



JAVA PROGRAMMING

Chap 1 : Java Fundamentals

Program: Master of Computer Application (MCA)				Semester: I	
Course: Java Programming				Code:	
Teaching Scheme				Evaluation Scheme	
Lecture (Hours per week)	Practical (Hours per week)	Tutorial (Hours per week)	Credit	Internal Continuous Assessment (ICA) (Marks - 50)	Term End Examinations (TEE) (Marks - 100)
2	4	0	4	Marks Scaled to 50	Marks Scaled to 50
Prerequisite: NA					
Course Objective This course will impart knowledge of object-oriented programming, building graphical user interface and database connectivity using Java.					
Course Outcomes After completion of the course, students will be able to - <ol style="list-style-type: none"> 1. Implement programs using object oriented programming paradigm 2. Implement programs using collection and generics concepts 3. Develop GUI application with database connectivity 					

Detailed Syllabus:		
Unit	Description	Duration
1	Java Fundamentals Overview of Java, Using Blocks of code, Lexical Issues, Java Class Libraries, Data Types, Variables and Arrays, Operators, Control Statements, Command Line Arguments.	02
2	Classes and Methods Class fundamentals, Declaring Objects, Constructors, Methods, Overloading of methods, Access control, Static and final variables.	04
3	Inheritance Inheritance Basics, method overriding, using abstract classes, using final with inheritance.	04
4	Packages and Interfaces Packages, Access Protection, importing packages, , Interfaces: Defining an Interface, Implementing Interfaces , Applying Interfaces, Variables in Interfaces.	03
5	Exception Handling Exception handling fundamentals, exception types, uncaught exceptions, using try and catch, throw, throws, finally, Java's built-in exceptions, creating your own exceptions.	02
6	Programs using String Handling String Constructors, Special String operators, Character Extraction, String Comparison, Searching Strings and Modifying Strings, Buffer class and its methods.	02
7	Generics and Collections	05

	Generics: Introduction, A Generic class with Type Parameters, General Form of a Generic class, Bounded Types, Using wildcard arguments, Creating a Generic Method, Generic class Hierarchies, Collection: Collection Framework, ArrayList class, List Iterator interface, Linked List class, TreeSet class	
8	GUI design and Event Handling using Java FX Introduction, JavaFX Architecture, application structure, JavaFX, Text, Effect, Anim, UI controls. Types of Events, Processing Events in JavaFX, Event Delivery Process, Event Handlers.	05
9	Java and Database Programming JDBC Architecture, Types of Drivers, JDBC components, JDBC classes and Interfaces, steps for querying the database with JDBC, Database connection, querying and updating database tables, passing parameters to a statement.	03
	Total	30
Text Books:		
1. Herbert Schildt, <i>Java The Complete Reference</i> , 11 th Edition, Oracle Press, 2020.		
2. Sergey Grinev, <i>Mastering JavaFX10</i> , Packt Publishing, 2018.		
Reference Books:		
1. Cay Horstmann, <i>Core Java Volume I- Fundamentals</i> , Pearson Education Inc., 2020.		
2. Carl Dea, Gerrit Grunwald, José Pereda, Sean Phillips, Mark Heckler, <i>JavaFX 9 by Example</i> , 3 rd Edition, Apress, 2017.		

- Java is a programming language and a platform.
- Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995.
- James Gosling is known as the father of Java.
- Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

- There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application : Also known as desktop applications or window-based applications are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

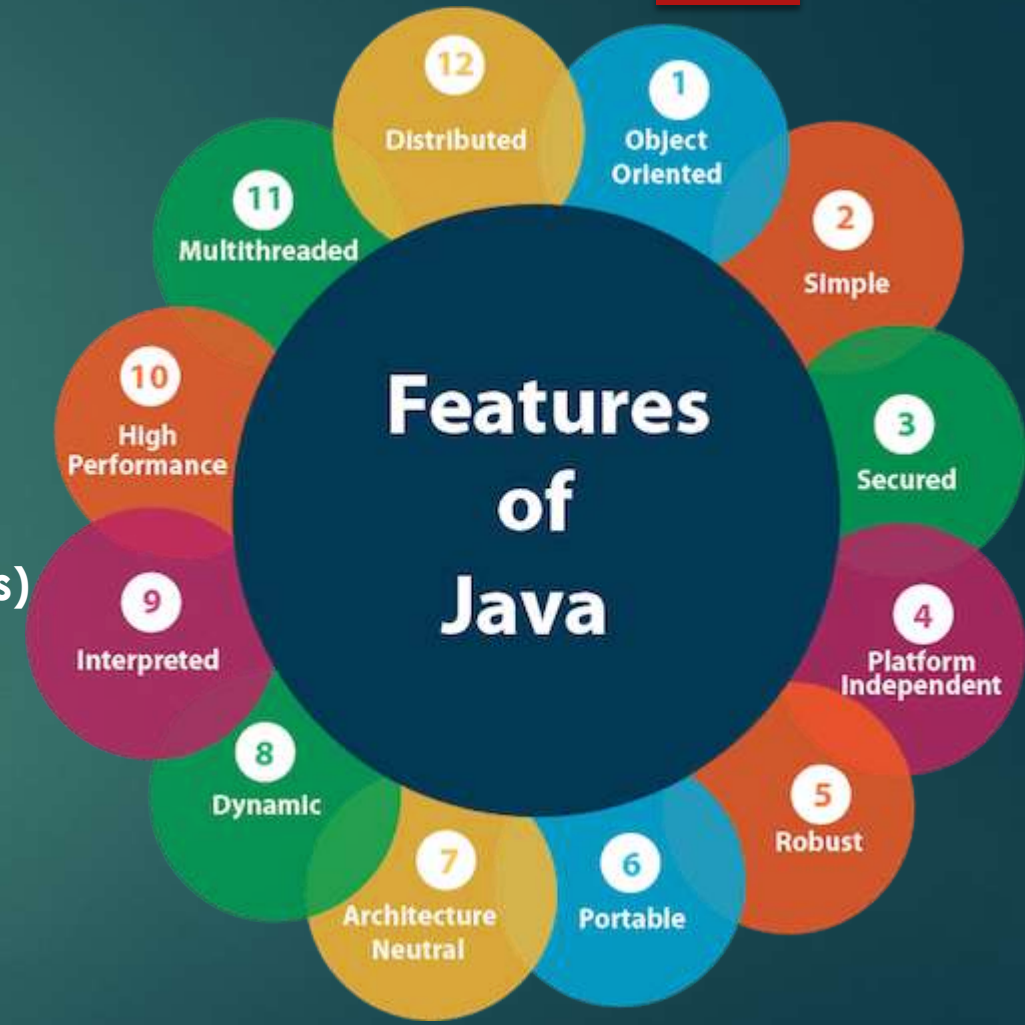
2) Web Application : An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application : An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application : An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Features of Java

- 1) Simple - (No pointers, no operator overloading, no garbage collection, no multiple inheritance)
- 2) Object-Oriented
- 3) Portable - (bytecode)
- 4) Platform independent - (bytecode)
- 5) Secured - (no explicit pointer)
- 6) Robust - (strong MM Mgmt, Exception Handling, Auto Garbage Collection, no pointers)
- 7) Architecture neutral - (bytecode, fixed size of primitive datatypes)
- 8) Interpreted
- 9) High Performance
- 10) Multithreaded
- 11) Distributed - (helps to create distributed applns using RMI & EJB)
- 12) Dynamic - (uses JIT)
- 13) Garbage Collected



Basic concepts of OOPs are:

