

# UNIT-1

## Introduction to OOP

By:  
Dr. Deepak Kumar Sharma  
UPES Dehradun  
Jan May 2025 Session

# UNIT-I

- JAVA BASICS
- History of Java
- Java buzzwords
- Data types
- Variables
- Simple java program
- scope and life time of variables
- Operators, expressions, control statements
- Arrays
- Strings

# History of Java

- Java started out as a research project.
- Research began in 1991 as the **Green Project** at Sun Microsystems, Inc.
- Research efforts birthed a new language, **OAK**. ( A tree outside of the window of **James Gosling**'s office at Sun).
- It was developed as an embedded programming language, which would enable embedded system application.
- It was not really created as web programming language.

# History of Java (contd...)

Language was created with 5 main goals:

- It should be **object oriented**.
  - A single representation of a program could be executed on multiple operating systems. (i.e. **write once, run anywhere**)
  - It should fully support **network programming**.
  - It should execute code from **remote** sources securely.
  - It should be **easy** to use.
- 
- Oak was renamed Java in 1994.
  - Now Sun Microsystems is a subsidiary of Oracle Corporation.
  - Java is available as *jdk* and it is an open source s/w.

# James Gosling



# Green Team



Dr. Deepak Sharma, UPES Dehradun

# Java Logo



# Versions of Java



Version	Release date	End of Free Public Updates <sup><a href="#">[3]</a><a href="#">[8]</a><a href="#">[9]</a><a href="#">[10]</a></sup>	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	September 2003	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
Java SE 5	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018 December 2026 for Azul <sup><a href="#">[11]</a></sup>
Java SE 7	July 2011	July 2019	July 2022
Java SE 8 (LTS)	March 2014	<b>March 2022 for Oracle (commercial)</b> December 2030 for Oracle (non-commercial) December 2030 for Azul May 2026 for IBM Semeru <sup><a href="#">[12]</a></sup> At least May 2026 for Eclipse Adoptium At least May 2026 for Amazon Corretto	December 2030 <sup><a href="#">[13]</a></sup>
Java SE 9	September 2017	March 2018 for OpenJDK	—
Java SE 10	March 2018	September 2018 for OpenJDK	—

Java SE 11 (LTS)	September 2018	September 2026 for Azul October 2024 for IBM Semeru <sup>[12]</sup> At least October 2024 for Eclipse Adoptium At least September 2027 for Amazon Corretto At least October 2024 for Microsoft <sup>[14][15]</sup>	September 2026 September 2026 for Azul <sup>[11]</sup>
Java SE 12	March 2019	September 2019 for OpenJDK	—
Java SE 13	September 2019	March 2020 for OpenJDK	—
Java SE 14	March 2020	September 2020 for OpenJDK	—
Java SE 15	September 2020	March 2021 for OpenJDK March 2023 for Azul <sup>[11]</sup>	—
Java SE 16	March 2021	September 2021 for OpenJDK	—
Java SE 17 (LTS)	September 2021	September 2029 for Azul At least September 2027 for Microsoft <sup>[14]</sup> At least September 2027 for Eclipse Adoptium	September 2029 or later September 2029 for Azul

Java SE 18		22nd March 2022	September 2022	—
Java SE 19		20th September 2022	March 2023	—
Java SE 20		21st March 2023	September 2023	—
Java SE 21	LTS	19th September 2023	September 2028 for Oracle <sup>[4]</sup> September 2028 for Microsoft Build of OpenJDK <sup>[16]</sup> December 2029 for Red Hat <sup>[10]</sup> December 2029 for Eclipse Temurin <sup>[14]</sup> October 2030 for Amazon Corretto <sup>[15]</sup> September 2031 for Azul <sup>[9]</sup>	September 2031 for Oracle <sup>[4]</sup> March 2032 for BellSoft Liberica <sup>[12]</sup>
Java SE 22		19th March 2024	September 2024	—
Java SE 23		17th September 2024	March 2025 for Oracle September 2032 for Azul <sup>[9]</sup>	—
Java SE 24		March 2025	September 2025	—
Java SE 25	LTS	September 2025	September 2030 for Oracle <sup>[4]</sup>	September 2033 for Oracle <sup>[4]</sup> March 2034 for BellSoft Liberica <sup>[12]</sup>

# Features of Java (Java Buzz Words)

- Simple
- Object Oriented
- Compiled, Interpreted and High Performance
- Portable
- Reliable
- Secure
- Multithreaded
- Dynamic
- Distributed
- Architecture-Neutral

# Java Features

- **Simple**

- No pointers
- Automatic garbage collection
- Rich pre-defined class library

- **Object Oriented**

- Focus on the data (objects) and methods manipulating the data
- All methods are associated with objects
- Potentially better code organization and reuse

# Java Features

- **Compile, Interpreted and High Performance**

- Java compiler generate byte-codes, not native machine code
- The compiled byte-codes are platform-independent
- Java byte codes are translated on the fly to machine readable instructions in runtime (Java Virtual Machine)
- Easy to translate directly into native machine code by using a just-in-time compiler.

- **Portable**

- Same application runs on all platforms
- The sizes of the primitive data types are always the same
- The libraries define portable interfaces

# Java Features

- **Reliable/Robust**

- Extensive compile-time and runtime error checking
- No pointers but real arrays. Memory corruptions or unauthorized memory accesses are impossible
- Automatic garbage collection tracks objects usage over time

- **Secure**

- Java's robustness features makes java secure.
- Access restrictions are forced (private, public)

# Java Features

- **Multithreaded**

- It supports multithreaded programming.
- Need not wait for the application to finish one task before beginning another one.

- **Dynamic**

- Libraries can freely add new methods and instance variables without any effect on their clients
- Interfaces promote flexibility and reusability in code by specifying a set of methods an object can perform, but leaves open how these methods should be implemented .



