

# Programming with Java

## Introduction to OOP's, Classes and Objects

### Unit - 1



# Points to be covered

## **Introduction to OOP's:**

- What is OOP
- Difference between Procedural and Object-oriented programming
- Basic OOP concept - Object, classes, abstraction, encapsulation, inheritance, polymorphism
- History of Java
- Features of Java
- Difference between C++ & JAVA
- JDK Environment
- Java Virtual Machine
- Java Tokens
- Data Types
- Type Conversion and type casting
- Garbage Collection
- Array (One Dimensional, Rectangular and Jagged Array)
- Conditional Statements (if, switch)
- Looping Statement (while, for, do-while)
- Command Line Arguments

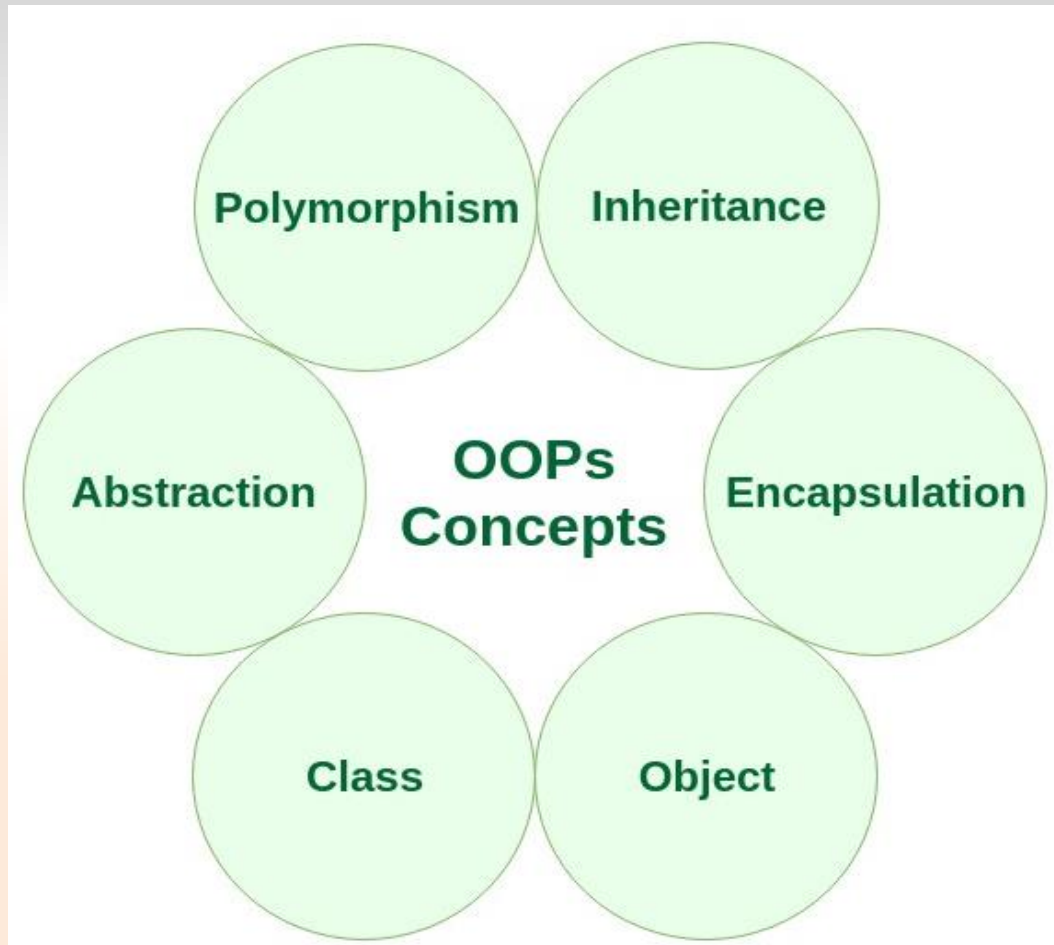
# Continue...

- **Class and objects:**

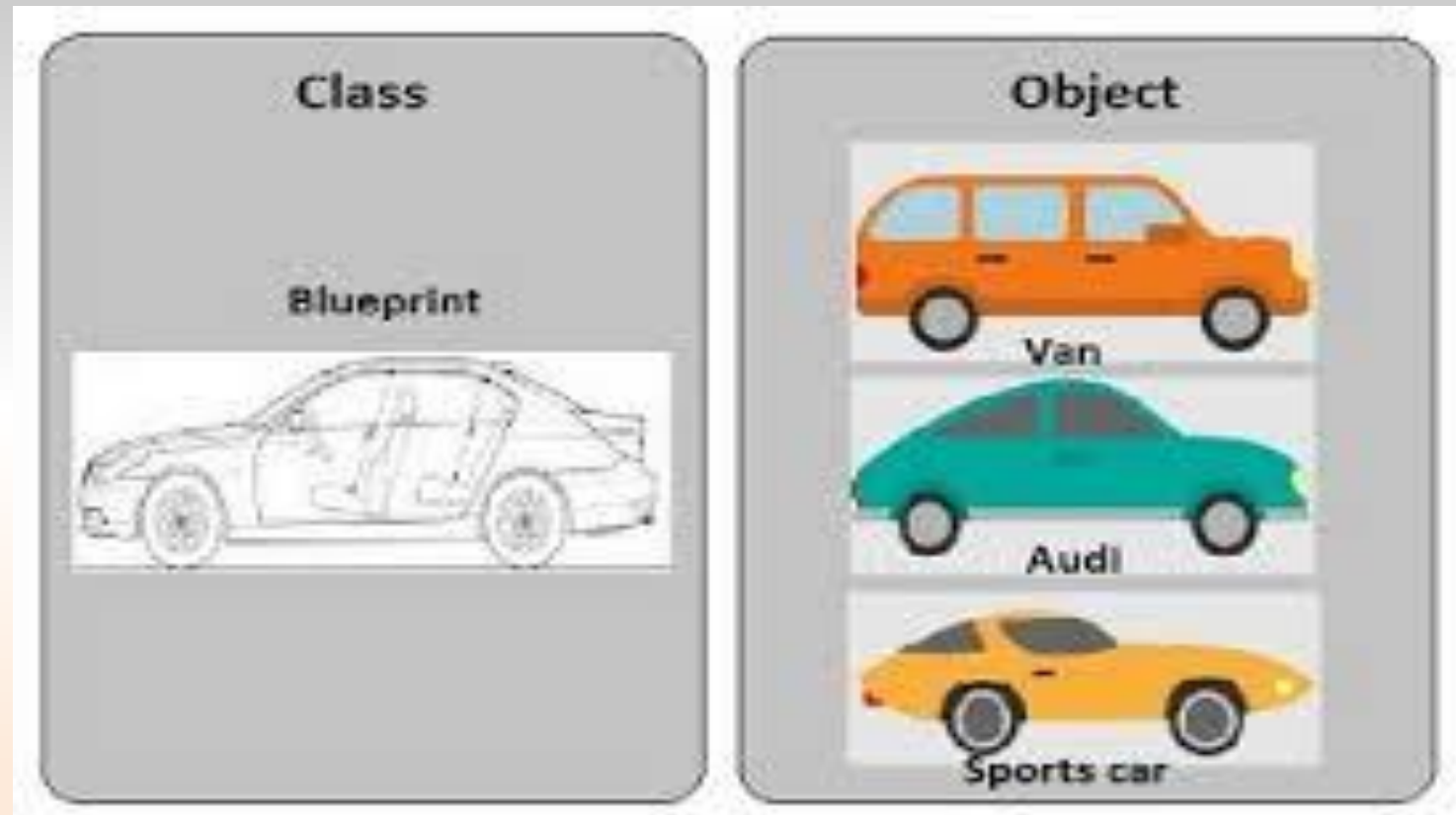
- Define class with instance variables and methods
- Object creation of class
- Accessing member of class
- Argument passing
- Constructors
- Method overloading
- static data, static methods, static blocks
- this keyword

# What is OOP?

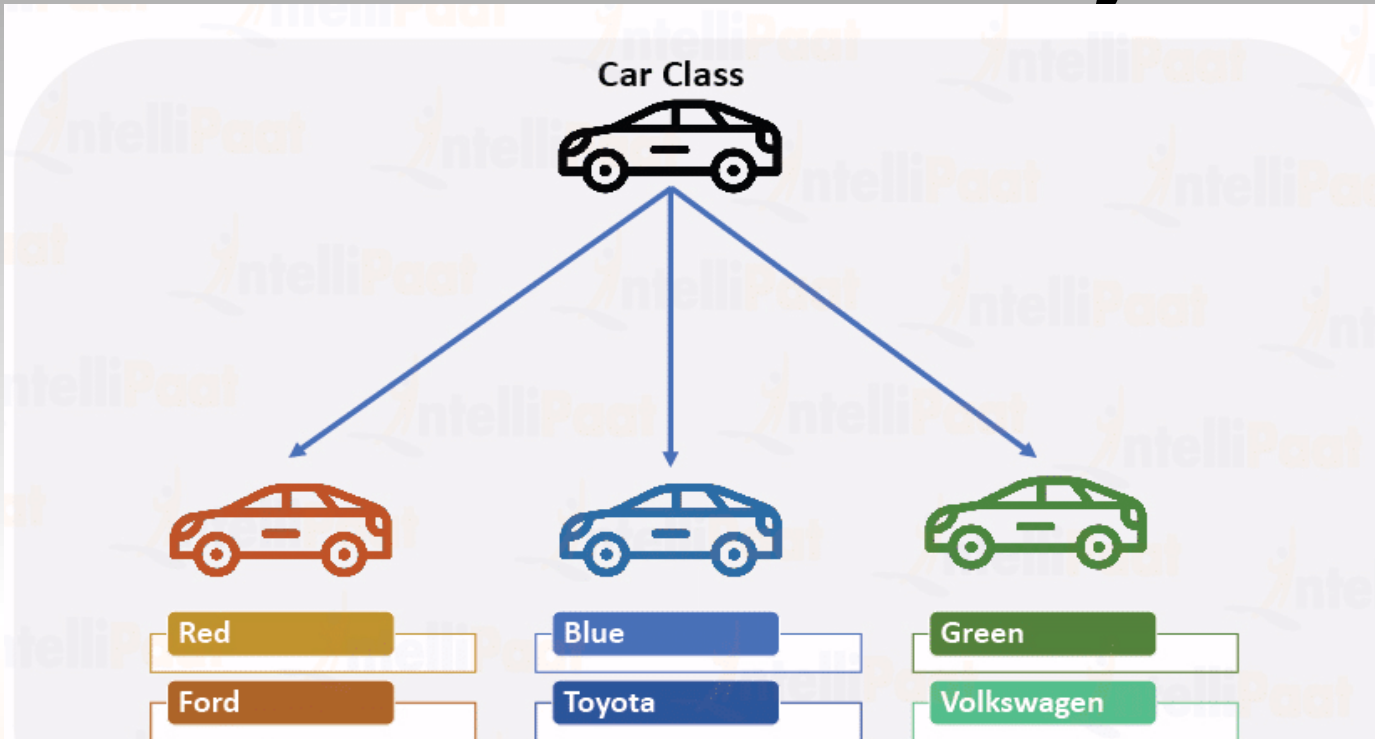
- OOP : Object Oriented Programming
- Uses **objects** in programming.



