement. It is similar to # include in c/c++.

Ex: import java.lang.String

The above statement instructs the interpreter to load the String class from the lang package.

Note:

- Import statement should be before the class definitions.
- A java file can contain N number of import statements.

Interface statements:

An interface is like a class but includes a group of method declarations.

Note: Methods in interfaces are not defined just declared.

Class definitions:

Java is a true oop, so classes are primary and essential elements of java program. A program can have multiple class definitions.

Main method class:

Every stand-alone program required a main method as its starting point. As it is true oop the main method is kept in a class definition. Every stand-alone small java program should contain at least one class with main method definition.

Rules for writing JAVA programs:

1. Every statement ends with a semicolon
2. Source file name and class name must be the same.
3. It is a case-sensitive language.
4. Every thing must be placed inside a class, this feature makes JAVA a true object oriented programming language.

Rules for file names:

- A source code file can have only one public class, and the file name must match the public class name.
- A file can have more than one non-public class.
- Files with no public classes have no naming restrictions.

Simple Java Program:

```
/* This is a simple Java program.
 Program name : Sample.java */
class Sample
{
        public static void main(String args[])
        {
                System.out.println("Hello! Vahida Welcomes U to JAVA Class");
         }
}
```

The name of the source file and name of the class name (which contains main method) must be the same, because in JAVA all code must reside inside a class.

Class Sample:

The keyword *class* is used to declare a class and *Sample* is the JAVA identifier ie name of the class.

public static void main(String args[])

public:

The keyword *public* is an access specifier that declares the *main* method as unprotected and therefore making it accessible to all other classes. The opposite of *public* is *private*.

Static:

The keyword static allows main() to be called without creating any instance of that class.

This is necessary because main() is called by java interpreter before creating any object.

Void:

The keyword *void* tells the compiler that main() does not return any value.

main():

main() is the method called when a java application begins.

String args[]:

Any information that you pass to a method is received by variables specified with in the set of parenthesis that follow the name of the method. These variables are called parameters. Here args[] is the name of the parameter of string type.

System.out.println:

It is equal to printf() or cout<<, since java is a true object oriented language, every method must be part of an object.

The *println* method is a member of *out* object, which is a static data member of *System* class.

Note: println always appends a newline character to the end of the string. So the next println prints the statements in next line.

Compiling the program:

```
C:> javac Sample.java
```

Extension is compulsory at the time of compiling.

javac is a compiler, which creates a file called Sample.class(contains the bytecode).

Note: when java source code is compiled each class is put into a separate

.class file.

Executing the program:

C:> java Sample

java is interpreter which accepts .class file name as a command line argument.

That's why the name of the source code file and class name should be equal, which avoids the confusion.

Output:

Hello! Vahida Welcomes U to JAVA Class

Important Note:

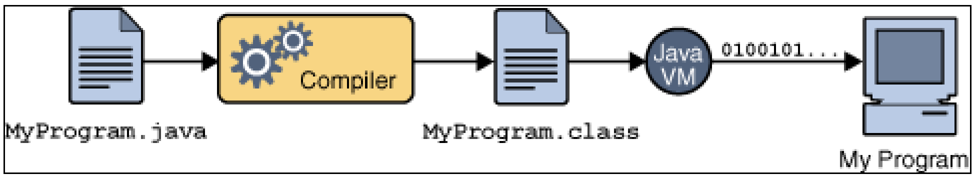
If the source code file name is Test.java and the class name is Hello, then to compile the program we have to give.