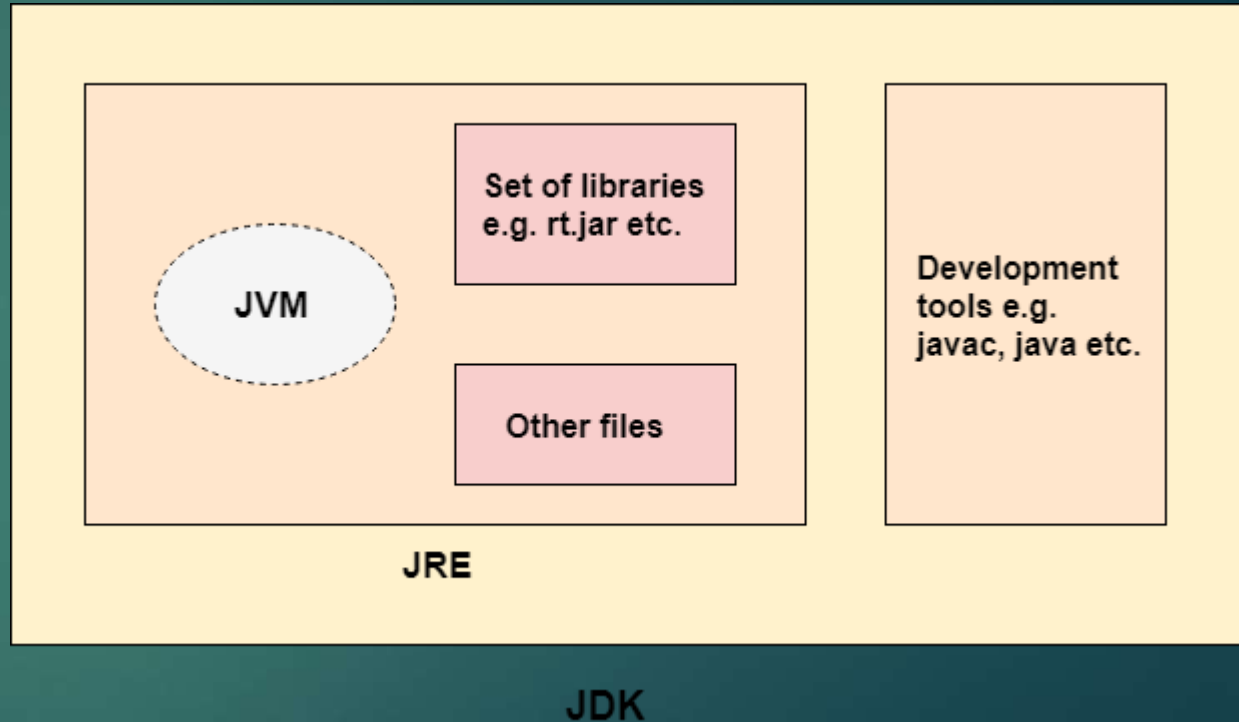
- 1) Object
- 2) Class
- 3) Inheritance
- 4) Polymorphism
- 5) Abstraction
- 6) Encapsulation

C++	Java
It is object oriented but not purely. That is a program can be written in C++ without using classes and objects.	It is purely object oriented that is no program can be written without classes and objects.
C++ is platform dependent	Java is platform independent
C++ supports the goto statement.	Java doesn't support the goto statement.
C++ supports pointers. This makes program system dependent.	Java doesn't support pointers to avoid platform dependency.
C++ supports multiple and hybrid inheritance	Java does not support multiple and hybrid inheritance. A slight implementation of the same can be done using a special feature called as Interface.
Supports operator overloading	Does not allow operator overloading
Supports 3 access specifiers : public, protected and private	Supports 5 access specifiers : public, protected, private, default and private protected
Destructors are used in C++	Java is garbage collected that is the objects are automatically destroyed once their use is over
C++ doesn't have exception handling	Java has exception handling
Doesn't support multithreading	Supports multithreading

[illegible]

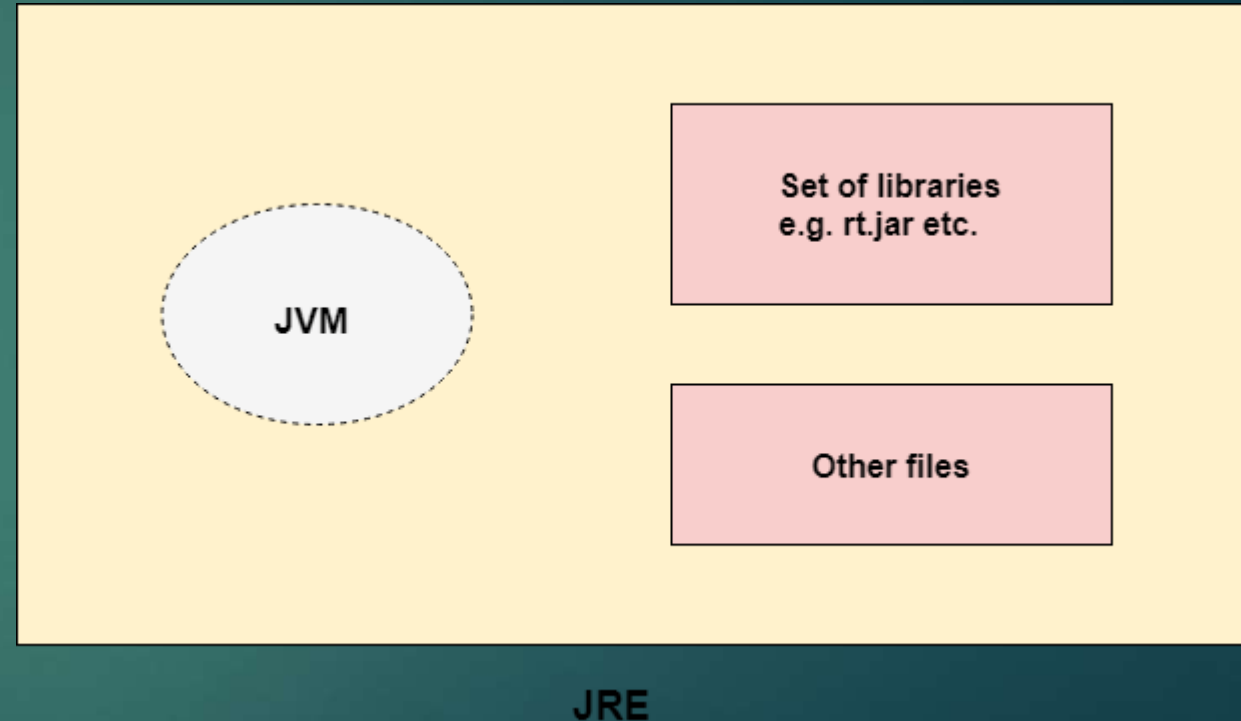
Java Development Kit (JDK)

- ▶ JDK is an acronym for Java Development Kit.
- ▶ The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.
- ▶ It physically exists. It contains JRE + development tools.
- ▶ The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



Java Runtime Environment (JRE)

- ▶ JRE is an acronym for Java Runtime Environment.
- ▶ It is also written as Java RTE.
- ▶ The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- ▶ It is used to provide the runtime environment.
- ▶ It is the implementation of JVM.
- ▶ It physically exists.
- ▶ It contains a set of libraries + other files that JVM uses at runtime.
- ▶ The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



Java Virtual Machine (JVM)

- ▶ JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- ▶ JVM is an interpreter. It is the main component of Java that makes java platform independent.
- ▶ It is because the bytecode file can be taken onto any system and the JVM residing in that system will interpret and translate the byte code into machine readable code.
- ▶ The JVM performs following operation:
 - ▶ Loads code
 - ▶ Verifies code
 - ▶ Executes code
 - ▶ Provides runtime environment

Tokens of Java

- ▶ Character Set
- ▶ Keywords
- ▶ Identifiers
- ▶ Constants & variables
- ▶ Datatypes
- ▶ operators

Character Set

- ▶ Alphabets: Both lowercase (a, b, c, d, e, etc.) and uppercase (A, B, C, D, E, etc.).
- ▶ Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- ▶ Special symbols: `_`, `(`, `)`, `{`, `}`, `[`, `]`, `+`, `-`, `*`, `/`, `%`, `!`, `&`, `|`, `~`, `^`, `<`, `=`, `>`, `$`, `#`, `?`, Comma `,`, Dot `.`, Colon `:`, Semi-colon `;`, Single quote `'`, Double quote `"`, Back slash `\`.
- ▶ White space: Space, Tab, New line.

Keywords

- ▶ There are 50 keywords currently defined in the Java language.
- ▶ These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
- ▶ **These keywords cannot be used as identifiers.**
- ▶ Thus, they cannot be used as names for a variable, class, or method.
- ▶ In addition to the keywords, Java reserves the following: **true**, **false**, and **null**. These are values defined by Java. You may not use these words for the names of variables, classes, and so on.

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

Table 2-1 Java Keywords

Identifiers

- ▶ Identifiers are used to name things, such as classes, variables, and methods.
- ▶ An identifier may be any descriptive sequence of **uppercase and lowercase letters**, **numbers**, or the **underscore** and **dollar-sign** characters. (The dollar-sign character is not intended for general use.)
- ▶ Following rules need to be followed while creating identifiers
 1. An identifier can consist of Alphabets, digits, and two special symbols _ (underscore) and \$ (dollar)
 2. An identifier cannot start with a digit. It has to start with an alphabet or _ or \$
 3. No other special symbols apart from _ and \$ are allowed. Not even blank spaces.
 4. An identifier cannot be a keyword.
 5. It is case sensitive. That is firstName, FirstName, firstname are three different identifiers.