# Class and Object

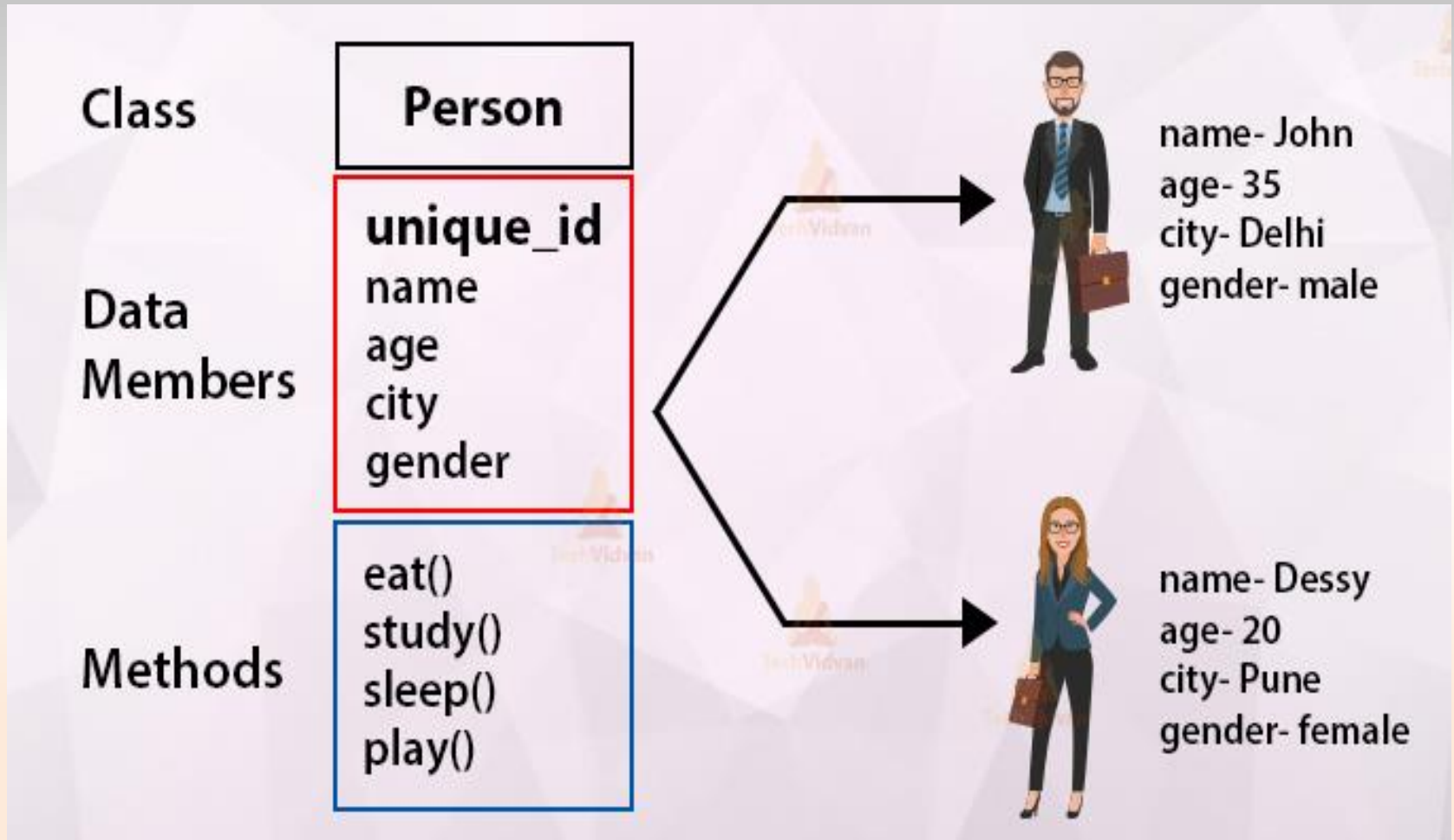


# Class and Objects

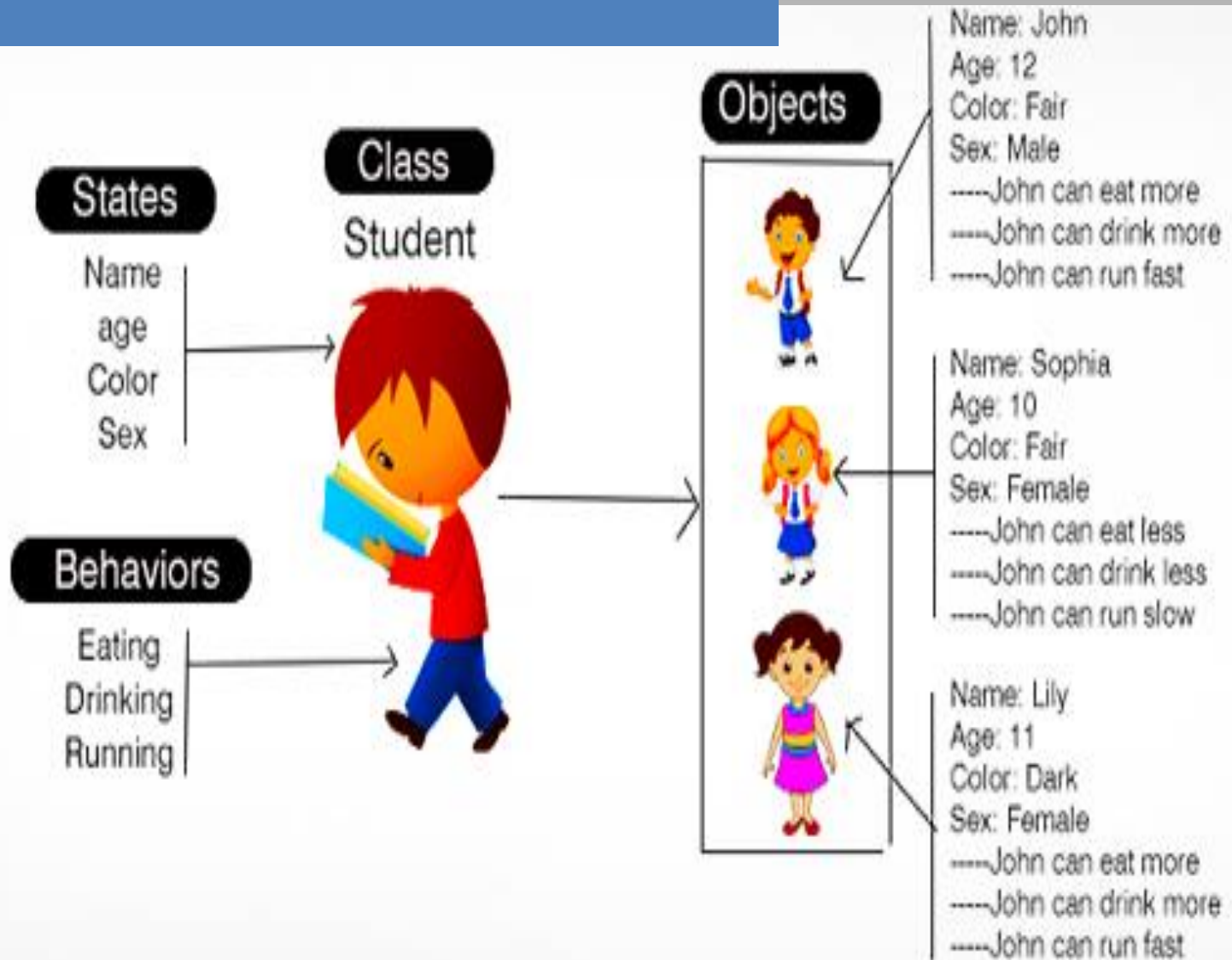


- ✓ **Class is user defined data type.**
- ✓ **An object is an instance of a class.**
- ✓ A class is a template or blueprint from which objects are created.
- ✓ An object is the instance(result) of a class.

# Class and Objects

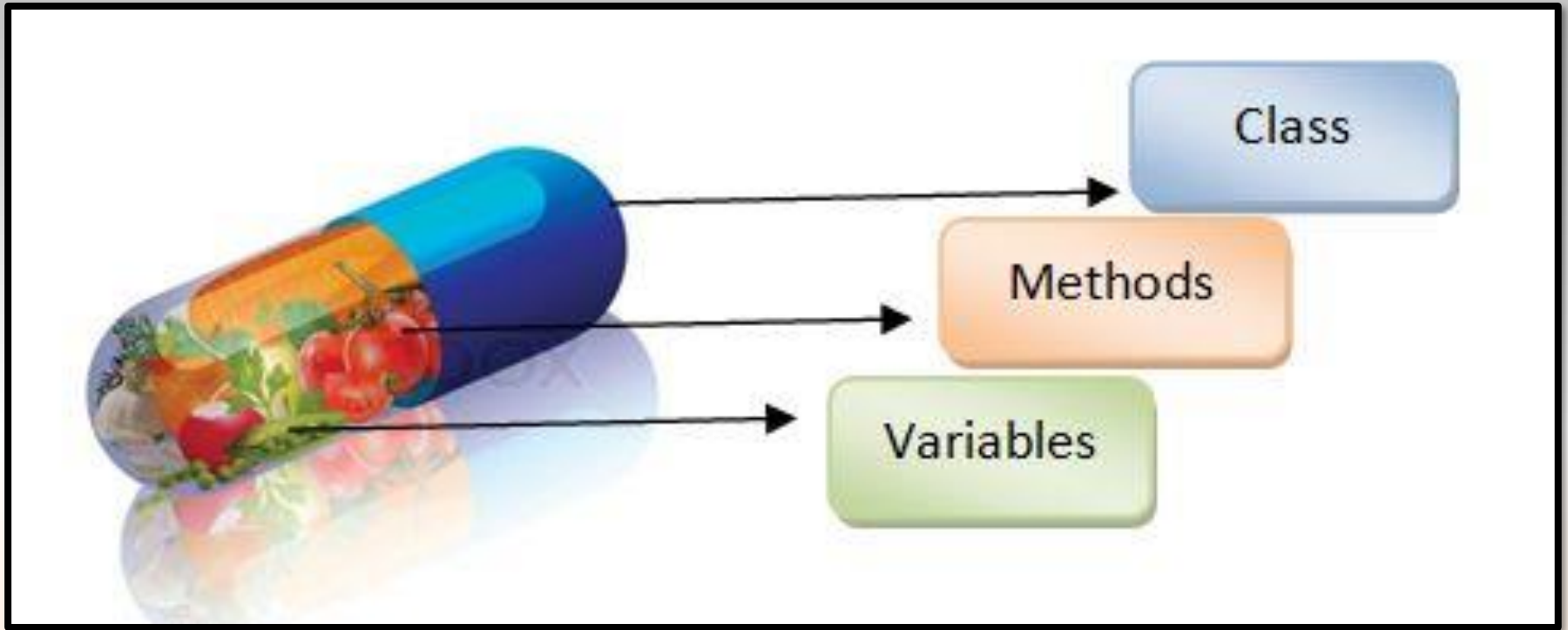


# Class and Objects



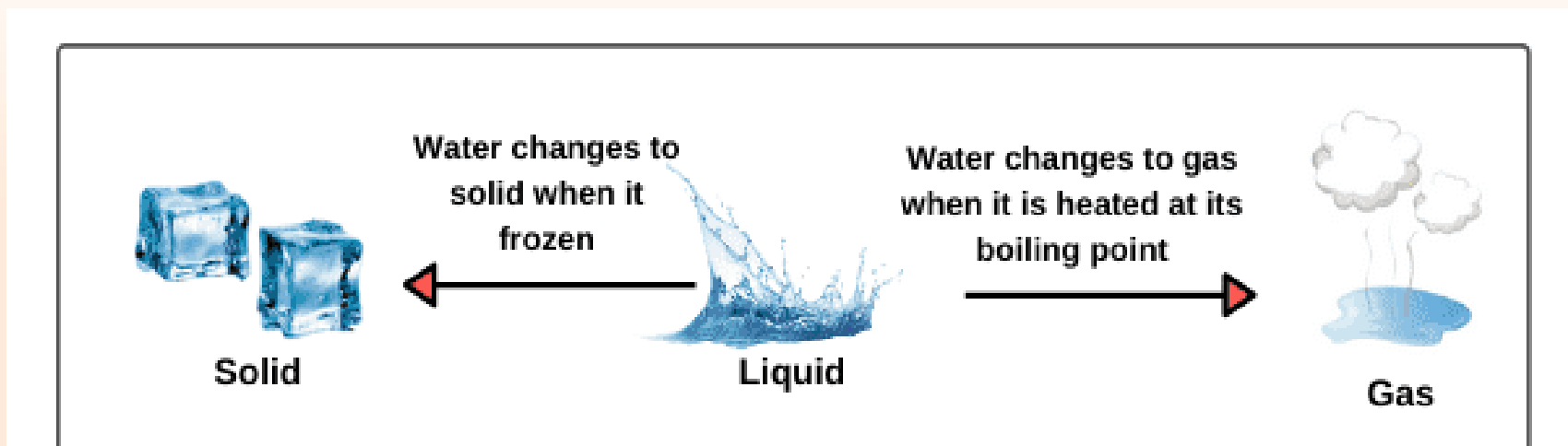
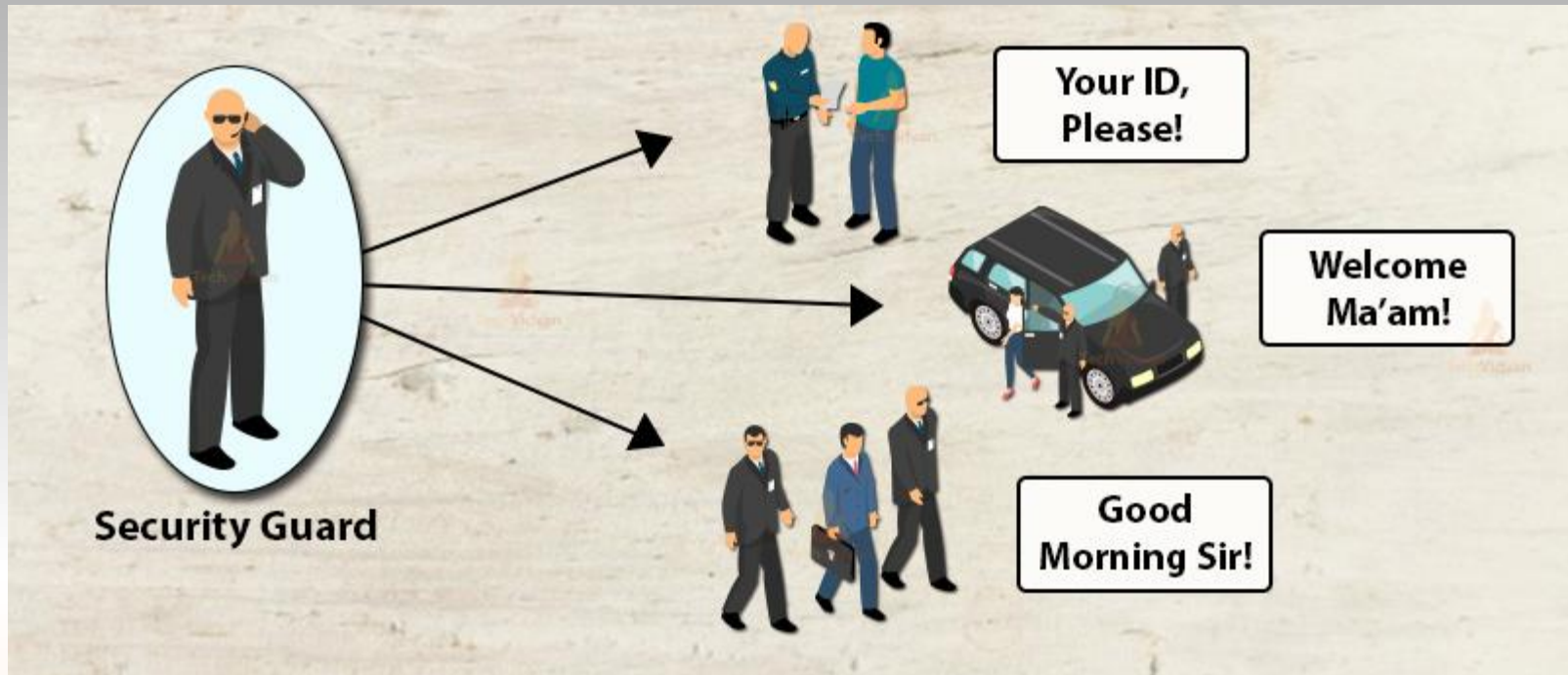