# Java Features

- **Distributed**

- Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols.
- Allows objects on two different computers to execute procedures remotely by using package called *Remote Method Invocation (RMI)*.

- **Architecture-Neutral**

- Goal of java designers is “write once; run anywhere, any time, forever.”

# Java Platforms

There are three main platforms for Java:

- **Java SE** (Java Platform, Standard Edition) – runs on desktops and laptops.
- **Java ME** (Java Platform, Micro Edition) – runs on mobile devices such as cell phones.
- **Java EE** (Java Platform, Enterprise Edition) – runs on servers.

# Java Terminology

The Java platform is the name given to the computing platform from Oracle that helps users to run and develop Java applications. The platform consists of two essential softwares:

- Java Runtime Environment (JRE), which is needed to run Java applications;
- Java Development Kit (JDK), which is needed to develop those Java applications

# Java Terminology

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

# Java Terminology

## Java Development Kit:

It contains one (or more) JRE's along with the various development tools like the Java source compilers, bundling and deployment tools, debuggers, development libraries, etc.

## Java Virtual Machine:

An abstract machine architecture specified by the Java Virtual Machine Specification.

It interprets the byte code into the machine code depending upon the underlying OS and hardware combination. JVM is platform **dependent**. (It uses the class libraries, and other supporting files provided in JRE)

# Java Terminology (contd...)

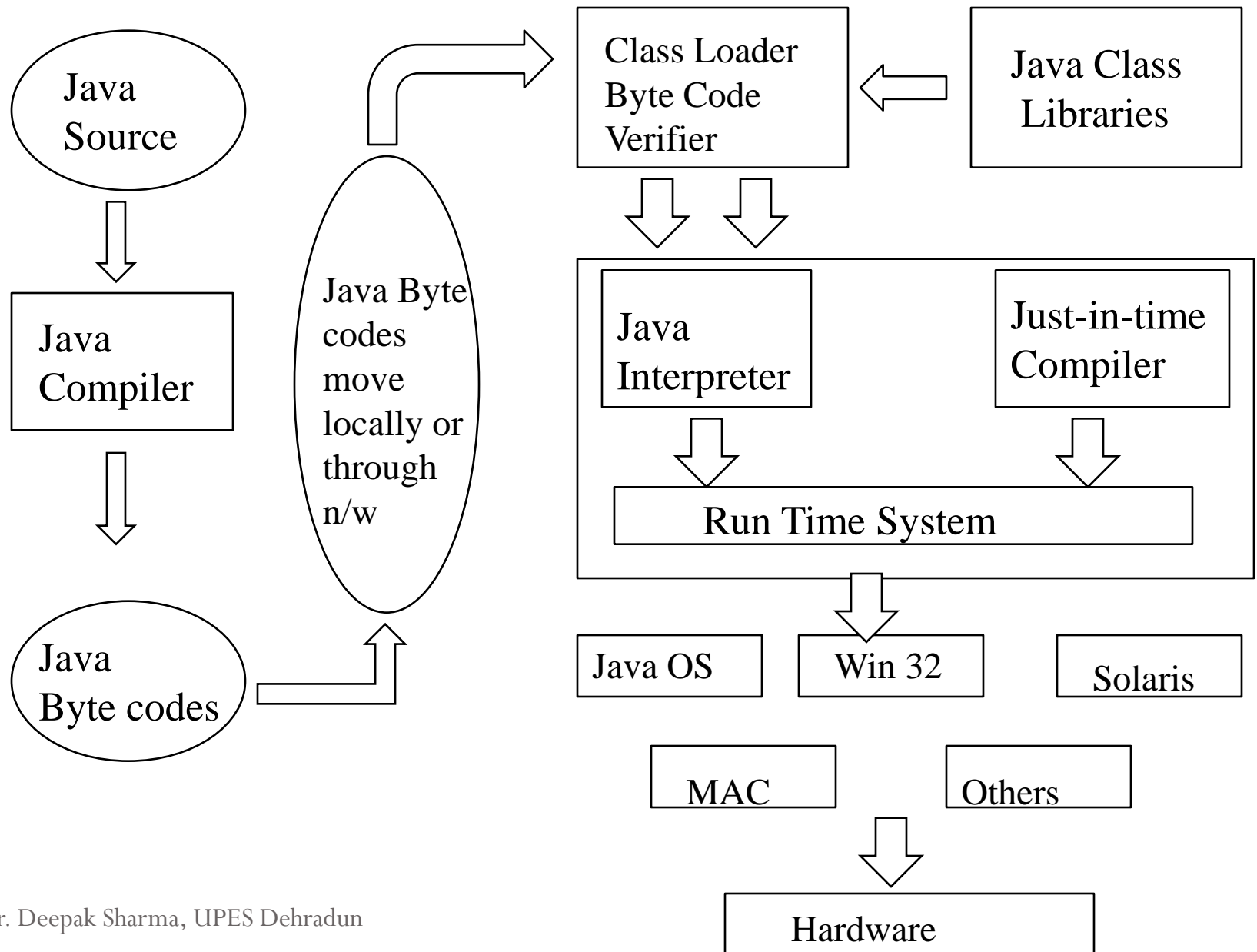
## ➤ Java Runtime Environment:

A runtime environment which implements Java Virtual Machine, and provides all class libraries and other facilities necessary to execute Java programs. This is the software on your computer that actually runs Java programs.