Identifiers

Exercise : Identify valid and invalid identifiers from the list given below –

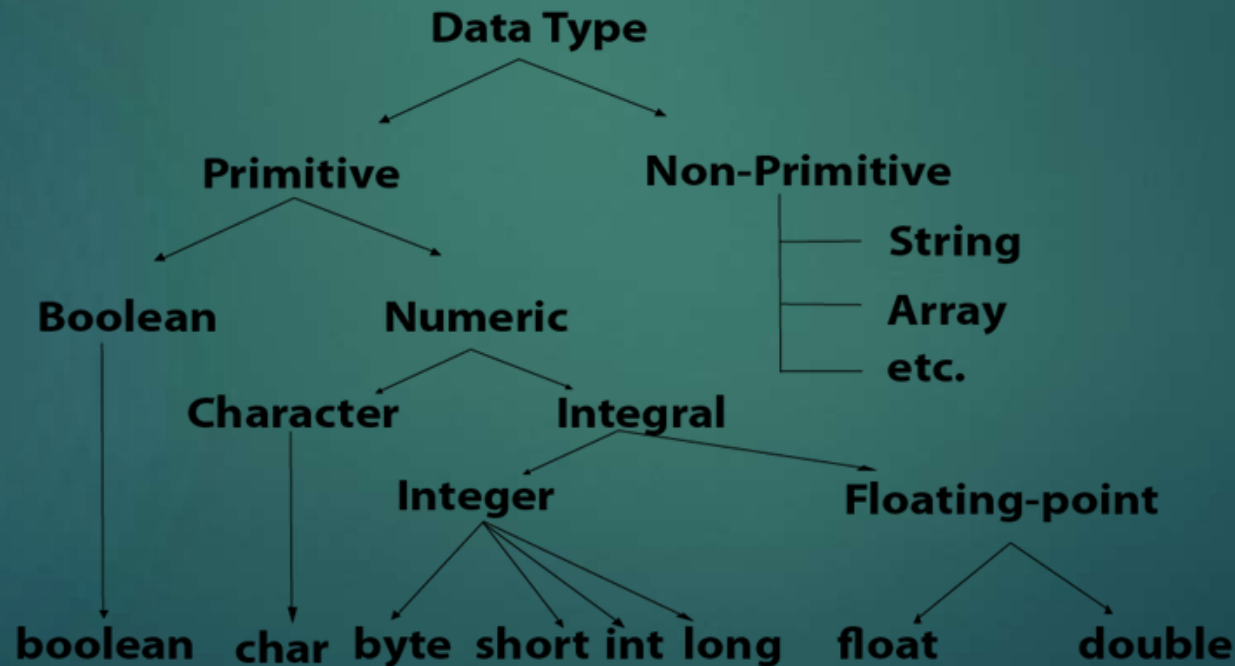
1. simple_interest
2. char
3. 3friends
4. _3\$friends
5. Simple interest
6. #3friends
7. void
8. Void
9. \$xyz

Constants and variables

- ▶ A **constant** is an entity in programming that is immutable.
 - ▶ In other words, the value that cannot be changed after assigning it.
 - ▶ In java constants can be defined by using the keyword "final" before the datatype.
 - ▶ Constants are used to declare values that remain constant like value of pi.
 - ▶ Eg : final double pi=3.14;
-
- ▶ **Variables** are containers for storing data values.
 - ▶ A variable is assigned with a data type.
 - ▶ A variable is the name of a reserved area allocated in memory.
 - ▶ In other words, it is a name of the memory location.
 - ▶ It is a combination of "vary + able" which means its value can be changed.
 - ▶ eg: **int** data=50; //Here data is a variable

Datatypes

- ▶ Data types specify the different sizes and values that can be stored in the variable.
- ▶ There are two types of data types in Java:
- ▶ **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- ▶ **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



Datatypes

Data Type	Default Value	Default size	Range
boolean	false	1 bit	true or false
char	'\u0000'	2 byte	'\u0000' (or 0) to '\uffff' (or 65,535 inclusive).
byte	0	1 byte	-128 to 127 (inclusive)
short	0	2 byte	-32,768 to 32,767 (inclusive)
int	0	4 byte	- 2,147,483,648 to 2,147,483,647 (inclusive)
long	0L	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive)
float	0.0f	4 byte	-3.4e38 to 3.4e38
double	0.0d	8 byte	Its value range is unlimited

Operators

► There are many types of operators in Java which are given below:

1. Unary Operator
2. Arithmetic Operator
3. Shift Operator
4. Relational Operator
5. Bitwise Operator
6. Logical Operator
7. Ternary Operator
8. Assignment Operator

Operators

► Operator precedence

Operator Type	Category	Precedence
Unary	postfix	<i>expr</i> ++ <i>expr</i> --
	prefix	++ <i>expr</i> -- <i>expr</i> + <i>expr</i> - <i>expr</i> ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical	logical AND	&&
	logical OR	
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Ternary Operator

- ▶ The operator that requires three operands is called as Ternary Operator.
- ▶ Java supports only one ternary operator `? :`
- ▶ Syntax : `(condition) ? (Value if true) : (value if false)`

```
import java.util.*;

class Ternary{

    public static void main(String args[]){
        int a, b, large;

        Scanner sc=new Scanner(System.in);
        System.out.println("Enter 2 numbers");
        a=sc.nextInt();
        b=sc.nextInt();
        large=(a>b)?a:b;
        System.out.println("Largest number is : "+large);
    }
}
```

```
F:\Java>javac Ternary.java
```

```
F:\Java>java Ternary
```

```
Enter 2 numbers
```

```
15
```

```
34
```

```
Largest number is : 34
```


Comments

- ▶ Single line Comments - `// comments`
- ▶ Multi line comments - `/* comments */`
- ▶ Documentation comments - `/** comments */`

Input/Output in Java

Output in Java

- ▶ `System.out.println("Hello");`
- ▶ `System` – class of `Java.lang` package
- ▶ `out` – variable of class `System` corresponding to the std o/p device i.e. monitor
- ▶ `println()/print()` – static methods for displaying content

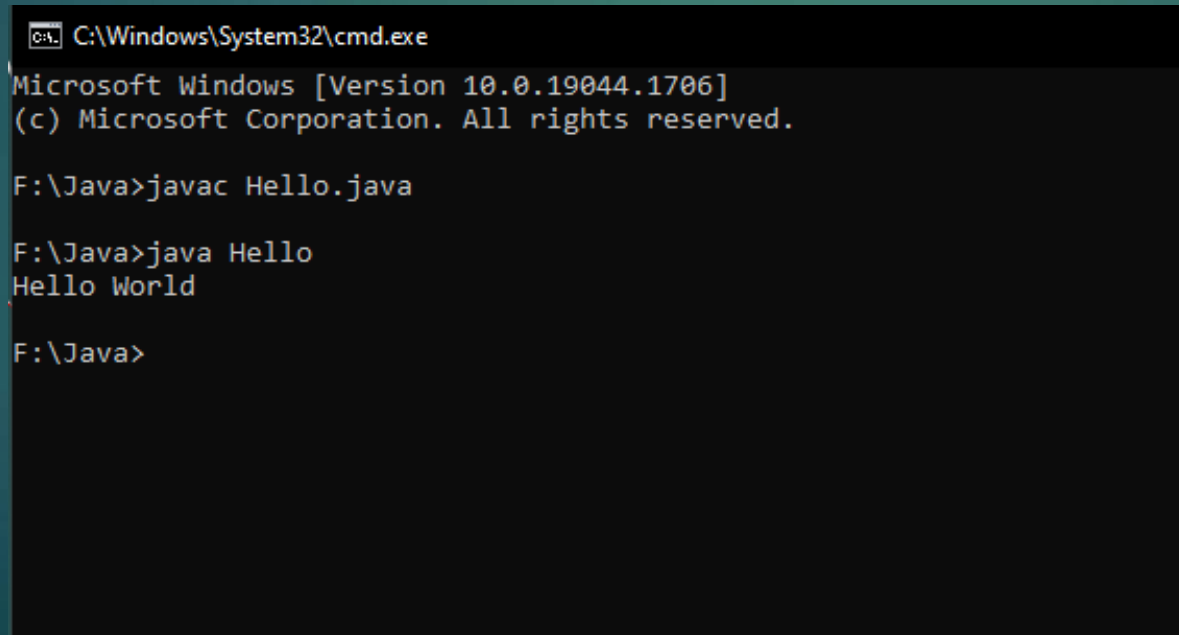
Input in Java

- ▶ `BufferedReader` – `readLine()`
- ▶ `Scanner` – `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, `next()`, `nextLine()`