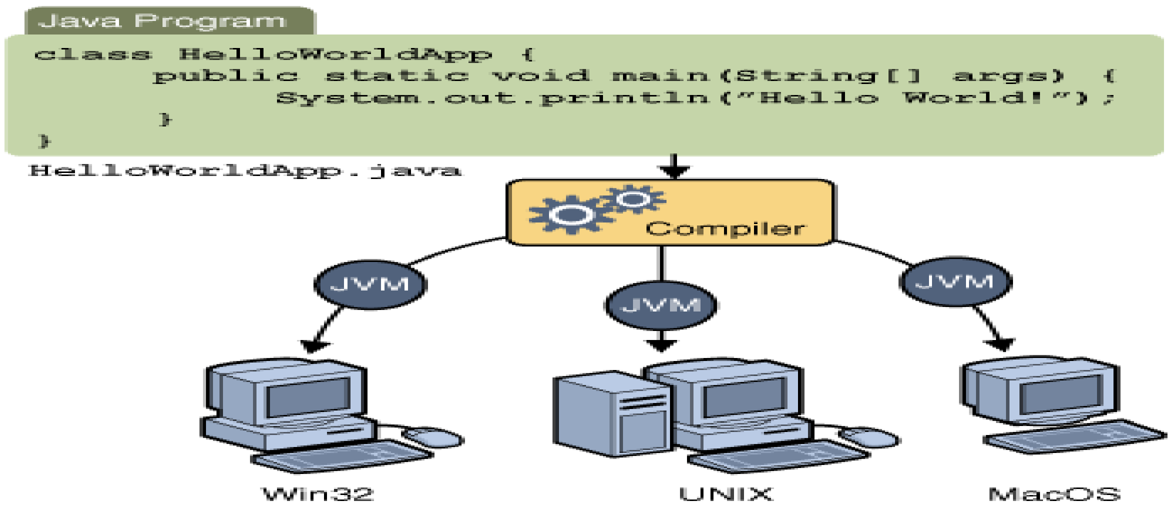
C:> Javac Test.java (which creates a Hello.class file)

To execute the program

C:> Java Hello

The following picture shows the execution of a java program/applet.

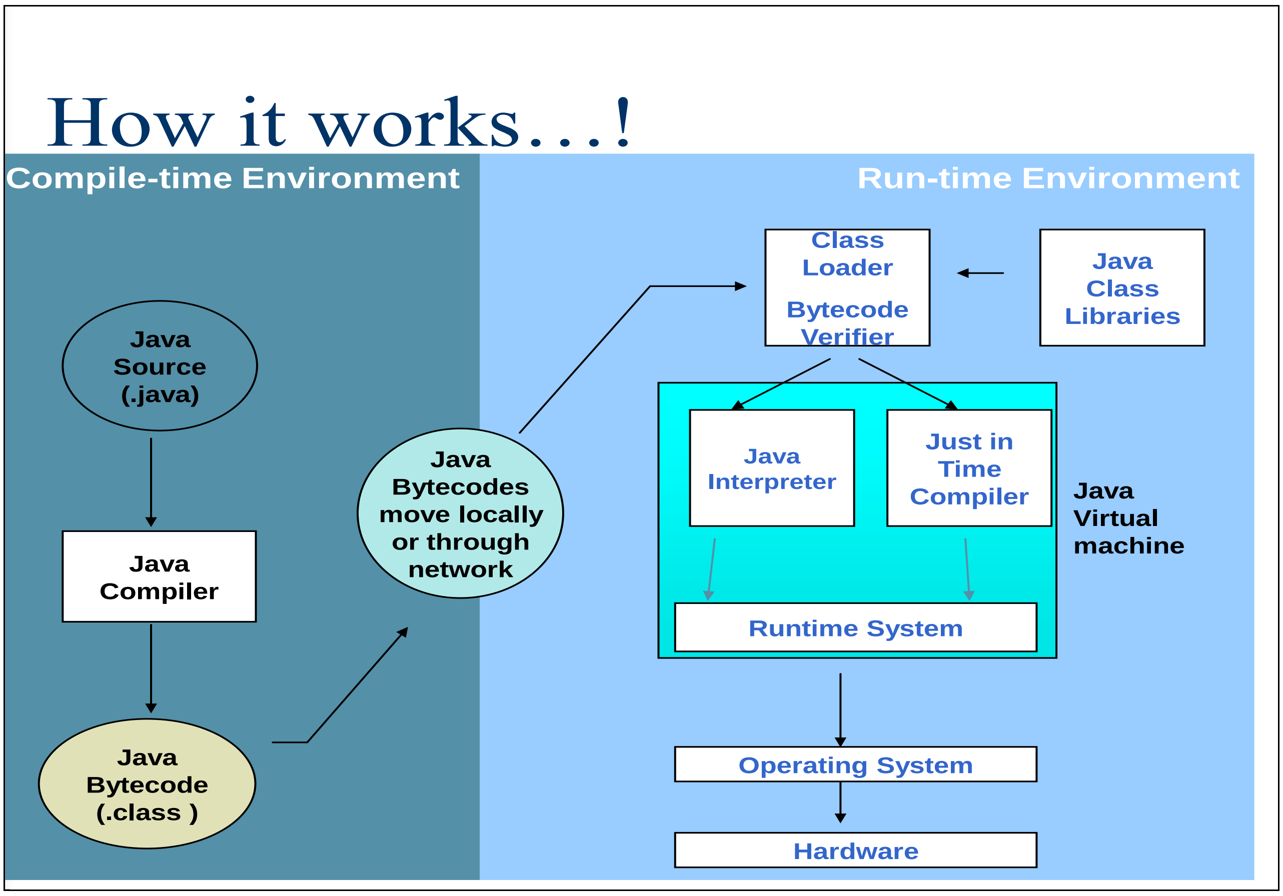




```
class HelloWorldApp {
        public static void main(String[] args) {
               System.out.println("Hello World!");
        }
}
```