**JRE = JVM + Java Packages Classes** (like util, math, lang, awt, swing etc) **+runtime libraries.**

***Byte code*** is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). The JVM is an interpreter for byte code.

# Java Execution Procedure



# Binary form of a .class file(partial)

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

```
0000: cafe babe 0000 002e 001a 0a00 0600 0c09 .....
0010: 000d 000e 0800 0f0a 0010 0011 0700 1207 .....
0020: 0013 0100 063c 696e 6974 3e01 0003 2829 .....<init>...()
0030: 5601 0004 436f 6465 0100 046d 6169 6e01 V...Code...main.
0040: 0016 285b 4c6a 6176 612f 6c61 6e67 2f53 ..([Ljava/lang/S
0050: 7472 696e 673b 2956 0c00 0700 0807 0014 tring;)V.....
0060: 0c00 1500 1601 000d 4865 6c6c 6f2c 2057 .....Hello, W
0070: 6f72 6c64 2107 0017 0c00 1800 1901 0005 orld!.....
0080: 4865 6c6c 6f01 0010 6a61 7661 2f6c 616e Hello...java/lan
0090: 672f 4f62 6a65 6374 0100 106a 6176 612f g/Object...java/
00a0: 6c61 6e67 2f53 7973 7465 6d01 0003 6f75 lang/System...ou ...
```



# Creating hello java example

Let's create the hello java program:

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

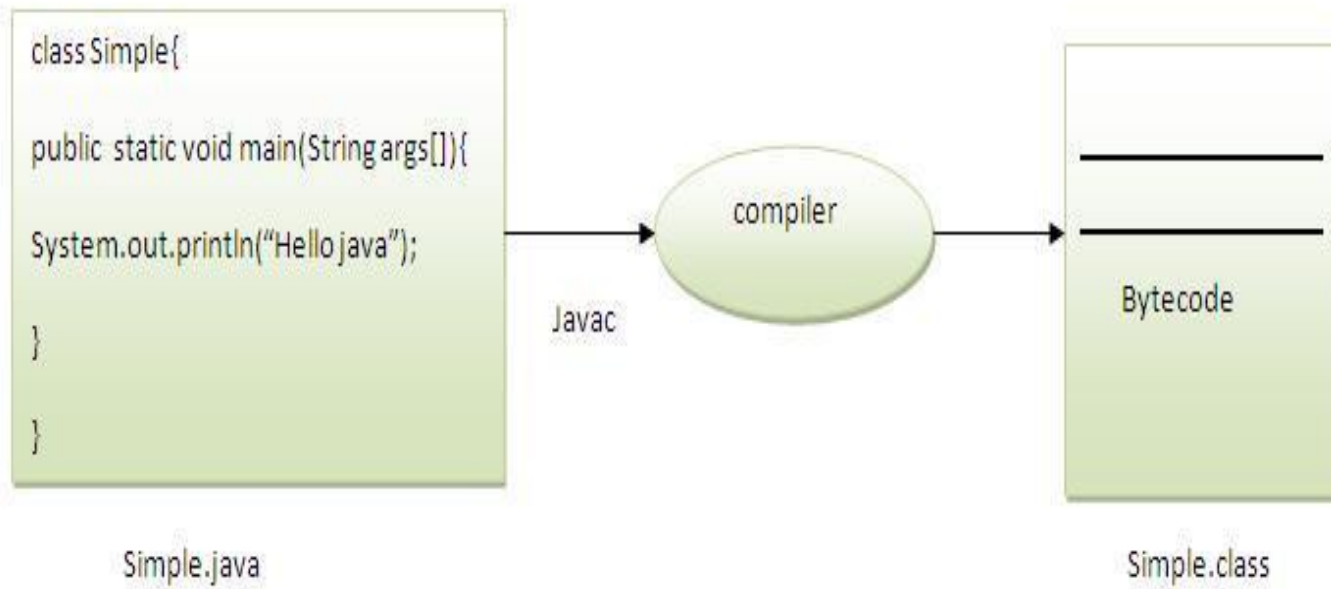
save this file as Simple.java

**To compile:** javac Simple.java

**To execute:** java Simple

**Output:** Hello Java

# At Runtime:



# Understanding first java program

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used as print statement.

# Object Oriented Programming Concepts

- Objects
- Classes
- Data abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

A **class** is collection of objects of similar type or it is a template.

**Ex:** **Student** manish;  
          ↓          ↓  
      class   object

**Objects** are instances of the type class.

The wrapping up of data and functions into a single unit ( called class) is known as **encapsulation**. Data encapsulation is the most striking features of a class.

**Abstraction** refers to the act of representing essential features without including the background details or explanations

**Inheritance** is the process by which objects of one class acquire the properties of another class. The concept of inheritance provides the **reusability**.

## **Polymorphism:**

It allows the single method to perform different actions based on the parameters.

**Dynamic Binding:** When a method is called within a program, if it is associated with the program at run time rather than at compile time is called dynamic binding.

# Benefits of OOP

- Through inheritance, we can **eliminate redundant code** and extend the use of existing classes.
- The principle of data hiding helps the programmer to build **secure** programs.
- It is easy to partition the work in a project based on objects.
- Object oriented system can be easily **upgraded** from small to large systems.
- Software complexity can be easily managed.

# Differences b/w C++ and Java

## C++

1. Global variable are supported.
2. Multiple inheritance is supported.

## Java

1. No Global variables.  
Everything must be inside the class only.
2. No direct multiple Inheritance.



# C++

3. Constructors and Destructors supported.

4. In c++ pointers are supported.

5. C++ supporting ASCII character set.

# Java

3. Java supporting constructors only & instead of destructors garbage collection is supported.

4. No pointer arithmetic in Java.

5. Java supports Uni code Character set.

Topic	Java	Python
<b>Compilation process</b>	Java is both compiled and interpreted language. The source code is first compiled and converted to bytecode, and afterward, it depends on JVM whether the bytecode will be compiled or interpreted	Python is an interpreted language, i.e., it is compiled and executed simultaneously line by line.
<b>Length of code</b>	The length of the code of java programs is more as compared to that of Python as every program has to be written in a class. For Eg- to write hello world program, the code is- <pre>public class HelloWorld {     public static void <b>main</b>(String[] args) {         System.out.<b>println</b>("Hello, World");     } }</pre>	Python has shorter lines of code as you directly write the code and it gets interpreted. For eg- <pre>print('Hello, world!')</pre>
<b>Complexity of syntax</b>	Java is a statically typed programming language. There are hardcore rules for braces and semi-colon.	Python is dynamically typed and there are no hardcore rules for semi-colon and braces. It works on indentation.
<b>Ease of typing</b>	Strongly typed, need to define exact types of variables.	Dynamically typed, no need to define the exact type of variables.
<b>Speed</b>	Java is faster as compared to Python.	Python is relatively slow as it is interpreted language and it determines the type of the variable at run time, which makes it slow.
<b>Usage</b>	It has been in trend for a long time and is vastly used in Android application development, embedded systems, and web applications.	Data science and machine language are made very simple, using Python. Also, it is being used for web development.