# Encapsulation



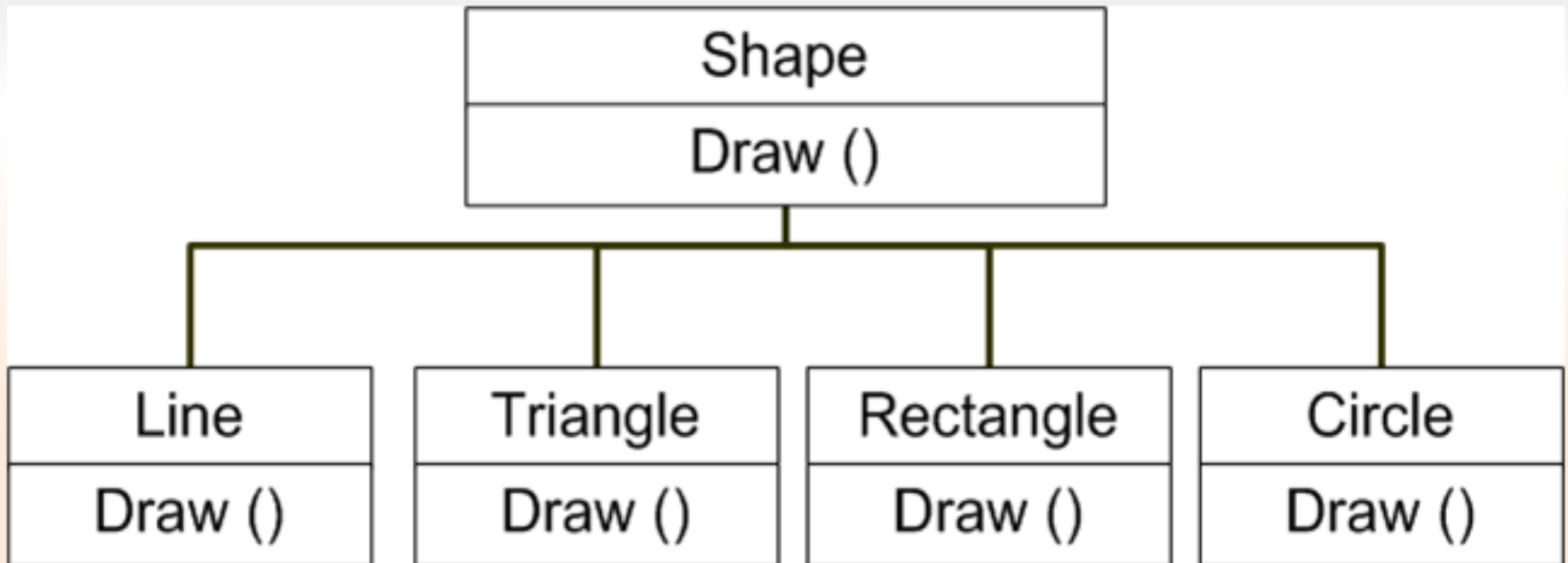
Encapsulation in Java is a **mechanism of wrapping the data (variables) and code acting on the data (methods)** together as a single unit.

# Polymorphism

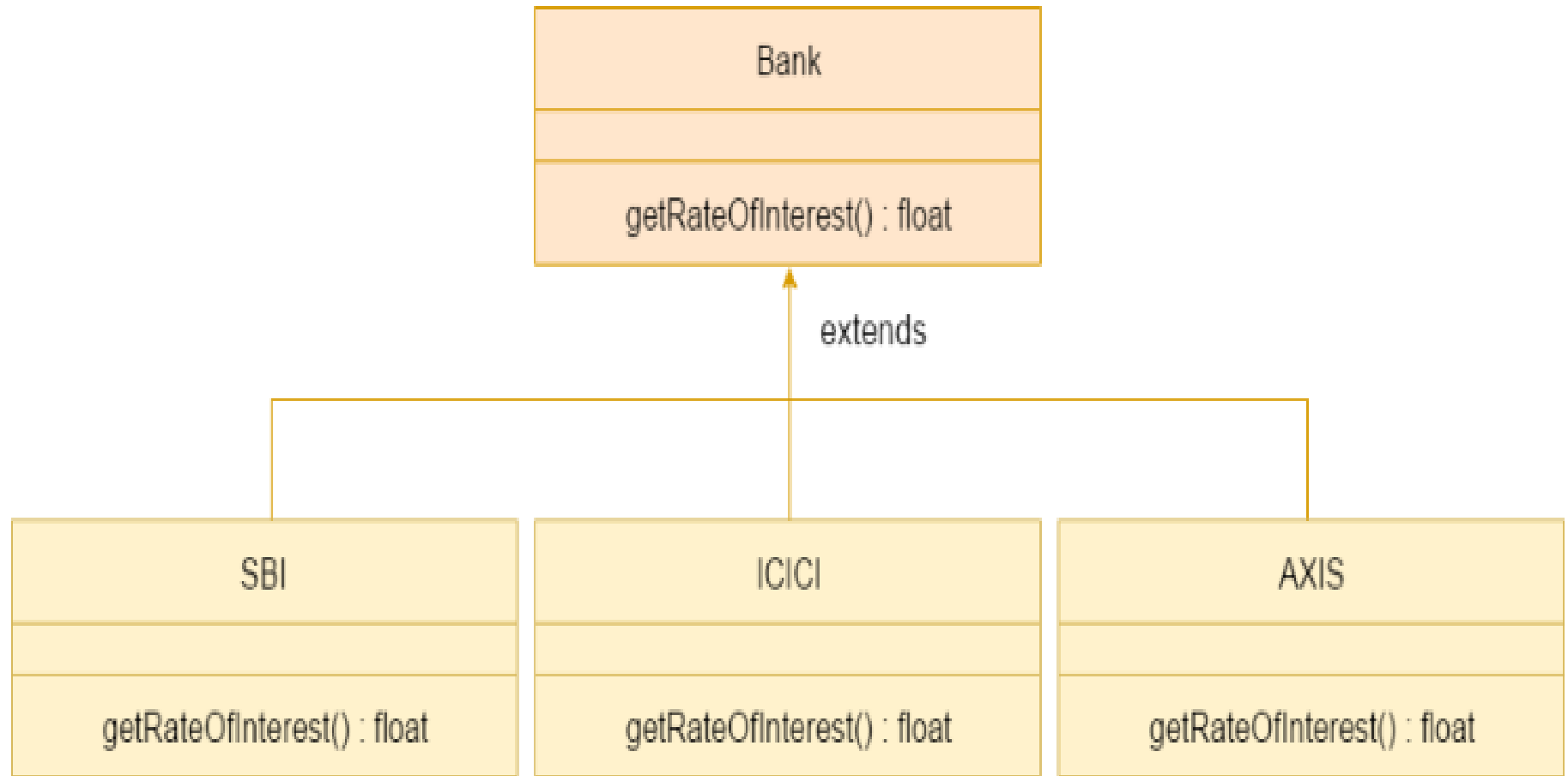


# Polymorphism

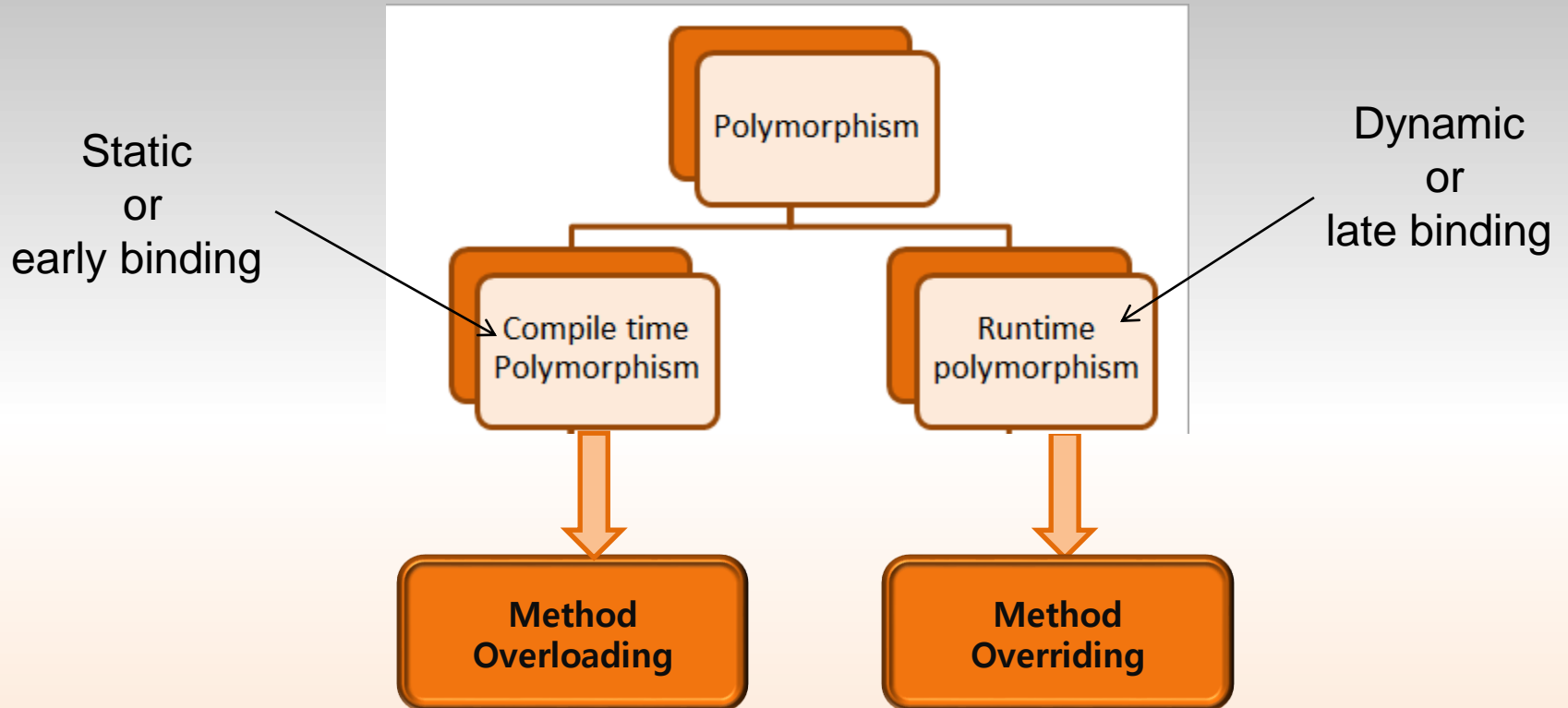
- ✓ Polymorphism means more than one forms
- ✓ One name many forms – Polymorphism
- ✓ **Polymorphism** is the ability of an object to take on many forms.