<u>Note:</u> JVM is an interpreter for *bytecode*.

***Java interpreter is different from machine to machine



<u>JAVA STATEMENTS:</u>

- A statement specifies an action in a Java program.
- Java statements can be broadly classified into three categories:
    - Declaration statement
    - Expression statement
    - Control flow statement

<u>Java Declaration Statement:</u> A declaration statement is used to declare a variable.

For example,

```
int num;
int num2 = 100;
String str;
```

Java Expression Statement: An expression with a semicolon at the end is called an expression statement. For example,

```
/Increment and decrement expressions
num++;
++num;
num--;
--num;

//Assignment expressions
num = 100;
num *= 10;

//Method invocation expressions
System.out.println("This is a statement");
someMethod(param1, param2);
```

Java Flow Control Statement

By default, all statements in a Java program are executed in the order they appear in the program. Sometimes you may want to execute a set of statements repeatedly for a number of times or as long as a particular condition is true.

All of these are possible in Java using flow control statements. The if statement, while loop statement and for loop statement are examples of control flow statements.

COMMAND LINE ARGUMENTS:

Command line arguments are parameters that are passed to the application program at the time of execution. The parameters are received by args array of string type.

The first argument is stored at args[0]

The second argument is stored at args[1] and so on…..

<u>Example 1:</u>

```
class Commarg
{
 public static void main(String args[])
 {
  int i=0,l;
  l=args.length;
  System.out.println("Number of arguments are:"+l);
  while(i<l)
  {
   System.out.println(args[i]);
   i++;
  }
 }
}
```

<u>Ex:</u>

```
C:..\> java Commarg hello how are u
```

<u>output:</u>

```
number of arguments are : 4
hello
how
are
u
```

<u>Note:</u> From the above output it is clear that *hello* is stored at *args[0]* Position and so on…..

Example-2:

Program to add two no’s using command line arguments.

```
class CmdAdd2
{
 public static void main(String args[])
 {
  // if two arguments are not entered then come out
  if(args.length!=2)
  {
        System.out.println("Please enter two values....");
        return;
  }
  int a=Integer.parseInt(args[0]);
  int b=Integer.parseInt(args[1]);
  int c=a+b;
  System.out.println("The Result is:"+c);
 }
}
```

To run the program:

```
> java CmdAdd2 10 20
```

USER INPUT TO THE PROGRAM:

- The Scanner class is used to get user input, and it is found in the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class.
- For Example, we will use the nextLine() method to read String.