# Points to remember

- **Case Sensitivity** – Java is case sensitive, which means identifier **Hello** and **hello** have different meaning in Java.
- **Class Names** – For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
- **Example:** *class MyFirstJavaClass*
- **Method Names** – All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
- **Example:** *public void myMethodName()*

# Points to remember

- **Program File Name** – Name of the program file should match the class name, if class is public.

**Note:** In case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.

- When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).
- **Example:** Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as '*MyFirstJavaProgram.java*'
- **public static void main(String args[])** – Java program processing starts from the main() method which is a mandatory part of every Java program.

# Java Tokens

Java tokens are smallest elements of a program which are identified by the compiler. Tokens in java include:

- Keywords
- Identifiers
- Literals
- Operators
- Separators
- Comments

# Keywords

- These are the **pre-defined** reserved words of any programming language.
- Each keyword has a special meaning.
- It is always written in lower case.

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. thro	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

# Identifier

Java components require names. Names used for classes, variables, and methods are called **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows —

- All identifiers should begin with a letter (A to Z or a to z), currency character (\$) or an underscore (\_).
- After the first character, identifiers can have any combination of characters.
- A keyword cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, \$salary, \_value, \_\_1\_value.
- Examples of illegal identifiers: 123abc, -salary.

# Data Types

- Java Is a Strongly Typed Language
  - Every variable has a type, every expression has a type, and every type is strictly defined.
  - All assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.
  - There are no automatic conversions of conflicting types as in some languages.
- *For example, in C/C++ you can assign a floating-point value to an integer. In Java, you cannot.*



# Data Types

## Simple Type

## Derived Type

## User Defined Type

E.g: Array, String...

Numeric Type

Non-Numeric

class

Interface

Integer

Float

Char

Boolean

float

double

byte

short

int

long

# Primitive Data Types

## byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 ( $-2^7$ )
- Maximum value is 127 (inclusive) ( $2^7 - 1$ )
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

# Primitive Data Types

## **short**

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 ( $-2^{15}$ )
- Maximum value is 32,767 (inclusive) ( $2^{15} - 1$ )
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

# Primitive Data Types

## **int**

- int data type is a 32-bit signed two's complement integer.
- Minimum value is  $-2,147,483,648$  ( $-2^{31}$ )
- Maximum value is  $2,147,483,647$  (inclusive) ( $2^{31} - 1$ )
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: `int a = 100000, int b = -200000`

# Primitive Data Types

## long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is  $-9,223,372,036,854,775,808$  ( $-2^{63}$ )
- Maximum value is  $9,223,372,036,854,775,807$  (inclusive) ( $2^{63} - 1$ )
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

# Primitive Data Types

## **float**

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

# Primitive Data Types

## **double**

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0
- Example: `double d1 = 123.4`

# Primitive Data Types

## **boolean**

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: `boolean one = true`



# Primitive Data Types

## **char**

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

# Data Types

<u>Name</u>	<u>Width in bits</u>	<u>Range</u>
long	64	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	−2,147,483,648 to 2,147,483,647
short	16	−32,768 to 32,767
byte	8	−128 to 127
double	64	4.9e−324 to 1.8e+308
float	32	1.4e−045 to 3.4e+038
char	16	0 to 65,536.

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

        System.out.println(" For an Integer ");
        System.out.println("Size is : "+Integer.SIZE);

        int i1 = Integer.MAX_VALUE;
        int i2 = Integer.MIN_VALUE ;

        System.out.println("Max value is : "+i1);
        System.out.println("Min Value is : "+i2);

        System.out.println(" For an Byte");
        System.out.println("Size is : "+Byte.SIZE);

        byte b1 = Byte.MAX_VALUE;
        byte b2 = Byte.MIN_VALUE ;

        System.out.println("Max value is : "+b1);
        System.out.println("Min Value is : "+b2);
```

```
        System.out.println(" For an Short");
        System.out.println("Size is : "+Short.SIZE);

        short s1 = Short.MAX_VALUE;
        short s2 = Short.MIN_VALUE ;

        System.out.println("Max value is : "+s1);
        System.out.println("Min Value is : "+s2);

        System.out.println(" For an Long");
        System.out.println("Size is : "+Long.SIZE);

        long l1 = Long.MAX_VALUE;
        long l2 = Long.MIN_VALUE ;

        System.out.println("Max value is : "+l1);
        System.out.println("Min Value is : "+l2);