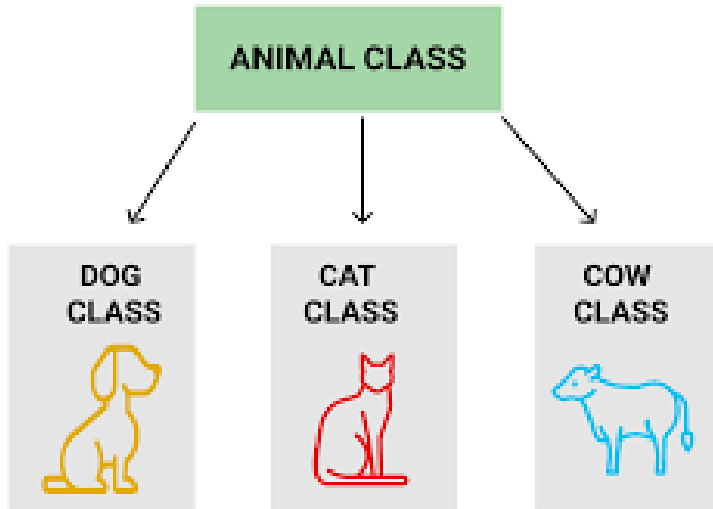
# Polymorphism



# Types of polymorphism



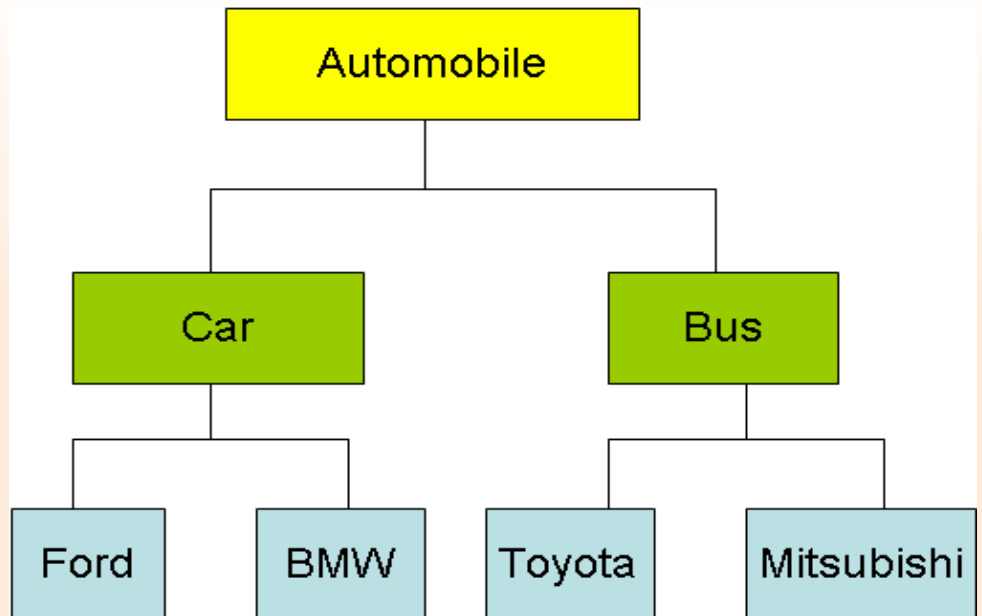
# Inheritance



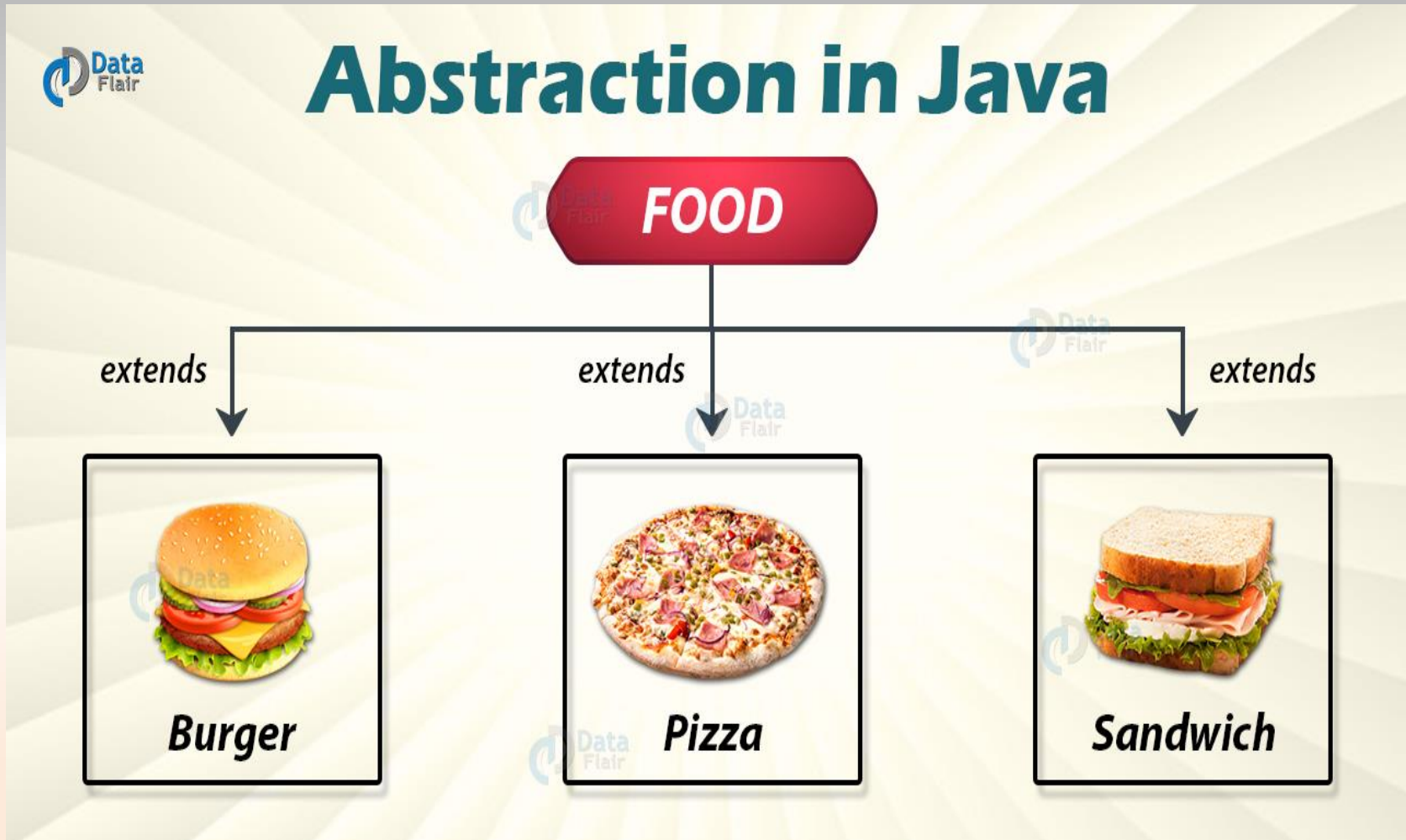
**Example 1**

**Inheritance in Java** is a Mechanism in which one object acquires all the properties and behaviors of a parent object.

**Example 2**



# Abstraction



- Abstraction is the process of identifying only the relevant details and ignoring the non-essential details.

# POP v/s OOP

Procedure Oriented Programming (POP)	Object Oriented Programming (OOP)
1. Large programs are divided into smaller parts known as function.	1. Large programs are divided into smaller parts know as objects.
2. Main focus is on functions.	2. Main focus is on data.
3. Most of the functions share global data.	3. Data structure characterizes objects.
4. Data moves openly among functions.	4. Data and functions that operate on them are tied into class.
5. Any function can access and transform data.	5. Only those function who are member of class can access and transform data.
6. Functions are unable to represent the entities of problem exactly.	6. Objects can easily represent the entities of problem exactly.
7. Follows top down approach in program design.	7. Follows bottom up approach in program design.



# Java v/s C++

Java	C++
Doesn't support Pointer concept	Support pointer concept.
Doesn't support multiple inheritances	Support multiple inheritance
Does not include structures or unions	It have structure and union concept
Includes automatic garbage collection	Requires explicit memory management
Java has method overloading, but no operator overloading	C++ supports both method overloading and operator overloading
Platform independent programming language	Platform dependent programming language
It is mainly used for design <b>web based application, desktop application</b>	It is used for design only <b>desktop application</b> like OS, Compiler etc.
Java uses <b>compiler</b> and <b>interpreter</b>	C++ use only <b>Compiler</b>

# Features of Java



The infographic consists of a central dark blue circle with the text 'Features of Java'. Surrounding this central circle are 12 smaller, overlapping circles, each containing a number and a feature name. The circles are arranged in a ring, with colors alternating between shades of blue, orange, green, and pink. The features are numbered 1 through 12 in a clockwise direction starting from the top right.

1  
Object  
Oriented

2  
Simple

3  
Secured

4  
Platform  
Independent

5  
Robust

6  
Portable

7  
Architecture  
Neutral

8  
Dynamic

9  
Interpreted

10  
High  
Performance

11  
Multithreaded

12  
Distributed

## Object Oriented

- ✓ Everything in Java is an object.
- ✓ Object-oriented means we organize our software as a combination of different types of objects that deals with both data and behaviour.

## Simple

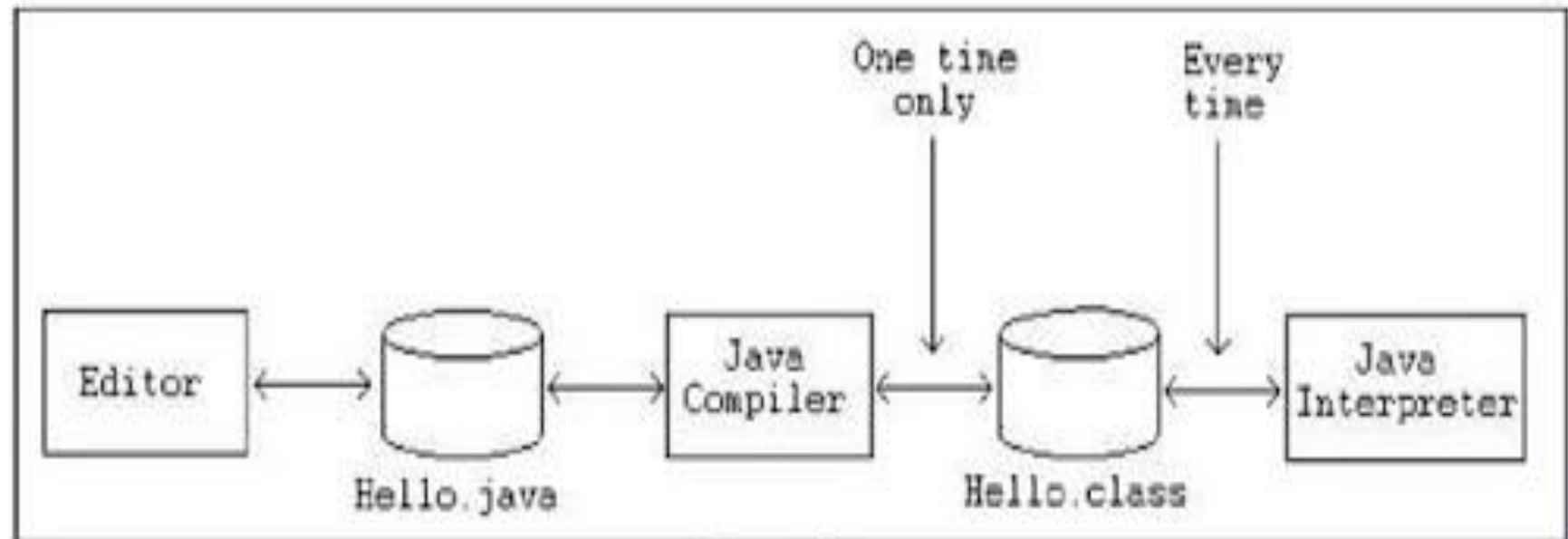
- ✓ Java syntax is based on C++ so easy for who know C++.
- ✓ Many complicated and rarely-used features are removed
- ✓ Ex. explicit pointers, operator overloading, etc.
- ✓ There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

# Platform Independent

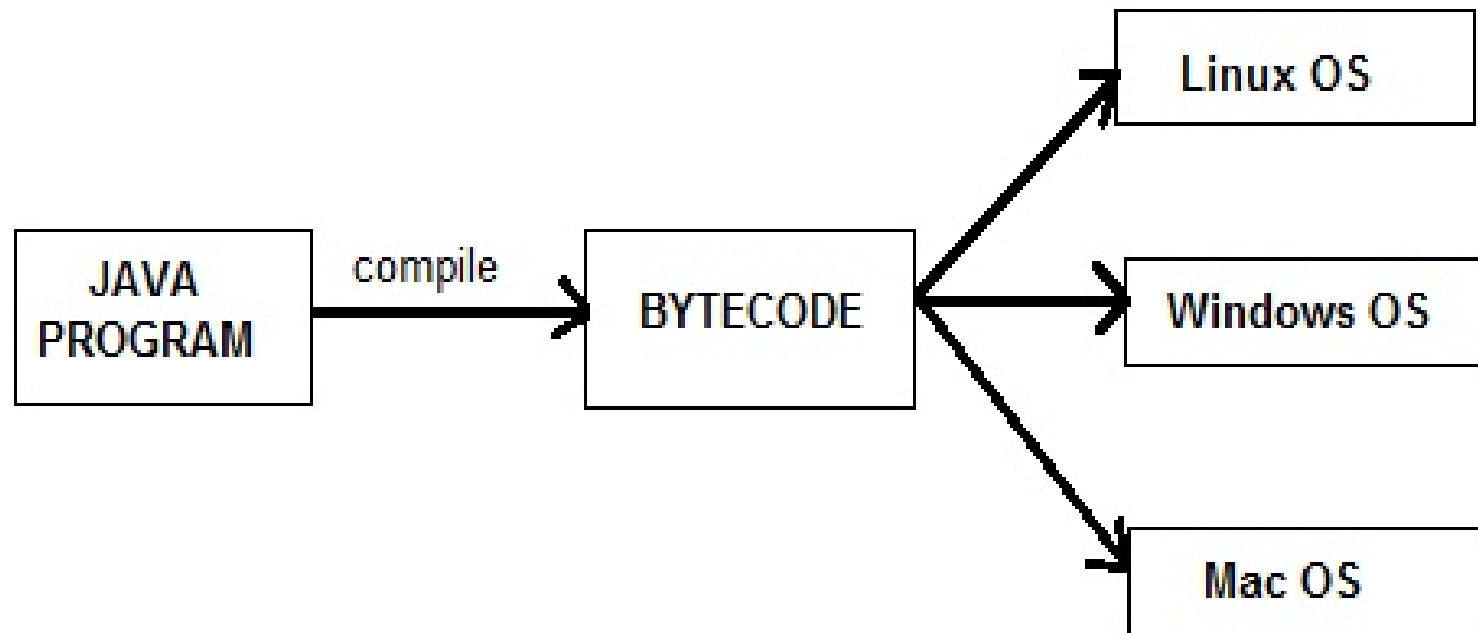
- ✓ A platform is the hardware or software environment in which a program runs.
- ✓ There are two types of platforms
  - ✓ Software-based
  - ✓ Hardware-based.
- ✓ Java provides software-based platform.
- ✓ It has two components:
  - ✓ Runtime Environment
  - ✓ API(Application Programming Interface)
- ✓ Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc.
- ✓ Java code is compiled by the compiler and converted into byte code.
- ✓ This bytecode is a platform independent code because it can be run on multiple platforms  
i.e. Write Once and Run Anywhere(WORA).