Input Types and their respective methods:

Method	**Description**
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Example:

```
import java.util.Scanner;

class Main {

  public static void main(String[] args) {

    Scanner s = new Scanner(System.in);

    System.out.println("Enter name, age and salary:");

    // String input

    String name = s.nextLine();

    // Numerical input

    int age = s.nextInt();

    double salary = s.nextDouble();

    // Output input by user

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

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

   System.out.println("Salary: " + salary);

  }

}
```

JAVA COMMENTS:

- Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.
- Single-line comments start with two forward slashes (//).
- Any text between // and the end of the line is ignored by Java (will not be executed).
- This example uses a single-line comment before a line of code:

  Example

  ```
  // This is a comment

  System.out.println("Hello World");
  ```

- Java Multi-line Comments
  - Multi-line comments start with /* and ends with */.
  - Any text between /* and */ will be ignored by Java.
  - This example uses a multi-line comment (a comment block) to explain the code:

    ```
    /* The code below will print the words Hello World

    to the screen, and it is amazing */

    System.out.println("Hello World");
    ```

ESCAPE SEQUENCE CHARACTERS:

A character preceded by a backslash (\) is an escape sequence and has a special meaning to the compiler.

The following table shows the Java escape sequences.

Escape Sequence	**Description**
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

JAVA VARIABLES:

- Variables are containers for storing data values.
- The Java programming language is strongly-typed, means all variables must be declared before they can be used.

Declaring (Creating) Variables: To create a variable, you must specify the type and assign it a value

Syntax: type variable = value;

Where *type* is one of Java's types (such as int or String), and *variable* is the name of the variable (such as **any identifier**). The equal sign is used to assign values to the variable.

In Java, there are different **types** of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false

Example:

```
int num = 5;
float num1 = 5.99f;
char ch = 'D';
boolean status = true;
String name = "sai";
```

Display Variables: The println() method is often used to display variables. To combine both text and a variable , use the + character:

Example

```
int num = 5;
float num1 = 5.99f;
char ch = 'D';
boolean status = true;
String name = "sai";
System.out.println("Value inside integer type variable " + num);
System.out.println("Value inside float type variable " + num1);
System.out.println("Value inside character type variable " + ch);
```

System.out.println("Value inside boolean type variable " + status);

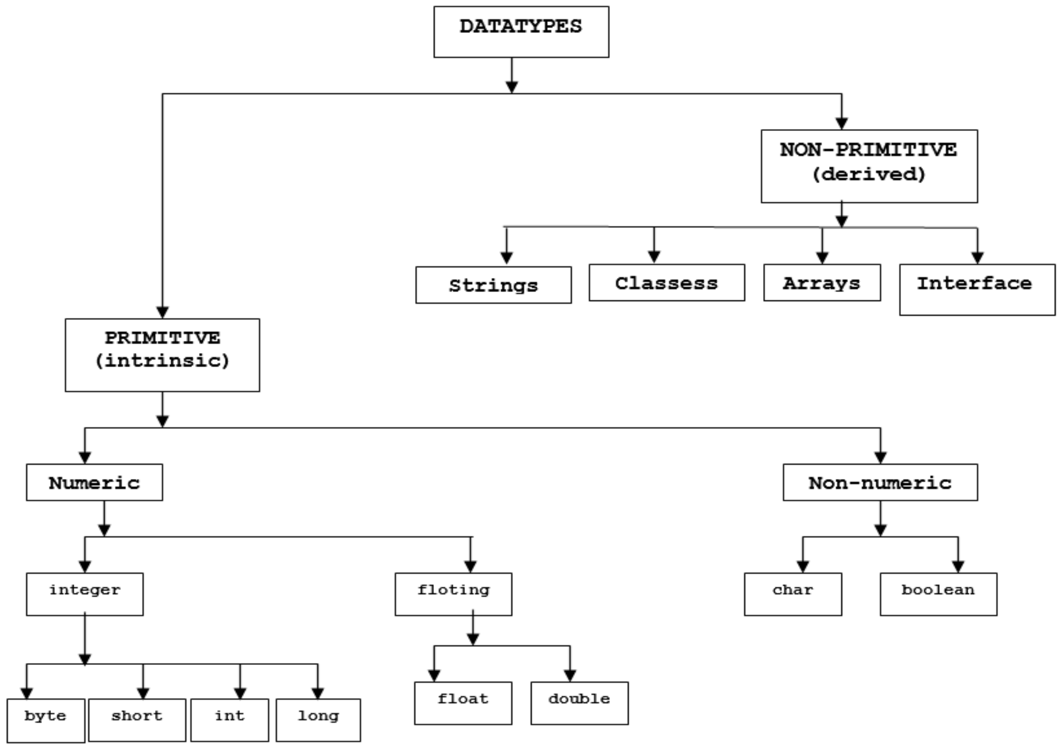
System.out.println("Value inside string type variable " + name);