    }
}
```

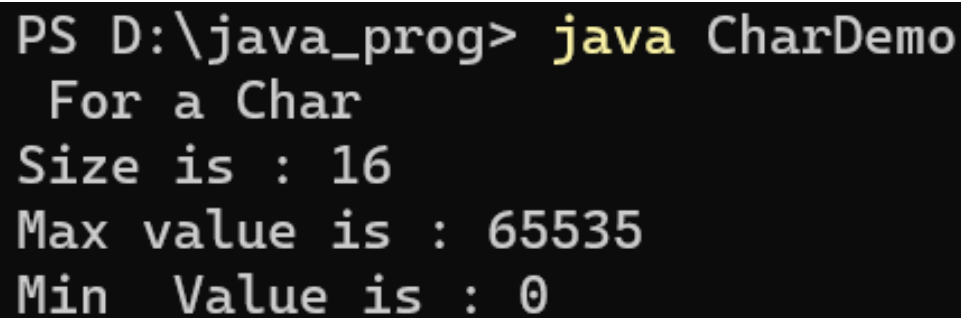
# Output

```
PS D:\java_prog> javac IntDemo.java
PS D:\java_prog> java IntDemo
  For an Integer
Size is : 32
Max value is : 2147483647
Min Value is : -2147483648
  For an Byte
Size is : 8
Max value is : 127
Min Value is : -128
  For an Short
Size is : 16
Max value is : 32767
Min Value is : -32768
  For an Long
Size is : 64
Max value is : 9223372036854775807
Min Value is : -9223372036854775808
```

```
public class FloatDemo{  
    public static void main(String args[]){  
  
        System.out.println(" For an Float");  
        System.out.println("Size is : "+Float.SIZE);  
  
        float f1 = Float.MAX_VALUE;  
        float f2 = Float.MIN_VALUE ;  
  
        System.out.println("Max value is : "+f1);  
        System.out.println("Min Value is : "+f2);  
  
        System.out.println(" For an Double");  
        System.out.println("Size is : "+Double.SIZE);  
  
        double d1 = Double.MAX_VALUE;  
        double d2 = Double.MIN_VALUE ;  
  
        System.out.println("Max value is : "+d1);  
        System.out.println("Min Value is : "+d2);  
    }  
}
```

```
PS D:\java_prog> java FloatDemo  
For an Float  
Size is : 32  
Max value is : 3.4028235E38  
Min Value is : 1.4E-45  
For an Double  
Size is : 64  
Max value is : 1.7976931348623157E308  
Min Value is : 4.9E-324
```

```
public class CharDemo {  
    public static void main(String args[]) {  
  
        System.out.println(" For a Char");  
        System.out.println("Size is : "+Character.SIZE);  
        int f1 = Character.MAX_VALUE;  
        long f2 = Character.MIN_VALUE ;  
        System.out.println("Max value is : "+f1);  
        System.out.println("Min Value is : "+f2);  
    }  
}
```



A terminal window with a black background and white text. The prompt is 'PS D:\java\_prog>'. The command 'java CharDemo' has been executed. The output consists of four lines: 'For a Char', 'Size is : 16', 'Max value is : 65535', and 'Min Value is : 0'.

```
PS D:\java_prog> java CharDemo  
For a Char  
Size is : 16  
Max value is : 65535  
Min Value is : 0
```

# Java Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Java provides five types of literals are as follows:

- Integer
- Floating Point
- Character
- String
- Boolean

Literal	Type
23	int
9.86	double
false, true	boolean
'K', '7', '-'	char
"HelloJava"	String
null	any reference type

- Literals can be assigned to any primitive type variable. E.g.  
byte a = 68;                      char a = 'A';
- Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example :  
int decimal = 100;              int octal = 0144;              int hexa = 0x64;

# Escape Sequence Characters

Notation	Character represented
<code>\n</code>	Newline (0x0a)
<code>\r</code>	Carriage return (0x0d)
<code>\f</code>	Formfeed (0x0c)
<code>\b</code>	Backspace (0x08)
<code>\s</code>	Space (0x20)
<code>\t</code>	tab
<code>\"</code>	Double quote
<code>\'</code>	Single quote
<code>\\</code>	backslash
<code>\ddd</code>	Octal character (ddd)
<code>\uxxxx</code>	Hexadecimal UNICODE character (xxxx)



# Comments

- Comments allow us to specify information about the program inside our Java code. Java compiler recognizes these comments as tokens but excludes it from further processing. The Java compiler treats comments as whitespaces. Java provides the following two types of comments:
  - **Single Line:** It begins with a pair of forwarding slashes (//).
  - **Multiple Line:** It begins with /\* and continues until it finds \*/.

# Separators

- The separators in Java is also known as **punctuators**. There are nine separators in Java, are as follows:
- separator <= ; | , | . | ( | ) | { | } | [ | ]
- **Square Brackets []**: It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.
- **Parentheses ()**: It is used to call the functions and parsing the parameters.
- **Curly Braces {}**: The curly braces denote the starting and ending of a code block.
- **Comma (,)**: It is used to separate two values, statements, and parameters.
- **Assignment Operator (=)**: It is used to assign a variable and constant.
- **Semicolon (;)**: It is the symbol that can be found at end of the statements. It separates the two statements.
- **Period (.)**: It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

# Operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

# Arithmetic Operators(1)

**Operator**

**Result**

+

Addition

—

Subtraction

\*

Multiplication

/

Division

%

Modulus

# Arithmetic Operators(2)

Operator	Result
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

## Example:

```
class IncDec{  
    public static void main(String args[]) {  
        int a = 1;  
        int b = 2;  
        int c,d;  
        c = ++b;  
        d = a++;  
        c++;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("c = " + c);  
        System.out.println("d = " + d);  
    }  
}
```

```
class OpEquals {  
    public static void main(String args[]) {  
        int a = 1;  
        int b = 2;  
        int c = 3;  
        a += 5;  
        b *= 4;  
        c += a * b;  
        c %= 6;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("c = " + c);  
    }  
}
```

# Bitwise Operators(1)

- *bitwise operators* can be applied to the integer types, **long**, **int**, **short**, **byte** and **char**.
- These operators act upon the individual bits of their operands.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
<<	Shift left



# Bitwise Operators(2)

## Operator

## Result

**&=**

Bitwise AND assignment

**|=**

Bitwise OR assignment

**^=**

Bitwise exclusive OR assignment

**>>=**

Shift right assignment

**<<=**

Shift left assignment

A	B	A   B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