# Phases of a Java Program Development

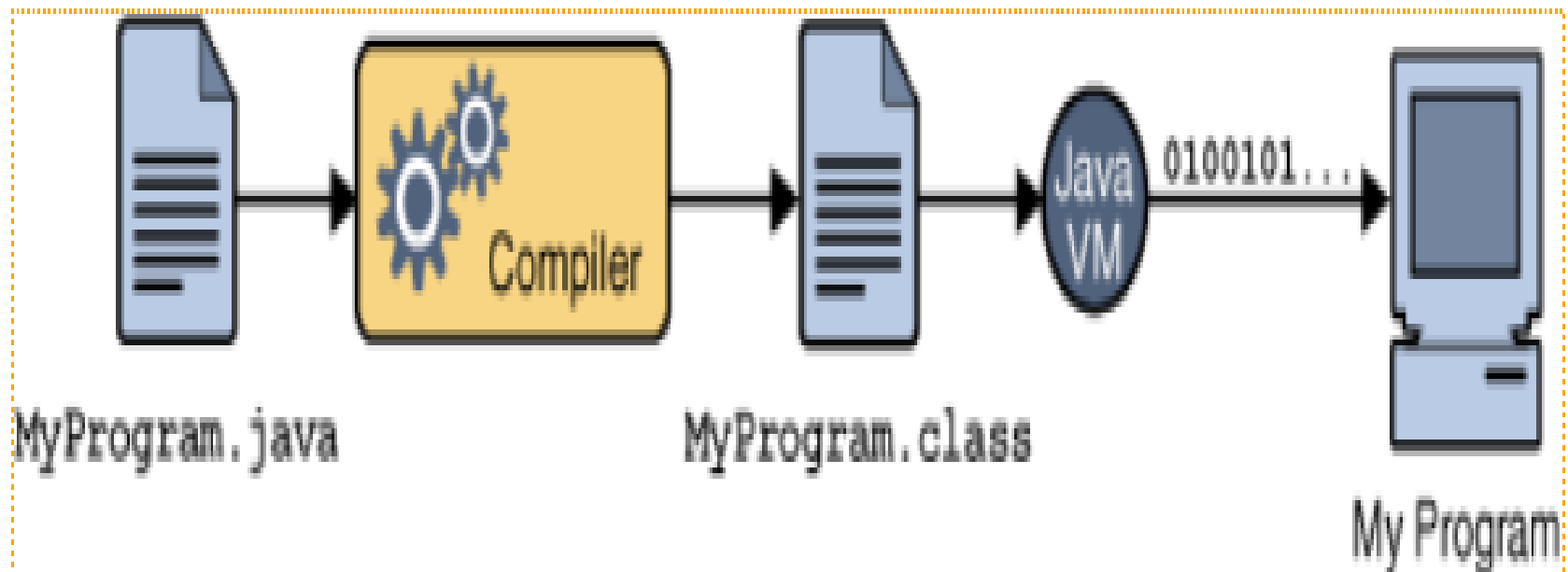
- The following figure describes the process of compiling and executing a Java program



# Platform independent

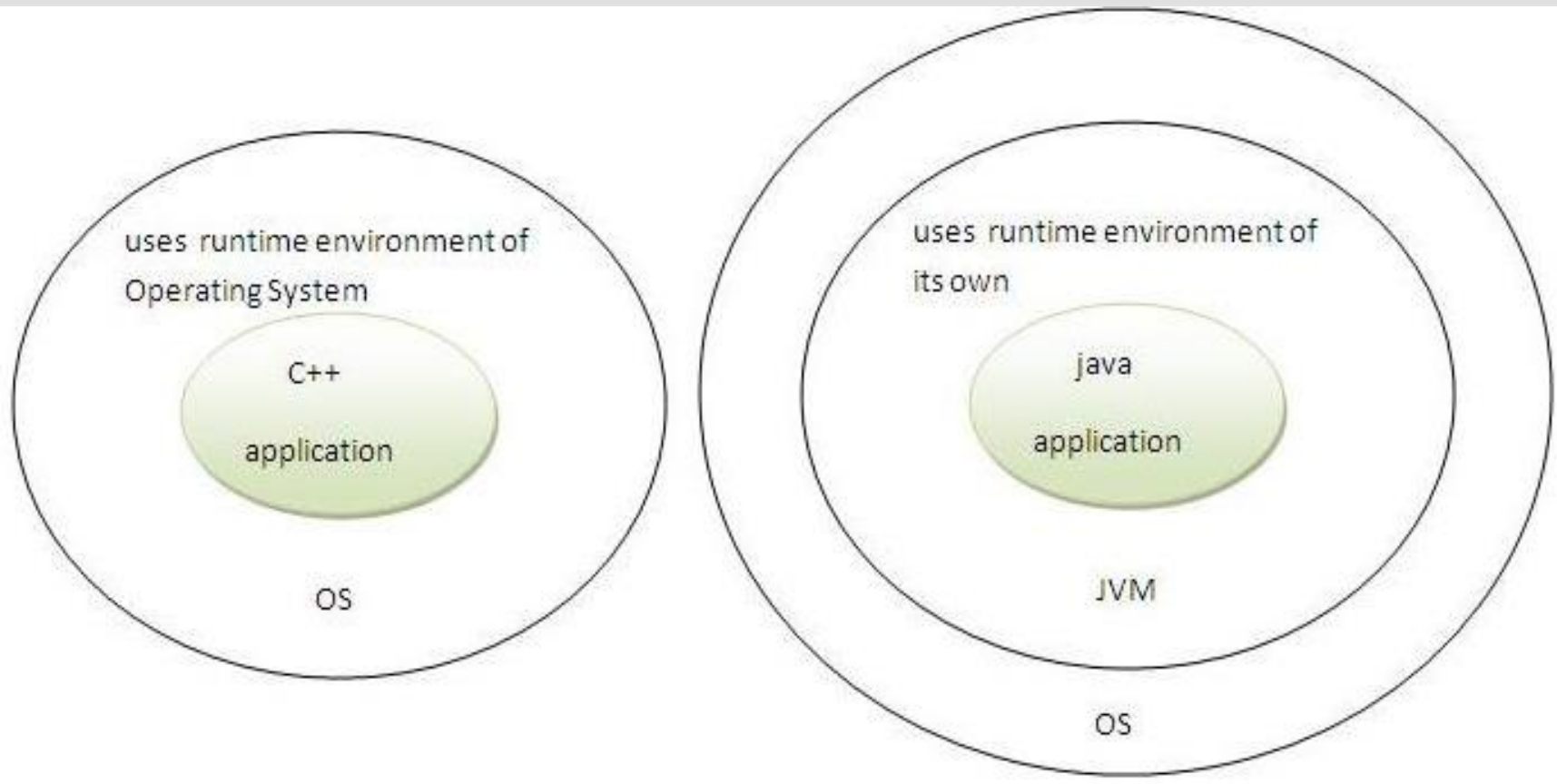


# WORA(Write Once Run Anywhere)



# Secured

- ✓ Java is secured because:
  - ✓ No explicit pointer
  - ✓ Programs run inside virtual machine sandbox.



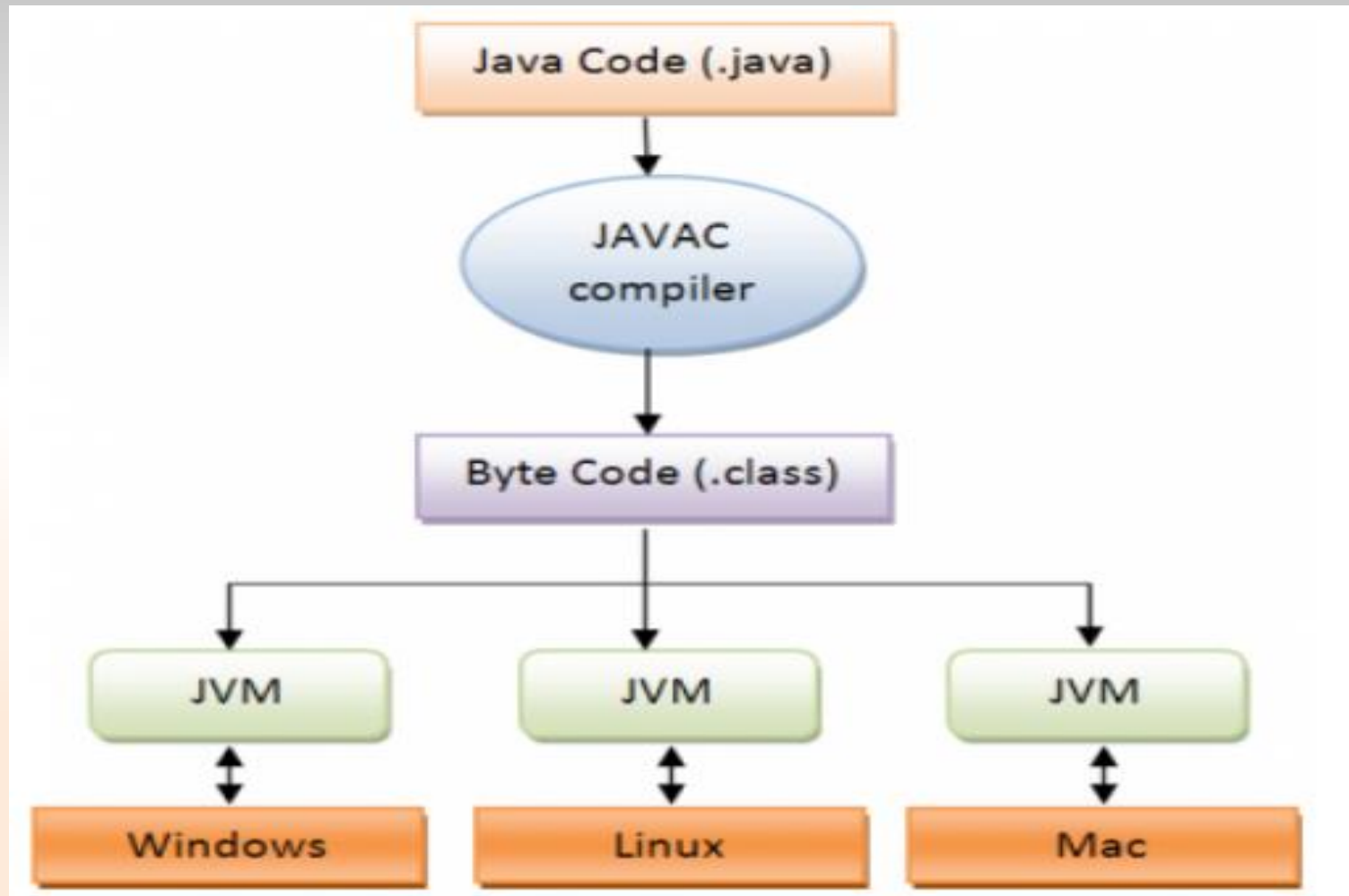


# Robust

- ✓ Robust simply means strong.
- ✓ Java uses strong memory management.
- ✓ There are lack of pointers that avoids security problem.
- ✓ There is automatic garbage collection in Java.
- ✓ There is exception handling and type checking mechanism in java.

# Java is architecture-neutral

## JAVA PROGRAM EXECUTION



# Other....

## ✓ **Portable**

- We may carry the java bytecode to any platform.

## ✓ **High-performance**

- Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

## ✓ **Distributed**

- We can create distributed applications in java.
- RMI and EJB are used for creating distributed applications.
- We may access files by calling the methods from any machine on the internet.

## ✓ **Multi-threaded**

- A thread is like a separate program, executing concurrently.
- The main advantage of multi-threading is that it shares the same memory.
- Threads are important for multi-media, Web applications etc.

# Java Environment

✓ Java environments includes :

1. JDK
2. JSL

- **Java Development Kit (JDK):**

- includes Development tools

- **Java Standard Library (JSL):**

- Also known as **Application Programming Interface (API)**

- It includes classes and methods.

✓ JDK includes tools like java compiler(javac),  
java interpreter(java)

✓ API includes hundreds of classes and methods grouped into  
several packages according to their functionality.