Declare Many Variables: To declare more than one variable of the **same type**, use a comma-separated list:

int x = 5, y = 6, z = 50;

System.out.println(x + y + z);

DATA TYPES IN JAVA: Datatype specifies the size and types of values that a variable hold. Java is rich in it’s data types.



Primitive Data Types: A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	**Size**	**Description**
Byte	1 byte	Stores whole numbers from -128 to 127
Short	2 bytes	Stores whole numbers from -32,768 to 32,767
Int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
Long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
Char	2 bytes	Stores a single character/letter or ASCII values

Note 1: Note that you should end the value with an "f"

Note 2: Note that you should end the value with a "d"

Example:

```
int num = 5;
float num1 = 5.99f;
double num3=5.7896d;
char ch = 'D';
boolean status = true;
String name = "sai";
System.out.println("Value inside integer type variable " + num);
System.out.println("Value inside float type variable " + num1);
System.out.println("Value inside double type variable " + num1);
System.out.println("Value inside character type variable " + ch);
System.out.println("Value inside boolean type variable " + status);
System.out.println("Value inside string type variable " + name);
```

Non-Primitive Data Types:

- Non-primitive data types are called **reference types** because they refer to objects.
- The main difference between **primitive** and **non-primitive** data types are:
    1. Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
    2. Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
    3. A primitive type has always a value, while non-primitive types can be null.
    4. A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
    5. The size of a primitive type depends on the data type, while non-primitive types have all the same size.

6. Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

String: The String data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example:

```
String greet = "Hello World";
System.out.println(greet);
```

Note: A String in Java is actually a **non-primitive** data type, because it refers to an object. The String object has methods that are used to perform certain operations on strings.

Type casting: Converting one datatype into another type is called "Type casting".

Data types:

There are two types of data types.

- primitive data types or Fundamental data types.
    - These data types will represent single entity (or value).
    - Ex : int, char, byte,…….
- Referenced Data types or Advanced Data types.
    - These Data types represent several values.
    - Ex: arrays, classes, enums,…

Note: We can convert a primitive type to another primitive type and a Referenced type to another Referenced type, but to convert Primitive type to Referenced type and Referenced type to Primitive type Wrapper classes are used.

Casting primitive Data types:

- It is done in two ways
    - Widening
    - Narrowing
- The primitives are classified into two types lower types and higher types.

byte, short, char, int, long, float double

Lower←---------------------------→higher

Note:

boolean is not included because it cannot be converted into any type.

Widening primitive data types: Converting lower data type into higher data type is called widening.

Ex: 1

```
char ch='A';

int no=(int)ch;
```

....here no will contain 65 ASCII value of A.

Ex: 2

```
int x=5000;

float sal=(float)x;
```

...here sal will contain 5000.0

Widening is safe because there will not be any loss of data or precision or accuracy. So widening is done by compiler automatically.

Ex 1:

```
 char ch='A';

 int no=ch;
```

Ex 2:

```
 int x=5000;

 float sal=x;
```

widening is also known as *implicit casting.*

Narrowing implicit Data types:

Converting higher data type into lower data type is called narrowing.

Ex 1:

```
  int n=65;

  char ch=(char)n;
```

... here ch will contain A

Ex 2:

```
  double d=12.890;

  int x=(int)d;
```

.. here x will contain 12

narrowing is unsafe because there will be loss of data or precision or accuracy. So narrowing is not done by compiler automatically. The programmer has to use cast operator explicitly.