```

00101010    42
&00001111    15
-----
00001010    10

```

```

int a = 64;
a = a << 2;    256

```

```

00101010    42
^00001111    15
-----
00100101    37

```

```

int a = 32;
a = a >> 2;    8

```

# Relational Operators

- The *relational operators* determine the relationship that one operand has to the other.
- They determine equality and ordering.

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Example:

```
public class RelationalOperatorsDemo
{
    public static void main(String args[])
    {
        int x=10,y=5;
        System.out.println("x>y:"+(x>y));
        System.out.println("x<y:"+(x<y));
        System.out.println("x>=y:"+(x>=y));
        System.out.println("x<=y:"+(x<=y));
        System.out.println("x==y:"+(x==y));
        System.out.println("x!=y:"+(x!=y));
    }
}
```

# Boolean Logical Operators(1)

Operator	Name	Description	Example	
&&	Logical and	Returns true if both statements are true	<code>x &lt; 5 &amp;&amp; x &lt; 10</code>	
	Logical or	Returns true if one of the statements is true	<code>x &lt; 5    x &lt; 4</code>	
!	Logical not	Reverse the result, returns false if the result is true	<code>!(x &lt; 5 &amp;&amp; x &lt; 10)</code>	

**&&** // Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.

```
class Test{  
    public static void main(String args[]){  
        int a=5,b=20;  
        if ( a+b>40 && ++b< 10)  
            System.out.println("Hi");  
        System.out.println("b="+b);  
    }  
}
```

```
public class TernaryOperatorDemo
{
    public static void main(String args[])
    {
        int x=10,y=12;
        int z;
        z= x > y ? x : y;
        System.out.println("Z="+z);
    }
}
```

# Expressions

- An expression is a combination of constants (like 10), operators ( like +), variables(section of memory) and parentheses ( like “(” and “)” ) used to calculate a value.

Ex1:             $x = 1;$

Ex2:             $y = 100 + x;$

Ex3:             $x = (32 - y) / (x + 5)$



# Java Operator Precedence Table

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Left to Right
13	++ -- + - ! ~ ( type )	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right

10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5		Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3		Logical OR	Left to right
2	? : = += -= *= /=	Ternary conditional Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment	Right to left
1	%=	Modulus assignment	Right to left

*Note: Larger number means higher precedence.*

# Control Statements

- Selection Statements:      **if & switch**
- Iteration Statements:      **for, while, do-while, for each**
- Jump Statements:      **break, continue and return**

# Selection Statements

```
if (condition)  
    statement1;  
else  
    statement2;
```

```
if(condition)  
    statement;  
else if(condition)  
    statement;  
else if(condition)  
    statement;  
...  
else  
    statement;
```

```
switch (expression)  
{  
    case value1:  
        // statement sequence  
        break;  
    case value2:  
        // statement sequence  
        break;  
    ...  
    case valueN:  
        // statement sequence  
        break;  
    default:  
        // default statement sequence  
}
```

The *condition* is any expression that returns a **boolean** value.

### **Switch(Expression):**

The *expression* must be of type **byte, short, int, or char**;  
Each of the *values* specified in the **case** statements must be of a type compatible with the expression.

# Iteration Statements

**while**(*condition*)

{

    // body of loop

}

**do**

{

    // body of loop

} **while** (*condition*);

**for**(*initialization; condition; iteration*)

{

    // body

}

# For-each loop

- It starts with the keyword **for** like a normal for-loop.
- Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
- In the loop body, you can use the loop variable you created rather than using an indexed array element.
- It's commonly used to iterate over an array or a Collections class (eg, ArrayList)

## Syntax:

```
for (type var : array)
{
    statements using var;
}
```



```
for (int i=0; i<arr.length; i++)
{
    type var = arr[i];
    statements using var;
}
```

# Example: For Each Loop

```
class For_Each
{
    public static void main(String[] arg) {
        int[] marks = { 125, 172, 95, 116, 110 };
        int highest_marks = marks[0];
        for (int num : marks)
        {
            if (num > highest_marks)
            {
                highest_marks = num;
            }
        }
        System.out.println("The highest score is " + highest_marks);
    }
}
```

**OUTPUT: The highest score is 172**

# Jump Statements

**continue;**    **//bypass the followed instructions**

**break;**    **//exit from the loop**

**label:**

**- - - -**

**- - - -**

**break label;**            **//it's like goto**  
**statement**

**return;**    **//control returns to the caller**



# Sample Program

```
class HelloWorld {  
    public static void main (String args []) {  
        System.out.println (“Welcome to Java Programming.....”);  
    }  
}
```

***public*** allows the program to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared. In this case, main ( ) must be declared as public, since it must be called by code outside of its class when the program is started.

*static* allows `main( )` to be called without having to instantiate a particular instance of the class. This is necessary since `main ( )` is called by the Java interpreter before any objects are made.

*void* states that the main method will not return any value.

*main()* is called when a Java application begins. In order to run a class, the class must have a main method.

*string args[]* declares a parameter named `args`, which is an array of `String`. In this case, `args` receives any command-line arguments present when the program is executed.

**System** is a class which is present in **java.lang** package.

**out** is a static field present in System class which returns a **PrintStream** object. As out is a static field it can call directly with classname.

**println()** is a method which present in PrintStream class which can call through the PrintStream object return by static field **out** present in System class to print a line to console.

# Examples

```
class Main {  
    public static void main(String[] args) {  
        int num = 15;  
        if (num >= 10 || num <= 20) {  
            System.out.println("Number is between 10 and 20");  
        }  
        else {  
            System.out.println("Number is out of range");  
        }  
    }  
}
```

**Predict Output=?**

**Any Issues in code?**

What will be the output if num is 35?

# Examples

```
class Main {  
    public static void main(String[] args) {  
        int i = 1;  
        while (i <= 5) {  
            System.out.println(i);  
        }  
    }  
}
```

**Predict Output=?**

**Any Issues in code?**

# Example:

```
class Main{  
    public static void main(String[] args) {  
        int sum = 0;  
        for (int i = 1; i < 10; i++) {  
            sum += i;  
        }  
        System.out.println("Sum: " + sum);  
    }  
}
```

**What are we calculating?**  
**Is it for finding sum of first 10 natural numbers?**  
**Predict Output=?**

# Questions For Practice

1. Write a Java program to check if a number is positive or negative using an if statement.
2. Write a Java program to check if a number is even or odd using an if-else statement.
3. Using if-else-if Ladder, write a Java program to grade students based on their marks:

Marks  $\geq$  90: Grade A

Marks  $\geq$  80: Grade B

Marks  $\geq$  70: Grade C

Marks  $<$  70: Fail

4. Write a Java program to check if a number is divisible by both 3 and 5.
5. Write a Java program to print the season based on the month number:

12, 1, 2: Winter

3, 4, 5: Spring

6, 7, 8: Summer

9, 10, 11: Autumn

6. Write a Java program to calculate the discount on a product based on its price:

Price  $>$  1000: 20% discount

Price  $>$  500: 10% discount

Otherwise: 5% discount

# Predict Output

```
for (int i = 0; i < 5; i--) {  
    System.out.println("Hello");  
}  
System.out.println("Done");
```

```
int i = 0;  
do {  
    System.out.print(i++ + " ");  
} while (i < 3);
```

```
int i = 5;  
while (i-- > 0) {  
    System.out.print(i + " ");  
}
```

```
for (int i = 0, j = 5; i < j; i++, j--) {  
    System.out.print(i + "," + j + " ");  
}
```

```
int i = 0;  
for (; i < 5; i++);  
System.out.println(i);
```

```
for (int i = 0; i < 5; i += 2, System.out.print(i + " "));
```



# Practice Questions

Q1. Write a while loop that takes an integer num = 9876 and prints its digits in reverse order.

Q2. Write a for loop to calculate the factorial of N = 8.

Q3. Write a loop to display all prime numbers between 1 to 100.

Q4. Write a while loop that checks if a given integer num = 121 is a palindrome

# Classes in Java

```
public class Person {
```

```
    String name;  
    int age;  
    String city;  
    String gender
```

```
    void eat() {  
    }
```

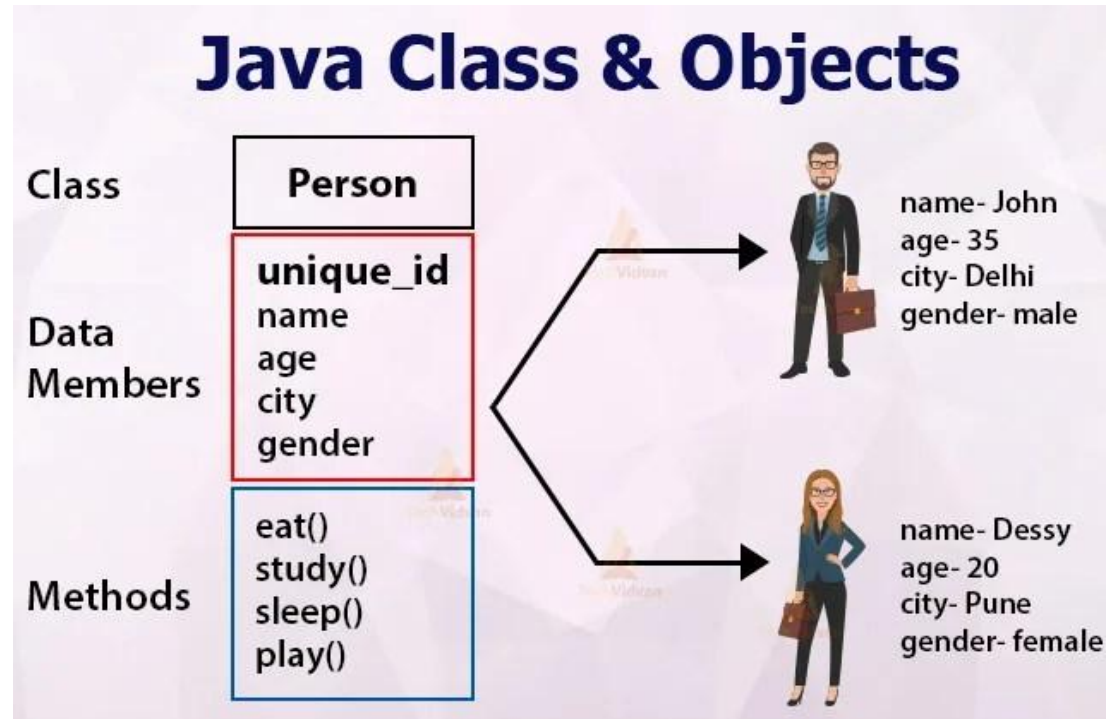
```
    void study() {  
    }
```

```
    void sleep() {  
    }
```

```
    void play() {  
    }
```

```
}
```

Dr. Deepak Sharma, UPES Dehradun



# Constructor

- Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.
- Each time a new object is created, at least one constructor will be invoked.
- The main rule of constructors is that they should have the same name as the class.
- A class can have more than one constructor.