Simple Java Program

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

Output :



```
C:\Windows\System32\cmd.exe  
Microsoft Windows [Version 10.0.19044.1706]  
(c) Microsoft Corporation. All rights reserved.  
  
F:\Java>javac Hello.java  
  
F:\Java>java Hello  
Hello World  
  
F:\Java>
```

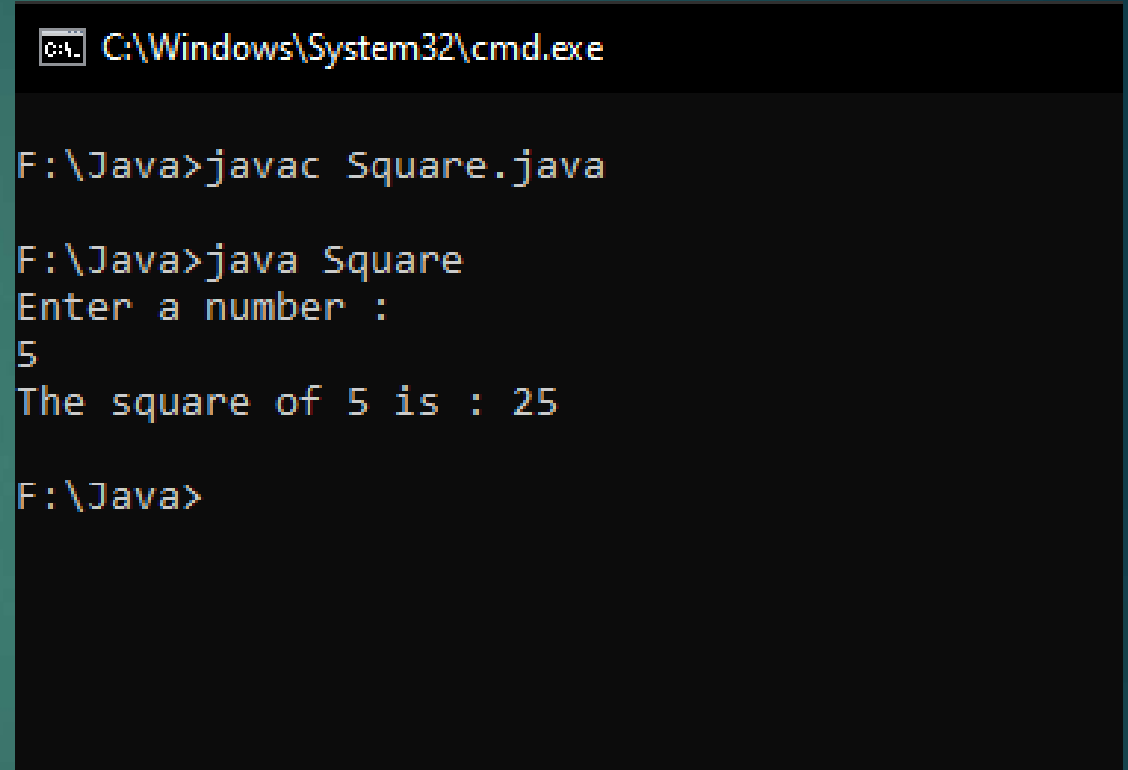
Program for accepting input (Scanner)

```
import java.util.*;

class Square{
    public static void main(String args[])
    {
        int i, res;

        Scanner sc=new Scanner (System.in);
        System.out.println("Enter a number : ");
        i=sc.nextInt();
        res=i*i;

        System.out.println("The square of " + i + " is :
" + res);
    }
}
```



```
C:\Windows\System32\cmd.exe

F:\Java>javac Square.java

F:\Java>java Square
Enter a number :
5
The square of 5 is : 25

F:\Java>
```

Program for accepting input (BufferedReader)

```
import java.io.*;

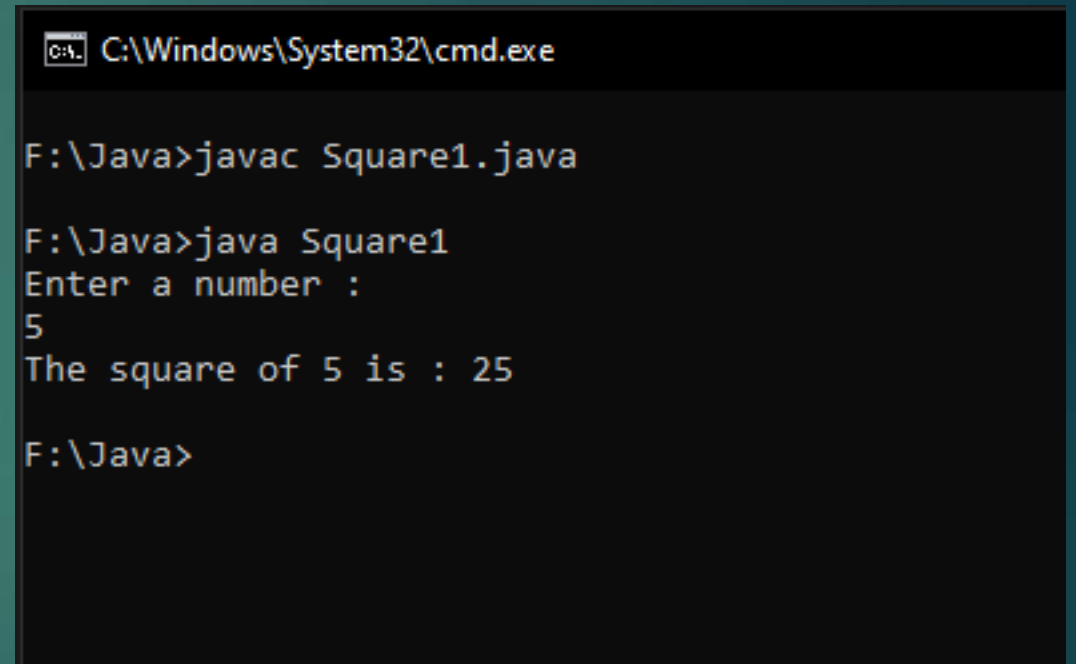
class Square{
    public static void main(String args[]) throws IOException
    {
        int i, res;

        BufferedReader br=new BufferedReader (new
InputStreamReader(System.in));

        String str;

        System.out.println("Enter a number : ");
        str=br.readLine();
        i=Integer.parseInt(str);
        res=i*i;

        System.out.println("The square of " + i + " is : " + res);
    }
}
```

A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The prompt is at 'F:\Java>'. The user has entered 'javac Square1.java' and 'java Square1'. The program prompts 'Enter a number :', the user enters '5', and the program outputs 'The square of 5 is : 25'. The prompt returns to 'F:\Java>'.

```
C:\Windows\System32\cmd.exe

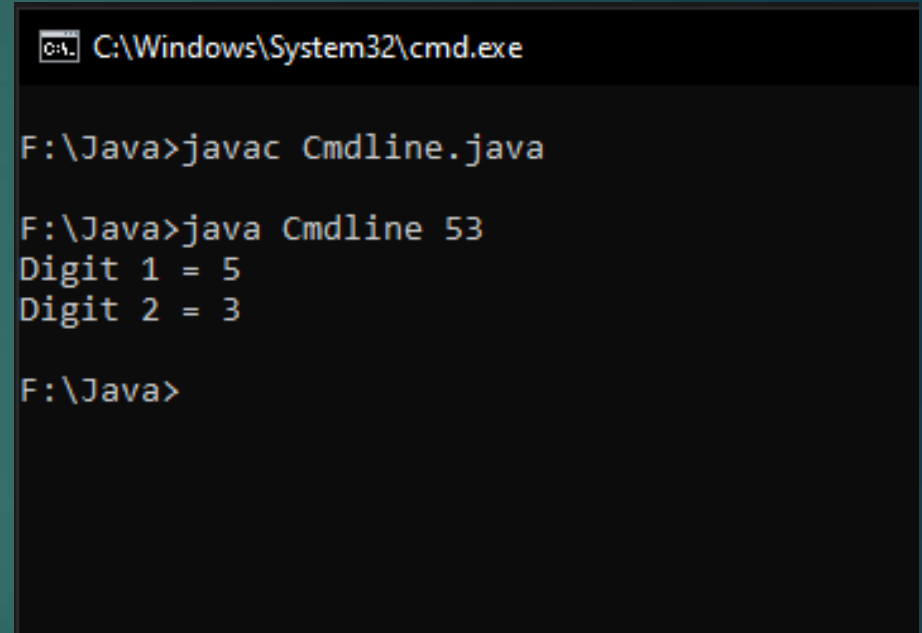
F:\Java>javac Square1.java

F:\Java>java Square1
Enter a number :
5
The square of 5 is : 25

F:\Java>
```