Narrowing is also called *Explicit casting.*

Static methods:

- A static method is a method that does not act upon instance variables of a class.
- A static method is declared by using the keyword static.
- Static methods are called using classname.methodName()
- The reason why static methods cannot act on instance variables is that JVM first loads static elements of the .class file one specific memory location inside the RAM and then searches for the main method, and then only it creates the objects.
- Since objects are not available at the time calling static methods, the instance variables are also not available at the time of calling the static methods.
- Static methods are also called as *class methods*.
- They cannot refer to this or super keywords (super used in inheritance)

Example:

```
class Smethod
{
  static void disp()
 {
  System.out.println("Hi....");
 }
 public static void main(String args[])
 {
   disp();
   disp();
   disp();
  }
}
```

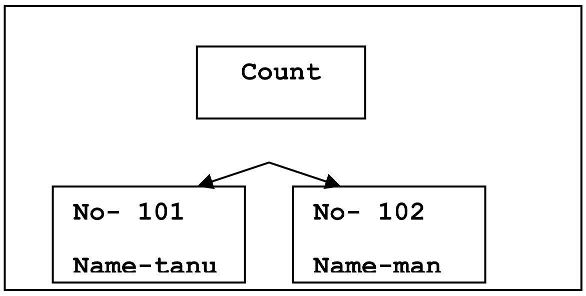
Static variable:

- The variables that are created comman to all the objects are called static variables/*class variables.*
- The static variables belong to the class as a whole rather than the objects.
- It is mainly used when we need to count no of objects created for the class

Example:

```
class emp
{
 int eno;
 String name;
 static int count;

 emp(int no,String str)
  {
   eno=no;
   name=str;
   count++;
  }
}
class EmpCount
{
   public static void main(String args[])
   {
        emp e1=new emp(101,"tan");
        emp e2=new emp(102,"manu");
        System.out.println("No of objects are:"+e1.count);
    }
}
```



Formatted Output with printf() Method(Displaying output with System.out.printf()):

To format and display the output, printf() method is available in PrintStream class. This method works similar to printf() function in C.

Note: printf() is also available from J2SE 5.0 or later.

The following format characters can be used in printf():

- %s – String
- %c – char
- %d – decimal integer
- %f – float number
- %o – octal number
- %b, %B – Boolean value
- %x, %X – hexadecimal value

- %e, %E – number scientific notation
- %n – new line character

Example :

```
class PrintfDemo

{

 public static void main(String[] args)

 {

  String s="sai";

  int n=65;

  float f=15.45f;

  System.out.printf(" String =%s %n number=%d %n HexaDecimal=%x %n
Float=%f",s,n,n,f);

  }

}
```

Attribute Final:

Final variables: It is just like const in c/c++, which restricts the user to modify. The value of final variable is finalized at the time of declaration it self.

Ex:

```
final double PI=3.14
```

example:

```
class Final
{
        public static void main(String args[])
        {
                final int a=10;
                System.out.println("before modification:"+a);
                a=a+10; //Error: cannot assign a value to final variable a a=a+10;
                System.out.println("after modification:"+a);
        }
}
```

For-each loop:

The enhanced for loop, new to java 5, is a specialized for loop that simplifies looping through an array or a collection.

Syntax:

```
for(declaration: expression)
  {
    ---
    ---
  }
```

Declaration: Its Data type should be same as the current array element.

Expression: It should be the array variable

Example:

```
class Loop
{
	public static void main(String args[])
	{
		int x[]={10,20,30,40,50};
		float y[]={1.5f,2.5f,3.5f,4.5f,5.5f};
		String names[]={"vahida","madhuri","lalitha","anil","manik"};
		for(int i:x)
		{
			System.out.println(i);
		}
		for(float j:y)
		{
			System.out.println(j);
		}
		for(String s:names)
		{
			System.out.println(s);
		}
	}
}
```

Note: For Operators and control statements concepts follow the same material what you prepared for c language.