# JDK tools

Tool	Description
<b>javac</b>	Java compiler
<b>java</b>	Java interpreter
<b>javadoc</b>	API documentation generator
<b>appletviewer</b>	Run and debug applets without a web browser
<b>jar</b>	Manage Java Archive (JAR) files
<b>jdb</b>	Java Debugger.
<b>javah</b>	C header and stub generator. Used to write native methods.
<b>javap</b>	Class file disassembler

# JSL / API

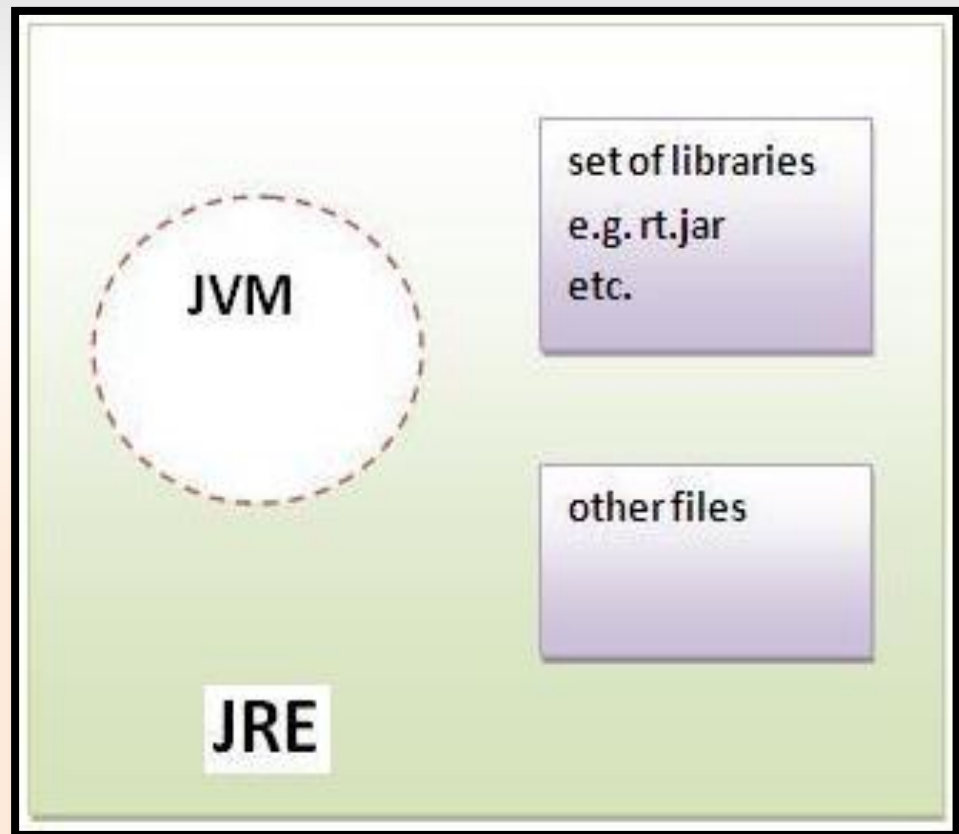
- Packages :
  - java.lang
  - java.util
  - java.applet
  - java.awt
  - java.io
  - java.awt.event etc.....

# JVM

- ✓ JVM (Java Virtual Machine) is an abstract machine.
- ✓ It is a specification that provides runtime environment in which Java byte code can be executed.
- ✓ JVMs are available for many hardware and software platforms.
- ✓ JVM, JRE and JDK are platform dependent because configuration of each OS differs.
- ✓ But, Java is platform independent.
- ✓ The JVM performs following main tasks:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

# JRE (Java Runtime Environment)

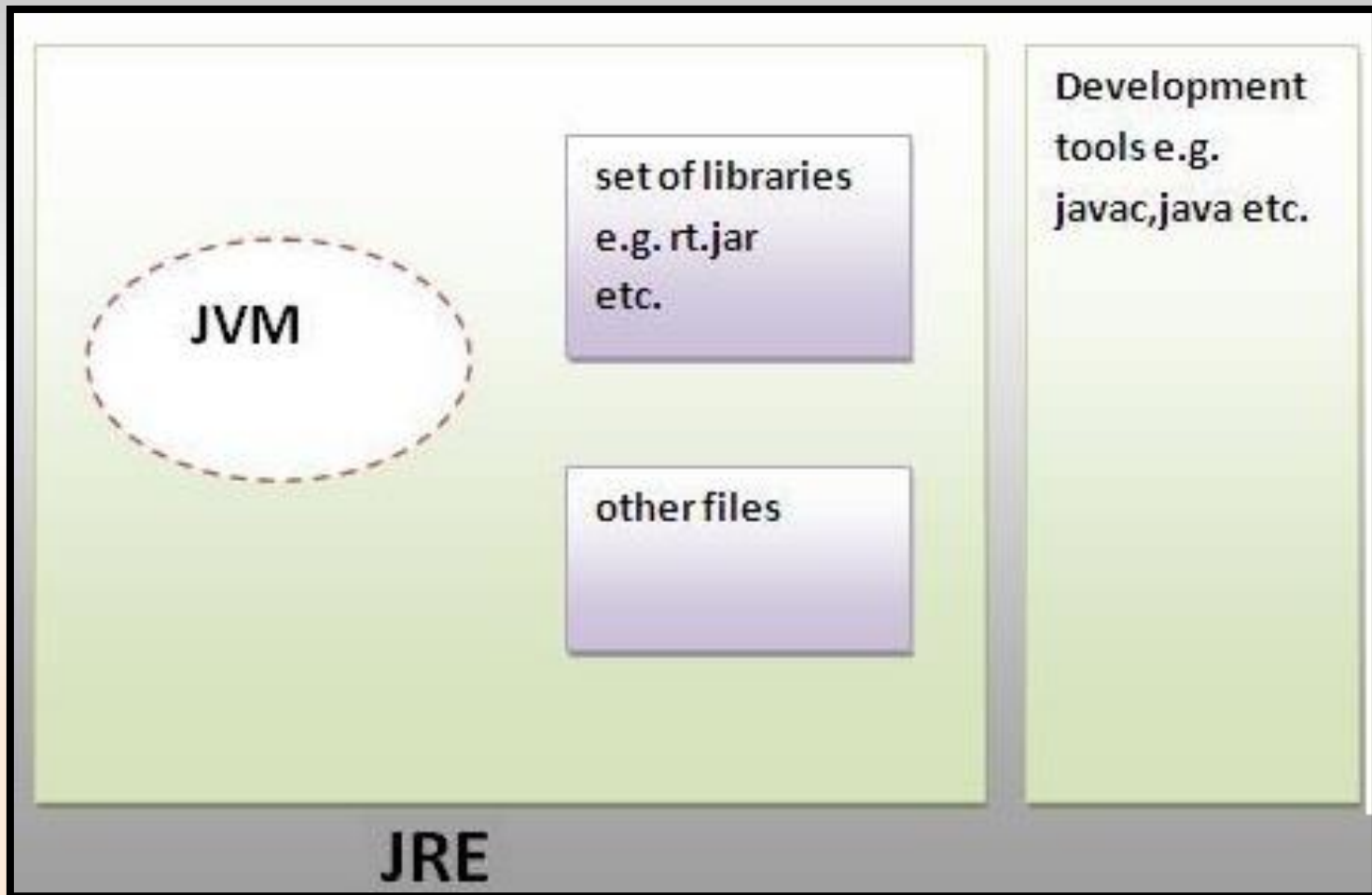
- ✓ It is used to provide runtime environment.
- ✓ It is the implementation of JVM.
- ✓ It contains set of libraries + other files that JVM uses at runtime.



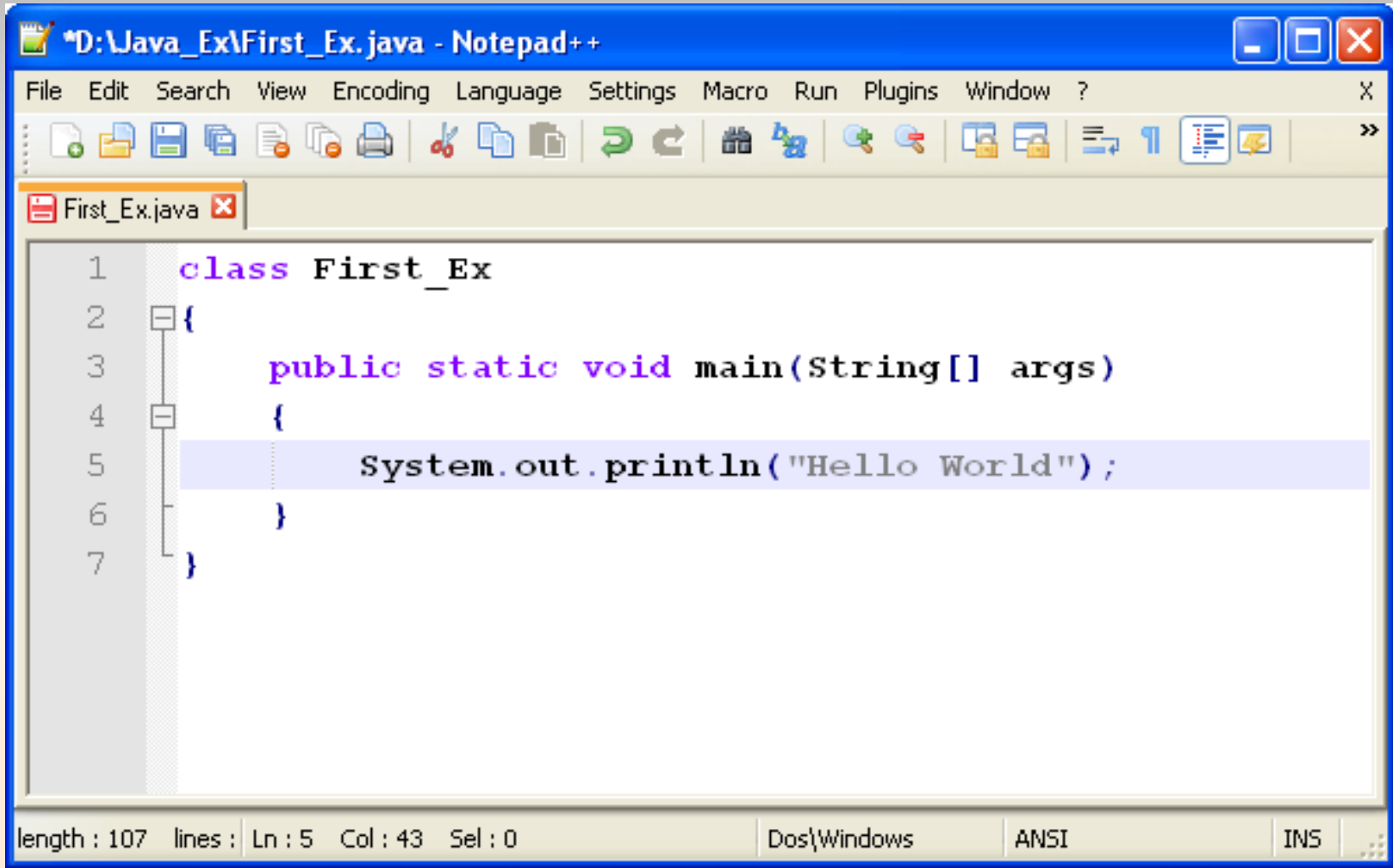


# JDK (Java Development Kit)

- ✓ It contains JRE + development tools.



# Hello world Program in java



The image shows a Notepad++ window titled "D:\Java\_Ex\First\_Ex.java - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The editor displays the following Java code:

```
1  class First_Ex
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("Hello World");
6      }
7  }
```

The status bar at the bottom shows "length : 107", "lines :", "Ln : 5", "Col : 43", "Sel : 0", "Dos\Windows", "ANSI", and "INS".

# Execution of Hello world Program



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" and standard window control buttons (minimize, maximize, close). The main area is black with white text. The text shows the following sequence of commands and output:

```
D:\Java_Ex>javac First_Ex.java
D:\Java_Ex>java First_Ex
Hello World
D:\Java_Ex>_
```

The window has a scroll bar on the right side and a status bar at the bottom.

# Example....

- **main( )** is the entry point of your Java program.  
The main method has to have this exact signature in order to be able to run your program.
- **public:** means that anyone can access it.
- **static:** means that you can run this method without creating an instance of that class.
- **void:** means that this method doesn't return any value.
- **main** is the name of the method.
- **System:** is a pre-defined class that Java provides us and it holds some useful methods and variables.
- **out:** is a static variable within System that represents the output of your program.
- **println:** is a method of out that can be used to print given text and point to next line.
- **print:** is a method of out that can be used to print a given text.