Program for accepting input (command line)

```
class Cmdline{
    public static void main(String args[])
    {
        int n,d1,d2;
        n=Integer.parseInt(args[0]);
        d2=n%10;
        d1=n/10;System.out.println("Digit 1 = "+d1+"\nDigit 2 = "+d2);
    }
}
```



```
C:\Windows\System32\cmd.exe

F:\Java>javac Cmdline.java

F:\Java>java Cmdline 53
Digit 1 = 5
Digit 2 = 3

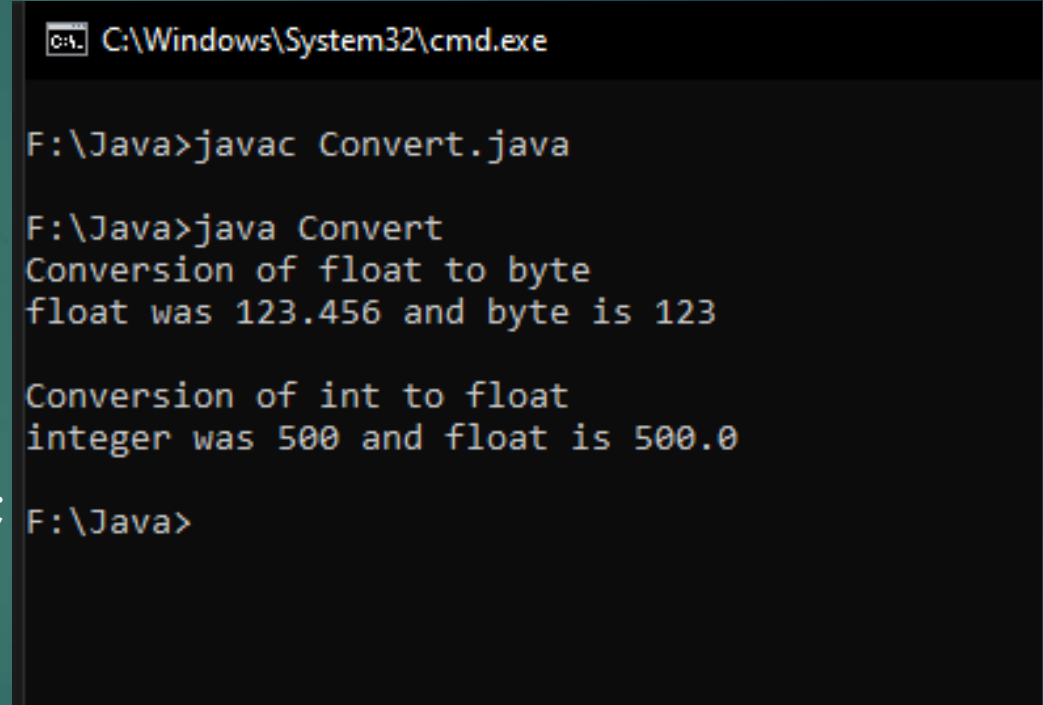
F:\Java>
```

Typecasting in Java

- ▶ **Type casting** is a method or process that converts a data type into another data type in both ways manually and automatically.
- ▶ The automatic conversion is done by the compiler and manual conversion performed by the programmer
- ▶ Converting a lower data type into a higher one is called as **implicit conversion** or **casting down**.
- ▶ It is done automatically.
- ▶ It takes place when:
 - ▶ Both data types must be compatible with each other.
 - ▶ The target type must be larger than the source type.
- ▶ Eg : typecasting int to long
- ▶ Converting a higher data type into a lower one is called as **explicit conversion** or **casting up**.
- ▶ It is done manually by the programmer.
- ▶ If we do not perform casting then the compiler reports a compile-time error.
- ▶ Eg : typecasting float to byte
- ▶ Syntax for explicit typecasting –
(target datatype)source datatype variable
- ▶ Eg : float x;
(byte) x;

Typecasting in Java

```
class Convert{
    public static void main(String args[])
    {
        byte b;
        int i=500;
        float f=123.456f;
        System.out.println("Conversion of float to byte");
        b=(byte) f; // explicit
        typecasting
        System.out.println("float was "+f+ " and byte is "+b+"\n");
        System.out.println("Conversion of int to float");
        f=i; // implicit typecasting
        System.out.println("integer was "+i+ " and float is "+f);
    }
}
```



```
C:\Windows\System32\cmd.exe

F:\Java>javac Convert.java

F:\Java>java Convert
Conversion of float to byte
float was 123.456 and byte is 123

Conversion of int to float
integer was 500 and float is 500.0

F:\Java>
```

Control statements

- ▶ Control statements are of two types
 - ▶ Conditional statements – if-else, switch case
 - ▶ Iterative statements – for, while, do while

Conditional Statements

If-Else statements

```
import java.util.*;

class PositiveNegativeExample {
    public static void main(String[] args) {
        int n;
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter n : ");
        n=sc.nextInt();
        if(n>0){
            System.out.println("POSITIVE");
        }
        else if(n<0){
            System.out.println("NEGATIVE");
        }
        else{
            System.out.println("ZERO");
        }
    }
}
```

Switch statements

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        int day;
        Scanner sc=new Scanner (System.in);
        System.out.println("Enter the day number : ");
        day=sc.nextInt();
        switch (day) {
            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;
        }
    }
}
```

Conditional Statements

WAP to check if the user entered year is leap or not

```
import java.util.*;
class Leap {
    public static void main(String[] args) {
        int yr;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the yr");
        yr=sc.nextInt();
        if((yr%4==0 && yr%100!=0) || (yr%100==0 && yr%400==0))
            System.out.println("Leap Year");
        else
            System.out.println("Not a Leap Year");
    } }
```

```
F:\Java>javac Leap.java
```

```
F:\Java>java Leap
Enter the yr
2024
Leap Year
```

```
F:\Java>java Leap
Enter the yr
2023
Not a Leap Year
```


Conditional Statements

- WAP to display the class according to the marks scored by student using switch case. The marks scored is taken input from the user and the class is displayed based on the range given below

Marks	Class
70-100	Distinction
60-69	First
50-59	Second
40-49	Pass
0-39	Fail

```
import java.util.*;
class Marks {
    public static void main(String[] args) {
        int marks;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter marks out of 100");
        marks=sc.nextInt();
        if (marks==100)
            System.out.println("Distinction");
        switch(marks/10)
        {
            case 0:
            case 1:
            case 2:
            case 3: System.out.println("Fail");
                    break;
            case 4: System.out.println("Pass Class");
                    break;
            case 5: System.out.println("Second Class");
                    break;
```

```
            case 6: System.out.println("First Class");
                    break;
            case 7:
            case 8:
            case 9: System.out.println("Distinction");
                    break;
            default: System.out.println("InvalidMarks");
        } } }
```

```
F:\Java>javac Marks.java

F:\Java>java Marks
Enter marks out of 100
85
Distinction

F:\Java>java Marks
Enter marks out of 100
37
Fail
```

Iterative Statements



For loop

```
import java.util.*;

class Natural{

    public static void main(String args[])
    {
        int i, n;

        Scanner sc=new Scanner
(System.in);

        System.out.println("Enter n : ");
        n=sc.nextInt();
        for(i=1;i<=n;i++)
        {
            System.out.println(i);
        }
    }
}
```

While loop

```
import java.util.*;

class NaturalWhile{

    public static void main(String args[])
    {
        int i, n;

        Scanner sc=new Scanner (System.in);
        System.out.println("Enter n : ");
        n=sc.nextInt();
        i=1;
        while(i<=n)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

Do-while loop

```
import java.util.*;

class NaturalWhile{

    public static void main(String args[])
    {
        int i, n;

        Scanner sc=new Scanner
(System.in);

        System.out.println("Enter n : ");
        n=sc.nextInt();
        i=1;
        do
        {
            System.out.println(i);
            i++;
        }while(i<=n);
    }
}
```

Iterative Statements

WAP to find factorial of a number

```
import java.util.*;
class Fact {
public static void main(String[] args) {
int n,i,fact=1;
Scanner sc=new Scanner(System.in);
System.out.println("Enter the number");
n=sc.nextInt();
for(i = 1; i <= n; i++)
{ fact = fact * i; }
System.out.println("Factorial of " + n + " is " + fact);
}
}
```

```
F:\Java>java Fact
Enter the number
5
Factorial of 5 is 120
```