```
public class Puppy {  
  
    public Puppy() {  
    }  
  
    public Puppy(String name) {  
        // This constructor has one parameter,  
        name.  
    }  
}
```

# Creating Objects

```
class sample {  
    public static void main(String args[]) {  
        System.out.println("sample:main");  
        sample s=new sample();  
        s.display();  
  
    }  
    void display() {  
        System.out.println("display:main");  
    }  
}
```

# Creating Objects

There are three steps when creating an object from a class –

Declaration – A variable declaration with a variable name with an object type.

Instantiation – The 'new' keyword is used to create the object.

Initialization – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

```
public class Puppy {  
    public Puppy(String name) {  
        // This constructor has one parameter, name.  
        System.out.println("Passed Name is :" + name );  
    }  
  
    public static void main(String []args) {  
        // Following statement would create an object myPuppy  
        Puppy myPuppy = new Puppy( "tommy" );  
    }  
}
```

# Variables

- The variable is the basic unit of storage in a Java program.
- A variable is defined by the combination of an identifier, a type, and an optional initializer.

## Declaring a Variable

- In Java, all variables must be declared before they can be used.  
*type identifier [ = value ][, identifier [= value] ...] ;*

## Types

- Instance Variable
- Class/Static Variable
- Local Variable

## **Local variables :**

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor or block.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

# Example

```
public class Test {  
    public void pupAge() {  
        int age = 0;  
        age = age + 7;  
        System.out.println("Puppy age is : " + age);  
    }  
    public static void main(String args[]) {  
        Test test = new Test();  
        test.pupAge();  
    }  
}
```

Output  
Puppy age is: 7



# Example

```
public class Test {  
    public void pupAge() {  
        int age;  
        age = age + 10;  
        System.out.println("Puppy age is : " + age);  
    }  
}
```

```
public static void main(String args[]) {  
    Test test = new Test();  
    test.pupAge();  
}  
}
```

Output

Test.java:4:variable number might not have  
been initialized

age = age + 10;

^

1 error

## Instance variables :

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the key word '**new**' and destroyed when the object is destroyed.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class.
- Instance variables have default values.
- Instance variables can be accessed directly by calling the variable name inside the class.
- However within static methods and different class ( when instance variables are given accessibility) that should be called using the fully qualified name **ObjectReference.VariableName**

## Class/Static variables :

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are stored in static memory.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables.
- Default values are same as instance variables.
- Static variables can be accessed by calling with the class name **ClassName.VariableName**

```
class Variables{
```

```
    int i;
```

```
    public int j
```

```
    static long l=10;
```

```
    public static float f;
```

```
    char c;
```

```
    boolean b;
```

```
    void display(int a){
```

```
        i=a;
```

```
        System.out.println("i value in display: "+i);
```

```
    }
```

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

```
        double d=0.0;
```

```
        //public double d=0.0; invalid
```

```
        Variables v1=new Variables();
```

```
        Variables v2=new Variables();
```

```
        Variables v3=new Variables();
```

```
        v1.display(100);
```

```
        v1.i=2;
```

```
        v2.i=3;
```

```
        v3.i=4;
```

```
        System.out.println("i value is: "+v1.i);
```

```
        System.out.println("i value is: "+v2.i);
```

```
        System.out.println("i value is: "+v3.i);
```

```
        System.out.println("i value is: "+v1.j);
```

```
        v1.l=20;
```

```
        v2.l=30;
```

```
        v3.l=40;
```

```
        System.out.println("l value is: "+v1.l);
```

```
        System.out.println("l value is: "+v2.l);
```

```
        System.out.println("l value is: "+v3.l);
```

```
        System.out.println("f value is: "+f);
```

```
        System.out.println("c value is: "+v1.c);
```

```
        System.out.println("b value is: "+v1.b);
```

```
        System.out.println("d value is: "+d);
```

```
    }
```

```
}
```

```
class Variables{
```

```
    int i;//instance variable
```

```
    public int j ;//instance variable
```

```
    static long l=10;//class variable
```

```
    public static float f;//class variable
```

```
    char c;//instance variable
```

```
    boolean b;//instance variable
```

```
    void display(int a){
```

```
        i=a;
```

```
        System.out.println("i value in display: "+i);
```

```
    }
```

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

```
        double d=0.0;//local variable
```

```
        //public double d=0.0; invalid
```

```
        Variables v1=new Variables();
```

```
        Variables v2=new Variables();
```

```
        Variables v3=new Variables();
```

```
        v1.display(100);
```

```
        v1.i=2;
```

```
        v2.i=3;
```

```
        v3.i=4;
```

```
        System.out.println("i value is: "+v1.i);
```

```
        System.out.println("i value is: "+v2.i);
```

```
        System.out.println("i value is: "+v3.i);
```

```
        System.out.println("i value is: "+v1.j);
```

```
        v1.l=20;
```

```
        v2.l=30;
```

```
        v3.l=40;
```

```
        System.out.println("l value is: "+v1.l);
```

```
        System.out.println("l value is: "+v2.l);
```

```
        System.out.println("l value is: "+v3.l);
```

```
        System.out.println("f value is: "+f);
```

```
        System.out.println("c value is: "+v1.c);
```

```
        System.out.println("b value is: "+v1.b);
```

```
        System.out.println("d value is: "+d);
```

```
    }
```

```
}
```

# The Scope and Lifetime of Variables

## Scope

The *scope* of a declared element is the portion of the program where the element is visible.

## Lifetime

The *lifetime* of a declared element is the period of time during which it is alive.

The lifetime of the variable can be determined by looking at the context in which they're defined.

- Java allows variables to be declared within any block.
- A block begins with an opening curly brace and ends by a closing curly brace.

- Variables declared inside a scope are not accessible to code outside.
- Scopes can be nested. The outer scope encloses the inner scope.
- Variables declared in the outer scope are visible to the inner scope.
- Variables declared in the inner scope are not visible to the outside scope.

```
public class Scope
{
    public static void main(String args[]){
        int x; //known to all code within main
        x=10;
        if(x==10){ // starts new scope
            int y=20; //Known only to this block
            //x and y are both known here
            System.out.println("x and y: "+x+" "+y);
            x=y+2;
        }
        // y=100; // error ! y not known here
        //x is still known here
        System.out.println("x is "+x);
    }
}
```



Unit 1 Cont..  
Continue Practicing!!!