# Program structure

## Syntax:

- [Definition section]
- [package declarations if needed]
- [import statements by default java. lang ]
- [class declaration]
- :
- [class declaration]

## Example :

```
// First Example of java Program
package abc;
import java.lang.*;

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

# History of Java

- Founder : James Gosling - Sun Microsystems
- Co founder – Vinod Khosla
- **Oak - Java, May 20, 1995, Sun World**
- JDK Evolutions
  - JDK 1.0 (January 23, 1996)
  - JDK 1.1 (February 19, 1997)
  - J2SE 1.2 (December 8, 1998)
  - J2SE 1.3 (May 8, 2000)
  - J2SE 1.4 (February 6, 2002)
  - J2SE 5.0 (September 30, 2004)
  - Java SE 6 (December 11, 2006)
  - Java SE 7 (July 28, 2011)
  - Java SE 8 (March 18, 2014)
  - Java SE 9 (May be released in July 2017)
  - Java SE 13.0.1 (released on September 2019)
  - **Java 17 is the latest released Java version.(September 2021)**

# Language Building Blocks(Tokens)

- ✓ Comments
- ✓ Keywords
- ✓ Identifiers
- ✓ Literals
- ✓ White spaces
- ✓ Separator

# Comments

## ✓ **Block comments**

- Can span several lines
- Begin with `/*`
- End with `*/`
- Compiler ignores all text between `/*` and `*/`

## ✓ **Line comments**

- Start with `//`
- Compiler ignores text from `//` to end of line



# Keywords

- ✓ Keywords or Reserved words are words that have a specific meaning to the compiler and cannot be used as identifiers (e.g., variable names, function names, class names).
- ✓ For example, when the compiler sees the word class, it understands that the word after class is the name for the class.
- ✓ The list of reserved words in Java is provided below.

abstract	else	int	super
assert	enum	interface	switch
boolean	extends	long	synchronized
break	false	native	this
byte	final	new	throw
case	finally	null	throws
catch	float	package	transient
char	for	private	true
class	goto	protected	try
const	if	public	void
default	implements	return	volatile
do	import	short	while
double	instanceof	static	continue

# Identifiers - symbolic names

- ✓ Identifiers are used to name classes, variables, and methods
- ✓ Identifier Rules:
  - Must start with a "Java letter"
    - A - Z, a - z, \_, \$...
  - Can contain essentially any number of Java letters and digits, but no spaces
  - Case sensitive!!
    - Number1 and number1 are different!
  - Cannot be keywords or reserved words

# Literals

- ✓ All values that we write in a program are literals.
- ✓ Each belongs to one of Java's four primitive types (int, double, boolean, char) or belongs to the special reference type String.
- ✓ All primitive type names are keywords in Java

Literal	Type
1	int
3.14	double
True	boolean
'3'	char ('P' and '+' are char too)
"Std ID"	String
Null	any reference type

# Literals

## ✓ **int, short, byte:**

- Optional initial sign (+ or -) followed by digits 0 – 9 in any combination.

## ✓ **long:**

- Optional initial sign (+ or -) followed by digits 0–9 in any combination, terminated with an L or l.
- Note: Use the capital L because the lowercase l can be confused with the number 1.

# Floating-Point Literals

- ✓ **float:**

- Optional initial sign (+ or -) followed by a floating-point number in fixed or scientific format, terminated by an F or f.

- ✓ **double:**

- Optional initial sign (+ or -) followed by a floating-point number in fixed or scientific format.

- **Note:** Commas, dollar signs, and percent signs (%) can not be used in integer or floating-point literals

# char and boolean Literals

- ✓ **char:**
  - Any printable character enclosed in single quotes
- ✓ **boolean:**
  - true or false

# String Literals

- ✓ **String is actually a class**, not a basic data type.
- ✓ String variables are objects
- ✓ String literal: text contained within double quotes.
- ✓ Example of String literals:
  - "Hello"
  - "Hello world"
  - "The value of x is "



# Building Blocks - White Space

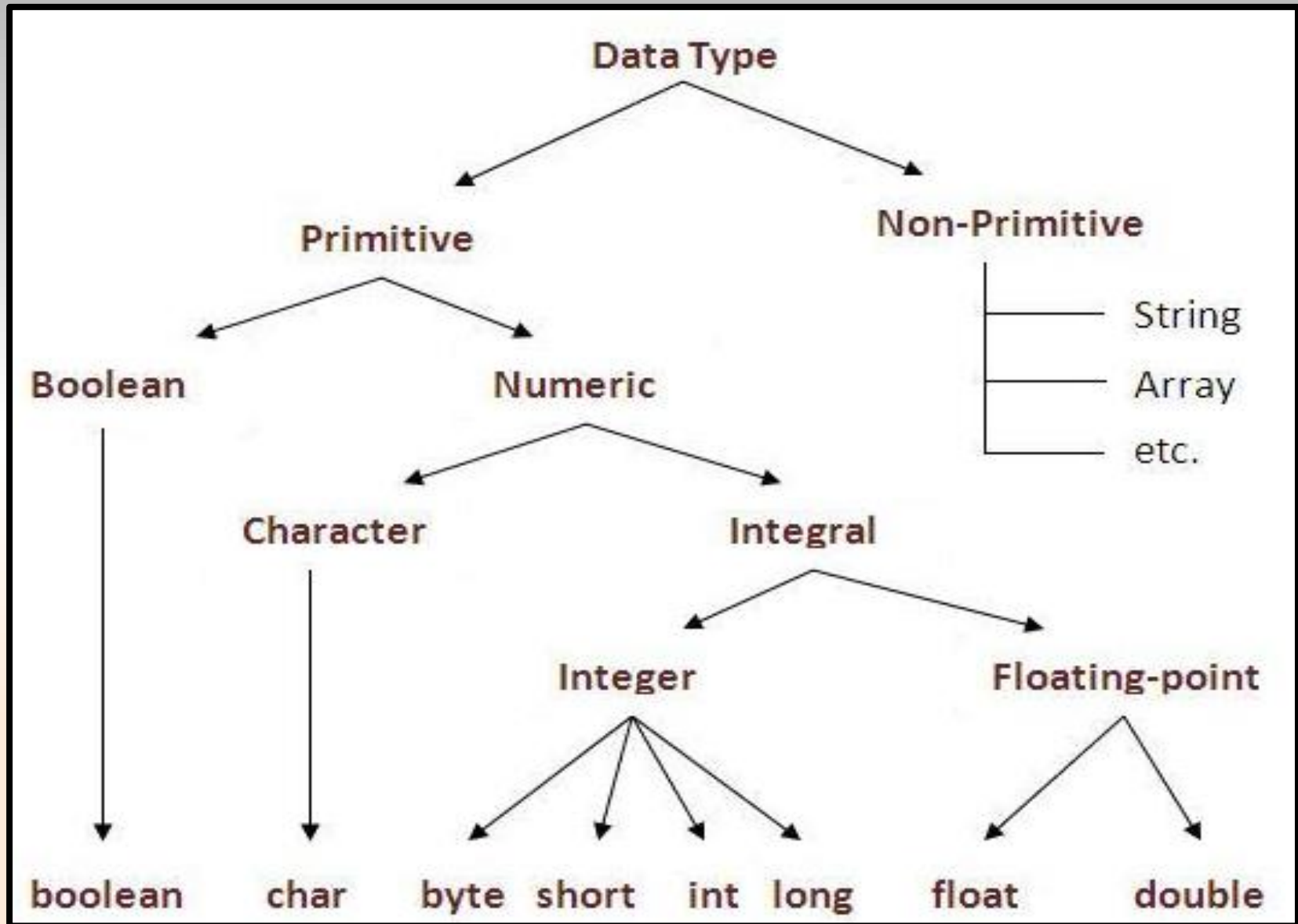
- ✓ Space, tab, newline are white space characters
- ✓ At least one white space character is required between a keyword and identifier
- ✓ Any amount of white space characters are permitted between identifiers, keywords, operators, and literals.

# Separators

- ✓ There are exactly 7 single character (separators) in Java

Separator	Use
;	Terminate statement
,	Declare more then one variable or pass argument in UD F
.	Used to access variable or method of a class.
:	Used to specify a label.
()	Used in method ,if condition and loops.
{ }	Enclose statement of methods, class or any block.
[ ]	Used in array.

# Data Types



# DataType

```
graph TD;
    DT[DataType] --> Integer;
    DT --> Floating;
    DT --> Character;
    DT --> Boolean;
    Integer --> byte1["byte(1)"];
    Integer --> short2["short(2)"];
    Integer --> int4["int(4)"];
    Integer --> long8["long(8)"];
    Floating --> float4["float(4)"];
    Floating --> double8["double(8)"];
    Character --> char2["char(2)"];
    Boolean --> boolean1["boolean(1)"];
```

## Integer

byte(1)

short(2)

int(4)

long(8)

## Floating

float(4)

double(8)

## Character

char(2)

## Boolean

boolean(1)

# Type Casting...

## ✓ Implicit casting

`double d = 3;` (type widening)

## ✓ Explicit casting

`int i = (int)3.0;` (type narrowing)

# Identifiers

- ✓ An identifier is a sequence of characters that consist of:
  - letters
  - digits
  - underscores (\_)
  - dollar signs (\$)
- ✓ An identifier must start with a letter, an underscore (\_), or a dollar sign (\$).
- ✓ It cannot start with a digit.
- ✓ An identifier cannot be a keyword.
- ✓ An identifier cannot be true, false, or null.
- ✓ An identifier can be of any length.

# Variables

## Declaring & Assignment of variable

- `int x;`
- `double radius;`
- `char a;     // Declare a to be a char;`
- `x = 1;         // Assign 1 to x;`
- `radius = 1.0;   // Assign 1.0 to radius;`
- `a = 'A';       // Assign 'A' to a;`

# Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`
- `float f = 1.4f;`

**OR**

- `float f = 1.4F;`



# Types of Variables

- **Local variable:**

- ✓ A variable that is declared inside the method is called local variable.

- **Instance Variable:**

- ✓ A variable that is declared inside the class but outside the method is called instance variable .
- ✓ It is not declared as static.

- **Static variable:**

- ✓ A variable that is declared as static is called static variable.

# Constants with **final** keyword

- **Syntax:**
  - final datatype CONSTANTNAME= VALUE;
- **Example:**
  - final double PI = 3.14159;
  - final int SIZE = 3;

## Input from User (Scanner Class)

[illegible]

# Input from User (Using Console)

```
import java.io.*;
class Test
{   public static void main(String[ ] args)
    {   Console co=System.console();
        System.out.println("Enter a Rollno:");
        String rno=co.readLine();
        int roll = Integer.parseInt(rno); //Convert to integer
        System.out.println("Enter a Name:");
        String name=co.readLine();
        System.out.println("Enter Fees:");
        String f=co.readLine();
        double fees=Double.parseDouble(f);
        System.out.println("Roll no:"+roll+" Name:"+ name+"
                                Fees:"+fees);
    }
}
```

# **Operators in Java**

# Operators

1. Assignment Operators
2. Increment Decrement Operators
3. Arithmetic Operators
4. Bitwise Operators
5. Relational Operators
6. Logical Operators
7. Ternary Operators (Conditional Operators)
8. Special Operators

# Operator

## Operators

Arithmetic Op. : + - \* / %

Relational Op. : > >= < <= == !=

Logical Op. : && || !

Inc/Dec Op. : ++ --

Bit Op. : & | ^ << >>

Conditional Op. : ?:

Assign Op. : = += -= \*= /= %= &= ^= |= >>= <<=

Instanceof Op. : instanceof

# Arithmetic Operators

Operator	Use	Description
+	op1 + op2	Adds op1 and op2; also used to concatenate strings
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2



# Arithmetic Operators...

- $+$ ,  $-$ ,  $*$ ,  $/$ , and  $\%$
- $5/2$  yields an integer 2.
- $5.0/2$  yields a double value 2.5
- $5 \% 2$  yields 1 (the remainder of the division)
- $5.0 \% 2$  is not defined : modulo is defined only for integers.

# Example: Addition

```
class Addition
{
    public static void main(String[] args)
    {
        // declare variables
        int x, y, z;
        // Specify values of x and y
        x = 2;
        y = 3;
        z = x + y;

        System.out.println("x has a value of " + x);
        System.out.println("y has a value of " + y);
        System.out.println("The sum of x + y is " + z);
    }
}
```

# Example: Division

```
class Division
{
    public static void main ( String[] args)
    {
        //declare variables
        int x, y, z ;
        x = 12;
        y = 4;
        z = x / y ;
        System.out.println("x has a value of " + x);
        System.out.println("y has a value of " + y);
        System.out.println("x divided by y is " + z);
    }
}
```

# Relational Operators

Operator	Meaning	Example
==	Equals	a == b
!=	Not equals	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equals	a >= b
<=	Less than or equals	a <= b

# Logical Operators

Symbol	Condition
&&	AND
	OR
!	NOT

# Assignment/Shorthand Assignment Operators

Operator	Example	Equivalent
=	i=10	
+=	i+=8	i = i+8
-=	f-=8.0	f = f-8.0
*=	i*=8	i = i*8
/=	i/=8	i = i/8
%=	i%=8	i = i%8

# Increment and Decrement Operators

suffix

`x++; // Same as x = x + 1;`

prefix

`++x; // Same as x = x + 1;`

suffix

`x--; // Same as x = x - 1;`

prefix

`--x; // Same as x = x - 1;`

# Example....