WAP to display the pattern

```
import java.util.*;
class StarPattern {
public static void main(String[] args) {
int n,i,j;
Scanner sc=new Scanner(System.in);
System.out.println("Enter the number");
n=sc.nextInt();
for(i=1;i<=n;i++)
{
for(j=1;j<=i;j++)
{
System.out.print("* ");
}
System.out.println();
}}}
```

```
F:\Java>javac StarPattern.java
F:\Java>java StarPattern
Enter the number
6
*
* *
* * *
* * * *
* * * * *
* * * * *
```

Arrays (single dimensional)

- ▶ an array is a collection of multiple data of same datatype
- ▶ Indexing in an array starts from 0. The last index is $n-1$ where n is the size of the array.
- ▶ Syntax for declaring array : `datatype array_name[]=new datatype[array_size];`
- ▶ Eg : `int arr[]=new int[10];`

Arrays (single dimensional)

WAP to input and display n elements in an array

```
import java.util.*;

class Array{

public static void main(String args[])

{

int i,n;

Scanner sc=new Scanner(System.in);

System.out.println("Enter no. of elements :");

n=sc.nextInt();

int a[]=new int[n];

for(i=0;i<=n-1;i++)

{

System.out.println("Enter a number :");

a[i]=sc.nextInt();

}
```

```
System.out.println("The elements in the array are :");

for(i=0;i<=n-1;i++)

{

System.out.print(a[i]+"\\t");

}}
```

```
F:\\Java>javac Array.java
```

```
F:\\Java>java Array
```

```
Enter no. of elements :
```

```
5
```

```
Enter a number :
```

```
1
```

```
Enter a number :
```

```
6
```

```
Enter a number :
```

```
7
```

```
Enter a number :
```

```
5
```

```
Enter a number :
```

```
4
```

```
The elements in the array are :
```

```
1      6      7      5      4
```

```
F:\\Java>
```

Arrays (single dimensional)

WAP to sort an array in ascending order

```
import java.util.*;

class SortedArray{

public static void main(String args[])

{

int i,n,temp;

Scanner sc=new Scanner(System.in);

System.out.println("Enter no. of elements :
");

n=sc.nextInt();

int a[]=new int[n];

for(i=0;i<=n-1;i++)

{

System.out.println("Enter a number :");

a[i]=sc.nextInt();

}
```

```
for(i=0;i<=n-2;i++)

{

for(int j=0;j<=n-2;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

System.out.println("Sorted Array: ");

for(i=0;i<=n-1;i++)

{

System.out.print(a[i]+"\\t");

}

}}
```

```
F:\Java>javac SortedArray.java
```

```
F:\Java>java SortedArray
```

```
Enter no. of elements :
```

```
5
```

```
Enter a number :
```

```
4
```

```
Enter a number :
```

```
9
```

```
Enter a number :
```

```
5
```

```
Enter a number :
```

```
2
```

```
Enter a number :
```

```
7
```

```
Sorted Array:
```

```
2
```

```
4
```

```
5
```

```
7
```

```
9
```


Arrays (multi dimensional)

- ▶ The Java multidimensional arrays are arranged as an array of arrays i.e. each element of a multi-dimensional array is another array.
- ▶ The representation of the elements is in rows and columns.
- ▶ Thus, you can get a total number of elements in a multidimensional array by multiplying row size with column size.
- ▶ So if you have a two-dimensional array of 3×4 , then the total number of elements in this array = $3 \times 4 = 12$
- ▶ The simplest of the multi-dimensional array is a two-dimensional array.
- ▶ A simple definition of 2D arrays is: A 2D array is an array of one-dimensional arrays.
- ▶ In Java, a two-dimensional array is stored in the form of rows and columns and is represented in the form of a matrix.
- ▶ The general declaration of a two-dimensional array is,
- ▶ `data_type[][] array_name = new data_type[row_size][column_size];`
- ▶ Eg : `int a[][]=new int[3][4];`

Arrays (multi dimensional)

```
import java.util.*;
class Array2d
{
    public static void main(String[] args) {
        int i,j,m,n;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no. of rows and columns :
");
        m=sc.nextInt();
        n=sc.nextInt();
        int a[][] = new int[m][n];
        for (i=0;i<=m-1;i++) {
            for (j=0;j<=n-1;j++) {
                System.out.println("Enter a number : ");
                a[i][j]=sc.nextInt();
            }
        }
```

```
        System.out.println("The Array in matrix form is:");
        for (i=0;i<=m-1;i++) {
            for (j=0;j<=n-1;j++) {
                System.out.print(a[i][j]+"\\t");
            }
            System.out.println();
        } }
```

```
F:\Java>javac Array2d.java
F:\Java>java Array2d
Enter no. of rows and columns :
3
2
Enter a number :
1
Enter a number :
2
Enter a number :
3
Enter a number :
4
Enter a number :
5
Enter a number :
6
The Array in matrix form is:
1      2
3      4
5      6
```

Arrays (multi dimensional)



WAP to find and display sum of the diagonal elements of a square matrix

```
import java.util.*;
class Diagonal
{
    public static void main(String[] args) {
        int i,j,m,n,sum=0;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no. of rows and columns :");
        m=sc.nextInt();
        n=sc.nextInt();
        int a[][] = new int[m][n];
        for (i=0;i<=m-1;i++) {
            for (j=0;j<=n-1;j++) {
                System.out.println("Enter a number :");
                a[i][j]=sc.nextInt();
            }
        }
```

```
        for (i=0;i<=m-1;i++) {
            for (j=0;j<=n-1;j++) {
                if(i==j)
                    sum=sum+a[i][j];
            }
        }
        System.out.println("Sum= "+sum);
    } }
```

```
F:\Java>javac SumofDiagonalElements.java

F:\Java>java Diagonal
Enter no. of rows and columns :
2
2
Enter a number :
4
Enter a number :
3
Enter a number :
9
Enter a number :
1
Sum= 5
```

Programming Practice Questions

- ▶ WAP to accept a number and display its equivalent ASCII character using type casting.
- ▶ WAP to find largest of three numbers accepted from the user through command line using ternary operator.
- ▶ WAP to display first n odd numbers
- ▶ WAP to calculate and display sum of square of first n natural numbers.
- ▶ WAP to display first n elements of Fibonacci series.
- ▶ WAP to calculate value of following series – $1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$
- ▶ WAP to calculate value of following series – $1 + 1/2! + 1/3! + 1/4! + \dots + 1/n!$
- ▶ WAP to calculate and display value of s for $t=1,5,10,15,20,\dots,100$. Formula : $s = s_0 + v_0 + \frac{1}{2} a t^2$
- ▶ WAP to display following patterns

```
  *
 ***
 *****
 *******
*****
```

```
 *
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
      1
    1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1
```

```
A
B C
D E F
G H I J
K L M N O
```

```
  *
 ***
*****
*****
*****
*****
***
 *
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Programming Practice Questions

- ▶ WAP to count number of digits in a user entered number using while loop.
- ▶ WAP to display a user entered number as a sequence of individual digits using while loop. Eg : 1691 should be displayed as 1 6 9 1
- ▶ WAP to check if a user entered number is divisible by 3 and 5 or not.
- ▶ WAP to check if the entered number is Armstrong number or not.
- ▶ WAP to display the class according to the marks scored by student using if else if. The marks scored is taken input from the user and the class is displayed based on the range given below

Marks	Class
70-100	Distinction
60-69	First
50-59	Second
40-49	Pass
0-39	Fail

- ▶ WAP to display a user digit number in words using switch case. Eg : i/p : 123 o/p : One Two Three
- ▶ WAP to find GCD and LCM of user entered two numbers.
- ▶ WAP to find largest element from the array.

Programming Practice Questions

- ▶ WAP to display average of n integers stored in an array
- ▶ WAP to search an element in an array and if found display the index.
- ▶ WAP to sort an array.
- ▶ WAP to add two matrices of size m x n
- ▶ WAP to find transpose of a matrix of size m x n
- ▶ The annual examination results of 5 students are tabulated as follows

Roll No	Subject 1	Subject 2	Subject 3
1			
2			
3			
4			
5			

WAP to read the data and determine the following

- Total marks obtained by each student
- The highest total marks and the student who obtained highest total marks.



JAVA PROGRAMMING

Chap 2 : Classes and Methods

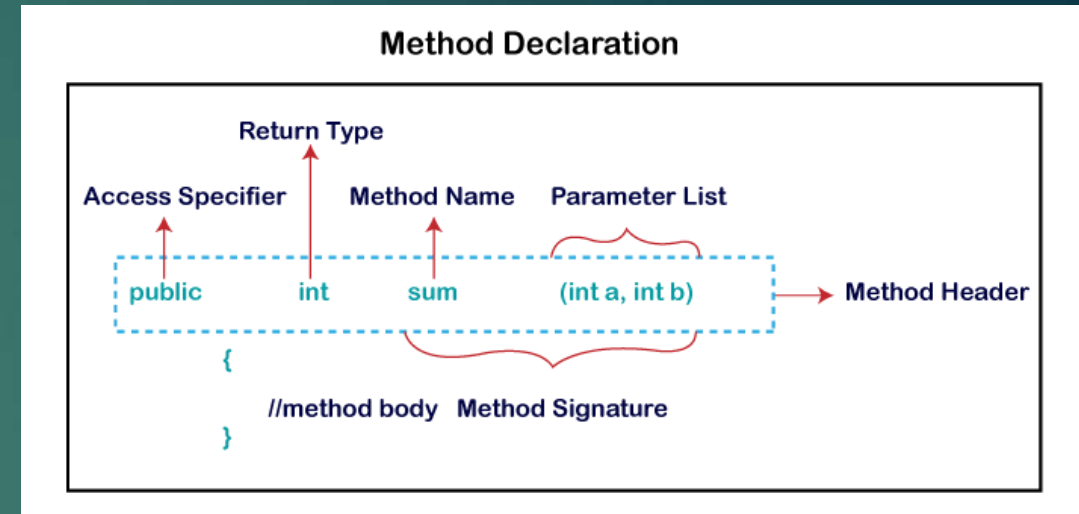
- ▶ In object-oriented programming technique, we design a program using objects and classes.
- ▶ An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.
- ▶ Method in Java is a collection of instructions that performs a specific task.
- ▶ It provides the reusability of code. We can also easily modify code using methods.

Methods in Java

- ▶ A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- ▶ It is used to achieve the **reusability** of code.
- ▶ We write a method once and use it many times.
- ▶ We do not require to write code again and again.
- ▶ It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.
- ▶ The method is executed only when we call or invoke it.
- ▶ The most important method in Java is the **main()** method.
- ▶ The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.

Methods in Java

- ▶ It has six components that are known as **method header**, as we have shown in the following figure
- ▶ **Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.
- ▶ **Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **five** types of access specifier
- ▶ **Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.
- ▶ **Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. A method is invoked by its name.
- ▶ **Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, we leave the parentheses blank. The parameters whose values are passed to the method are called as **actual parameters** and the variables in which the parameter values are received are called as **formal parameters**. **Number of formal parameters should always be equal to number of actual parameters.**
- ▶ **Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.



Methods in Java (Static Method Examples)

```
import java.util.*;

class Add{

    public static void main(String args[])
    {
        int a,b,sum;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two nos.");
        a=sc.nextInt();
        b=sc.nextInt();
        sum=add(a,b);

        System.out.println("The Sum is : "+sum);
    }

    public static int add(int x, int y)
    {
        return(x+y);
    }
}
```

```
import java.util.*;

class Add {

    public static void main(String args[])
    {
        int a,b,sum;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two nos.");
        a=sc.nextInt();
        b=sc.nextInt();
        add(a,b);
    }

    public static void add(int x, int y)
    {
        System.out.println("The Sum is : "+(x+y));
    }
}
```

```
F:\Java>javac Add.java
F:\Java>java Add
Enter two nos.
5
8
Sum is : 13
```

Recursive Methods

- ▶ a method that calls itself is known as a recursive method.
- ▶ And, this process is known as recursion.

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

Normal Method Call

Recursive Call

- ▶ In order to stop the recursive call, we need to provide some conditions inside the method. Otherwise, the method will be called infinitely.
- ▶ Hence, we use the if...else statement (or similar approach) to terminate the recursive call inside the method.

Recursive Methods

```
import java.util.*;

class Factorial{

public static void main(String[] args){
int n, f;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the
number");

n=sc.nextInt();

f=fact(n);

System.out.println("factorial of "+n+" is
: "+f);

}

public static int fact(int x)
{
    if(x==1)
```

```
        return 1;
    else
        return(x*fact(x-1));
    }
}
```

```
F:\Java>java Factorial
Enter the number
5
factorial of 5 is : 120

F:\Java>
```


Java Access Specifiers

Access Modifier ↓ Access Location →	Public	Protected (subclass and pkg)	Default (friendly) (in same pkg)	Private	Private Protected (class & its sub class)
Same Class	Yes	Yes	Yes	Yes	Yes
Sub class in same package	Yes	Yes	Yes	No	Yes
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	No	Yes
Non-subclasses in other packages	Yes	No	No	No	No

Java Class

- ▶ A class is a group of objects which have common properties.
- ▶ It is a template or blueprint from which objects are created.
- ▶ It is a logical entity. It can't be physical.
- ▶ A class in Java can contain:

- ▶ **Fields**

- ▶ **Methods**

- ▶ **Constructors**

- ▶ **Blocks**

- ▶ **Nested class and interface**

- ▶ Syntax to declare a class:

```
class <class_name>{  
    field;  
    method;  
}
```

Java Objects

- ▶ An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.
- ▶ It can be physical or logical (tangible and intangible)
- ▶ An object has three characteristics:
 - ▶ **State:** represents the data (value) of an object.
 - ▶ **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
 - ▶ **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.
- ▶ **An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.
- ▶ **Object Definitions:**
 - ▶ An object is a *real-world entity*.
 - ▶ An object is a *runtime entity*.
 - ▶ The object is *an entity which has state and behavior*.
 - ▶ The object is *an instance of a class*.

Java Objects

- ▶ We can make objects of the class and memory space will be allocated to the fields of that object.
- ▶ The methods of a class can be accessed using the objects using the dot(.) operator.
- ▶ Syntax of making an object of a class is –

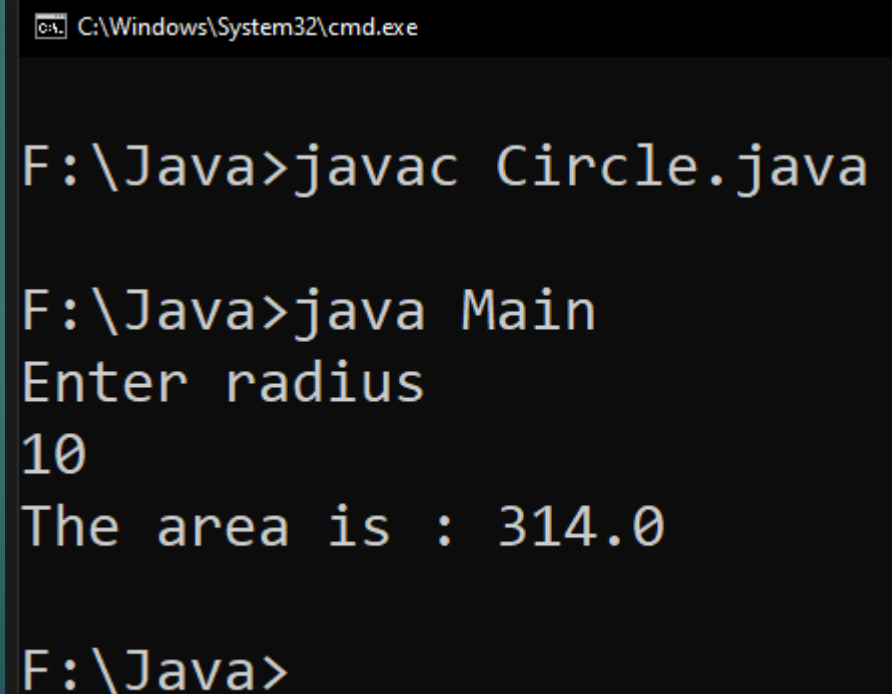
`class_name object_name = new class_name(parameters to be passed)`

Eg : WAP to create a class Circle. It should have three methods namely accept radius, calculate area and display the area.