```
class Assign_Op
{
    public static void main(String[] args)
    {
        int x = 10; int y = 5; int z = 3;
        System.out.println("x = "+x+", y = "+y+", z = "+z);

        x++; // x = x+1;
        y += x; // y = y+x;
        z *= x; // z = z*x;

        System.out.println("Now x = "+x+", y = "+y+", z = "+z);

        x--; // x = x-1;
        y *= x; // y = y*x;
        z %= x; // z = z%x;

        System.out.println("And now x="+x+", y="+y+", z="+z);
    }
}
```



# Conditional Operator (? :)

- Boolean-expression ? expression-1 : expression-2

- Example

```
if (a > b)
    max = a;
else
    max = b;
```

OR

```
max = (a > b) ? a : b;
```

# Special Operators : **.(dot), instanceof**

- **.(dot) operator :**

- ✓ It is used to access the instance variables and methods of class objects.

**Ex: stud.age, stud.salary()**

- **Instanceof :**

- ✓ It is used only for object reference variables.
- ✓ The operator checks whether the object is of a particular type(class type or interface type).
- ✓ instanceof operator is written as:

(Object) instanceof (class/interface type)

**Ex: stud instanceof student**

# Programming Errors

- **Syntax Errors**
  - Detected by the compiler
- **Logical Errors**
  - Produces incorrect result
- **Runtime Errors**
  - Causes the program to abort

# Compilation Errors

ShowSyntaxErrors

```
{  
    public static void main(String[] args)  
    {  
        i = 30  
        System.out.println(i+4);  
    }  
}
```

# Runtime Errors

```
class ShowRuntimeErrors
{
    public static void main(String[] args)
    {
        int i = 1 / 0;
    }
}
```

# Logical Errors

```
class ShowLogicalErrors
{
    public static void main(String[] args)
    {
        int sum=0;
        for(int i=1; i>10; i++)
        { sum= sum + i;}
        System.out.println("Sum of 1-10 is "+sum);
    }
}
```

# Garbage Collection

- In Java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
- In other words, it is a way to destroy the unused objects.
- To do so, we were using `free()` function in C language and `delete()` in C++. But, in Java it is performed automatically. So, java provides better memory management.

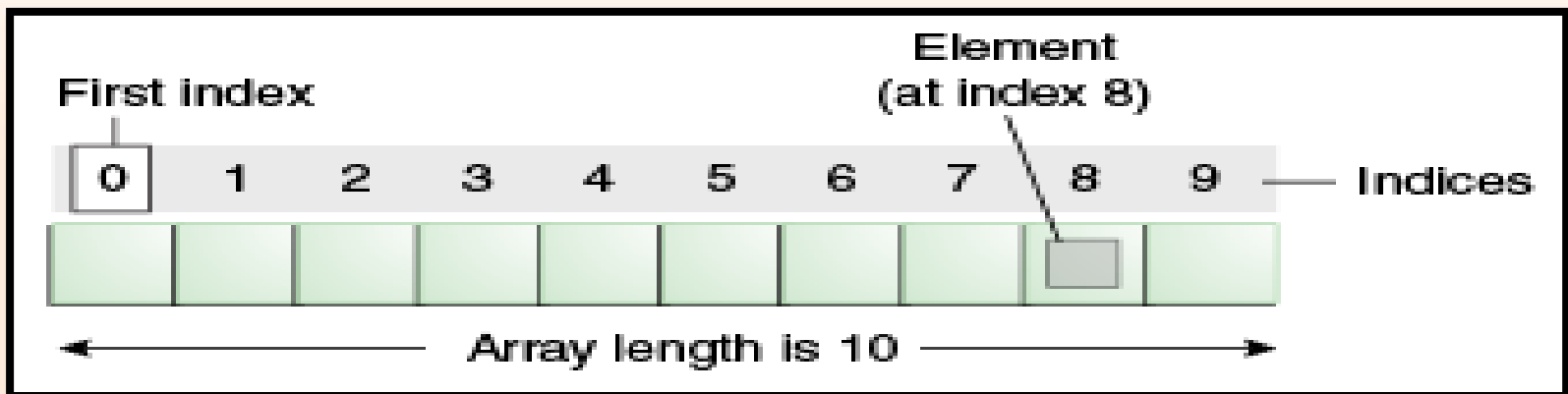
# Advantages...

- Programmer doesn't need to worry about dereferencing an object.
- It is done automatically by JVM.
- Increases memory efficiency and decreases the chances for memory leak.



# Java Array

- ✓ Array is a collection of similar type of elements that have contiguous memory location.
- ✓ Java array **is an object** that contains elements of similar data type.
- ✓ We can store only fixed set of elements in a Java array.
- ✓ Array in Java is index based, first element of the array is stored at 0 index.



# Java Array...

## ✓ **Advantage of Java Array:**

- Code Optimization: It makes the code optimized,
- We can retrieve or sort data easily.
- Random access: We can get any data located at any index position.

## ✓ **Disadvantage of Java Array:**

- Size Limit: We can store only fixed size of elements in the array.
- It doesn't grow its size at runtime.
- To solve this problem, collection framework is used in java.

## ✓ **Types of Array :**

- Single Dimensional Array
- Multidimensional Array
- Jagged Array

# Single Dimensional Array

- ✓ Syntax :

`dataType[] arr;`

or

`dataType []arr;`

or

`dataType arr[];`

- ✓ For ex: `int[] primes;`
- ✓ All syntaxes are equivalent
- ✓ No memory allocation at this point

# Array..

- ✓ Define an array as follows:
  - variable\_name=**new** <type> [Size];
  - Ex : primes=new int[10];
- ✓ Declaring and defining in the same statement:
  - **int[] primes=new int[10];**
- **Note :** In JAVA, int is of 4 bytes, total space= $4*10=40$  bytes

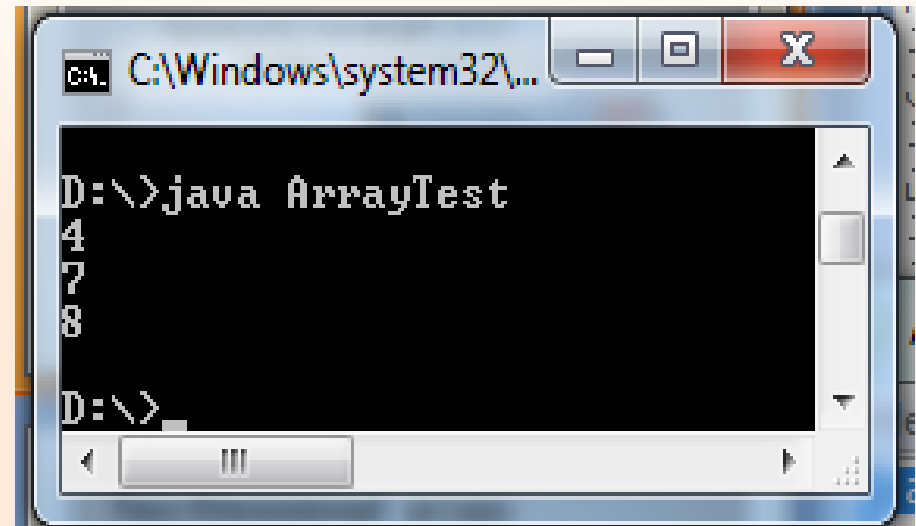
# Example

```
class Test_Array
{ public static void main(String args[])
{   int a[]=new int[5];//declaration and instantiation
    a[0]=10;//initialization
    a[1]=20;
    a[2]=70;
    a[3]=40;
    a[4]=50;
    for(int i=0;i<a.length;i++)//property of array
        System.out.print("\t"+a[i]);
    }
}
```

**Output: 10 20 70 40 50**

# One Dimensional Array

```
class ArrayTest
{
    public static void main(String args[])
    {
        int a[]={4,7,8};
        for(int i=0;i<a.length;i++)
            System.out.println(a[i]);
    }
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\...' and standard window controls. The command prompt shows the command 'D:\>java ArrayTest' being executed. The output of the program is displayed on three separate lines: '4', '7', and '8'. The prompt 'D:\>' is visible at the bottom of the window.

# Multidimensional Array:Array of Array

✓ Two-Dimensional array :

```
int[][] marks=new int[60][3];
```

- 60 arrays each having 3 elements
- First index: specifies array (row)
- Second Index: specifies element in that array (column)
- In JAVA int is 4 bytes,
  - total Size= $4*60*3=720$  bytes

# Array of Array...

marks[0]

0	1	2

marks[1]

0	1	2

marks[2]

0	1	2



# Initialize Array of Array

- `int[][] marks= { {99, 42, 74}, {90, 91, 72}, {88, 61,74}};`
- `//3 arrays with 3 elements each`

marks[0]	0	1	2
	99	42	74

marks[1]	0	1	2
	90	91	72

marks[2]	0	1	2
	88	61	74

# Example...

```
class Test_Array2D
{
    public static void main(String args[])
    {
        int[][] marks= { {99, 42, 74}, {90, 91, 72}, {88, 61, 74}};
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(marks[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

99 42 74

90 91 72

88 61 74

# Jagged Array

- A 2-D array in which the sizes of the second dimension are unequal is known as **Jagged Array**
- In Jagged Array **each row has different number of columns.**
- It is also known as Variable Length Array.

**class Test\_Jagged**

```
{    public static void main(String args[])
    {        int k;
              int twoD[][] = new int[4][];
              twoD[0] = new int[1];
              twoD[1] = new int[2];
              twoD[2] = new int[3];
              twoD[3] = new int[4];
              for (int i = 0; i < 4; i++)
              {                k=1;
                                for (int j = 0; j < i + 1; j++) {
                                    twoD[i][j] = k++;
                                    System.out.print(twoD[i][j] + " ");
                                }

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

**Output:**

```
D:\>java Test_Jagged
1
1 2
1 2 3
1 2 3 4
```

# Conditional Statements

**Branching Statements:**

**if condition**

**switch-case**

## if condition (Simple If-Else) :

### Structure:

```
if ( condition_to_test )
{
    Statement Blocks;
}
else
{
    Statement Blocks;
}
```

## Nested If-Else

### Structure:

```
if ( condition_to_test 1)
{
    if ( condition_to_test 1.1)
        {Statement Blocks;}
    else {Statement Blocks;}
}
else
{
    if ( condition_to_test2 )
    {
        Statement Blocks;}
    else
    {
        Statement Blocks;}
}
```

## Example...

To check number is zero, positive or negative

```
import java.util.*;
class CheckNumber
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number :");
        int x = sc.nextInt();
        if(x == 0)
            System.out.println("x is a zero number);
        else
            if(x < 0)
                System.out.println("x is a negative having value " + x);
            else
                System.out.println("x is a positive having value " + x);
    }
}
```

**Output:**

```
D:\>java CheckNumber
Enter a number :
20
x is a positive number having value 20
```

# Example...

```
import java.util.*;
class Test_If
{   public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter age of user:");
        int user = sc.nextInt();
        if(user>18)
        { System.out.println("User is older than 18");   }
        else
        {System.out.println("User is younger than 18");      }
    }
}
```

# Types of Loop

- for loop
- while loop
- do...while loop
- **for** loop (Advanced)
- Break and Continue with loop



# Ex: Refer Array using for loop

```
public class Test
{
    public static void main(String args[])
    {
        int [] numbers = {10, 20, 30, 40, 50};
        for(int x : numbers )
        {
            System.out.print( x + " , " );
        }
        System.out.println();
        String [] names ={"James", "Larry", "Tom", "Lacy"};
        for( String name : names )
        {
            System.out.print( name+" , " );
        }
    }
}
```

**Output:**

```
10 , 20 , 30 , 40 , 50 ,
James , Larry , Tom , Lacy ,
```

# break keyword

- ✓ The **break** keyword is used to stop the entire loop.
- ✓ The break keyword must be used inside any loop or a switch statement.
- ✓ The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

## Example :

```
public class Test_Break
{
    public static void main(String args[])
    {
        int[] numbers = {10, 20, 30, 40, 50};
        for(int x : numbers )
        {
            if( x == 30 )
            { break; }
            System.out.println( x );
        }
    }
}
```

# continue Keyword

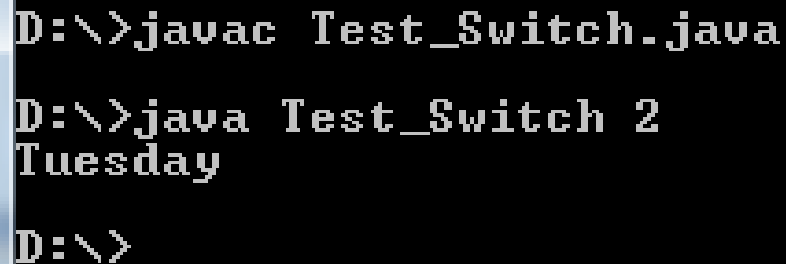
- ✓ The **continue** keyword can be used in any of the loop control structures.
- ✓ It causes the loop to immediately jump to the next iteration of the loop.

## Example :

```
public class Test_Continue
{
    public static void main(String args[])
    {
        int [] numbers ={10, 20, 30, 40, 50};
        for(int x : numbers )
        {
            if( x == 30 )
                { continue; }
            System.out.println( x );
        }
    }
}
```

# Command Line Arguments & Switch Case

```
class Test_Switch
{
    public static void main(String args[])
    {
        int x=Integer.parseInt(args[0]);
        switch(x)
        {
            case 1: System.out.println("Monday"); break;
            case 2: System.out.println("Tuesday"); break;
            case 3: System.out.println("Wednesday"); break;
            case 4: System.out.println("Thursday"); break;
            case 5: System.out.println("Friday"); break;
            case 6: System.out.println("Saturday"); break;
            case 7: System.out.println("Sunday"); break;
            default :System.out.println("Invalid Number of Day");
        }
    }
}
```



```
D:\>javac Test_Switch.java
D:\>java Test_Switch 2
Tuesday
D:\>
```