```
import java.util.*;

class Circle{
    private float r,area;
    public void accept(float x)
    {
        r=x;
    }
    public void calculate()
    {
        area=3.14f*r*r;
    }
    public void display()
    {
        System.out.println("The area is :
"+area);
    }
}
```

```
class Main{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter radius");
        rad=sc.nextFloat();
        Circle c=new Circle();
        c.accept(rad);
        c.calculate();
        c.display();
    }
}
```



C:\Windows\System32\cmd.exe

```
F:\Java>javac Circle.java

F:\Java>java Main
Enter radius
10
The area is : 314.0

F:\Java>
```

Constructor

- ▶ A constructor in Java is a **special method** that is used to initialize objects.
- ▶ The constructor is called when an object of a class is created.
- ▶ It can be used to set initial values for object attributes
- ▶ At the time of calling constructor, memory for the object is allocated
- ▶ Every time an object is created using the new() keyword, at least one constructor is called.
- ▶ It calls a default constructor if there is no constructor available in the class.
- ▶ In such case, Java compiler provides a default constructor by default.
- ▶ There are two types of constructors in Java: no-arg(i.e. default) constructor, and parameterized constructor.
- ▶ It is called constructor because it constructs the values at the time of object creation.
- ▶ It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Constructor

- ▶ Rules for creating constructors :
 - ▶ Constructor should always be public/default/protected.
 - ▶ Name of the constructor should always be same as that of the class
 - ▶ Constructor should not have a return type, **not even void**.
 - ▶ Constructor is automatically called when an object of the class is created
 - ▶ There can be more than one constructor having same name but different parameter list. This is known as **Constructor Overloading**
- ▶ There are the following reasons to use constructors:
 - ▶ We use constructors to initialize the object with the default or initial state. The default values for primitives may not be what are you looking for.
 - ▶ Another reason to use constructor is that it informs about dependencies. In other words, using the constructor, we can request the user of that class for required dependencies.
 - ▶ We can find out what it needs in order to use this class, just by looking at the constructor.
- ▶ In short, we use the constructor to **initialize the instance variable of the class**.

Default Constructor Example

```
import java.util.*;

class CircleDefConst{
    private float r,area;
    CircleDefConst()
    {
        Scanner sc=new
Scanner(System.in);
        System.out.println("Enter Radius");
        r=sc.nextFloat();
    }

    public void calculate()
    {
        area=3.14f*r*r;
    }

    public void display()
    {
        System.out.println("The area is :
"+area);
    }
}
```

```
class Main{
    public static void main(String args[])
    {
        // Default Constructor
        CircleDefConst c=new CircleDefConst();
        c.calculate();
        c.display();
    }
}
```

```
F:\Java>java Main
Enter radius
10
The area is : 314.0

F:\Java>
```

Parameterized Constructor Example

```
import java.util.*;

class Circle{
    private float r,area;

    Circle(float x)
    {
        r=x;
    }

    void calculate()
    {
        area=3.14f*r*r;
    }

    void display()
    {
        System.out.println("The area is : "+area);
    }
}
```

```
class Main{
    public static void main(String args[])
    {
        float rad;

        Scanner sc=new
Scanner(System.in);

        System.out.println("Enter radius");
        rad=sc.nextFloat();

        // parameterized constructor
        Circle c=new Circle(rad);
        c.calculate();
        c.display();
    }
}
```

```
F:\Java>java Main
Enter radius
10
The area is : 314.0

F:\Java>
```

Constructor Overloading

- ▶ Multiple constructors of same class with different parameters is called **constructor overloading**
- ▶ Eg : Program on next slide (slide 18)

<pre>import java.util.Scanner; class Student { private String name; private int roll_no; //parameterized constructor Student(String n, int rn) { name=n; roll_no=rn; } //Default constructor Student() { name = "John"; roll_no = 1; } }</pre>	<pre>//method for printing the values void show() { System.out.println("Name of the student: "+name); System.out.println("Roll No of the student: "+roll_no); } } class Main{ public static void main(String args[]) { Scanner sc = new Scanner(System.in); System.out.println("Enter the name of the student: "); String fname = sc.nextLine(); System.out.println("Enter the roll no of the student: "); int rno = sc.nextInt(); } }</pre>	<pre>//Calling the parameterized constructor System.out.println("calling Show() method for the parameterized constructor: "); Student s=new Student(fname, rno); s.show(); //Calling the default constructor System.out.println("Show() method for the default constructor: "); Student s1=new Student(); s1.show(); } }</pre> <pre>F:\Java>javac Student.java F:\Java>java Main Enter the name of the student: abc Enter the roll no of the student: 5 calling Show() method for the parameterized constructor: Name of the student: abc Roll No of the student: 5 Show() method for the default constructor: Name of the student: John Roll No of the student: 1 F:\Java></pre>
---	--	---

Method Overloading

- ▶ Multiple methods with same name in same class or in base class and derived class with different parameters is called **method overloading**
- ▶ Eg1 : WAP to demonstrate Method Overloading by overloading the methods for calculating Area of circle, rectangle and triangle

```
class MethodOverloading{
private float a;
public void area(float r)
{
    a=3.14f*r*r;
    System.out.println("Area of circle is :"+a);
}
public void area(float l, float b)
{
    a=l*b;
    System.out.println("Area of rectangle is :"+a);
}
public void area(float s1, float s2, float s3)
{
    float s;
    s=0.5f*(s1+s2+s3);
    s=s*(s-s1)*(s-s2)*(s-s3);
    a=(float)(Math.sqrt(s));
    System.out.println("Area of triangle is :"+a);
}}
```

```
class Main {
public static void main(String args[]){
    MethodOverloading obj=new MethodOverloading();
    obj.area(10);
    obj.area(10,10);
    obj.area(10,10,10);
}}
```

```
F:\Java>javac MethodOverloading.java
```

```
F:\Java>java Main
```

```
Area of circle is :314.0
```

```
Area of rectangle is :100.0
```

```
Area of triangle is :43.30127
```

```
F:\Java>
```

Method Overloading

- ▶ Eg2 : WAP to demonstrate Method Overloading in a base and derived class by overloading the methods for displaying the value of variable contained in the class.

```
class Parent{
public void display(int x)
{
    System.out.println("Value of x is :"+x);
}
}
class Child extends Parent{
public void display(int x, int y)
{
    System.out.println("Value of x is :"+x+"\nValue of y is :"+y);
}
}
```

```
class Main {
public static void main(String args[]){
    Child c=new Child();
    c.display(10);
    c.display(5,5);
}}
```

```
F:\Java>javac MethodOverloading.java
```

```
F:\Java>java Main
```

```
Value of x is :10
```

```
Value of x is :5
```

```
Value of y is :5
```

```
F:\Java>
```

Array of Objects

- ▶ Array of objects is created to store information about multiple objects having similar functionality.
- ▶ Syntax : `class_name object_name[] = new class_name[size of the array]`
- ▶ Eg : `Student s[]=new Student[10];`
- ▶ Note: memory is not allocated to each object with the above declaration. We need to use “new” keyword with every object of the array separately for allocating memory.
- ▶ Problem : WAP to create an array of objects of a class student. The class should have field members name, id, total marks and marks in three subjects namely Physics, Chemistry and Maths. Accept information of ‘n’ students and display them in tabular form.


```

import java.util.Scanner;
class Student
{
private String name;
private int id,p,c,m,t;
void accept()
{
Scanner sc=new Scanner(System.in);
System.out.println("Enter name, ID, Marks in Physics,
Chemistry and Maths : ");
name=sc.nextLine();
id=sc.nextInt();
p=sc.nextInt();
c=sc.nextInt();
m=sc.nextInt();
t=p+c+m;
}
void display()
{
System.out.println(name+"\t"+id+"\t"+p+"\t"+c+"\t"+m+"\t"
+t);
}
}

```

```

class Main{
public static void main(String args[])
{
int i,n;
Scanner sc = new Scanner(System.in);
System.out.println("Enter total number of students: ");
n = sc.nextInt();

//creating array of objects of size n

Student s[]=new Student[n];

for(i=0;i<=n-1;i++)
{
    System.out.println("For student "+(i+1));
    s[i]=new Student();
    s[i].accept();
}
System.out.println("Name\tID\tPhy\tChem\tMaths\tTotal");
for(i=0;i<=n-1;i++)
{
    s[i].display();
}
}
}

```

C:\Windows\System32\cmd.exe

F:\Java>javac Array_of_objects.java

F:\Java>java Main

Enter total number of students:

2

For student 1

Enter name, ID, Marks in Physics, Chemistry and Maths :

John

1

50

60

40

For student 2

Enter name, ID, Marks in Physics, Chemistry and Maths :

Mary

2

50

55

70

Name	ID	Phy	Chem	Maths	Total
------	----	-----	------	-------	-------

John	1	50	60	40	150
------	---	----	----	----	-----

Mary	2	50	55	70	175
------	---	----	----	----	-----

F:\Java>

Static Class Members

- ▶ A method or a field can be static.
- ▶ **static** keyword is used to declare class members as static
- ▶ A static method is a method that works on a class and not on the object of the class.
- ▶ To call a static method we **need not** make object of that class
- ▶ A static method is called by syntax : `class_name.method_name()`
- ▶ A static field member or a static variable will be common for all the objects of that class.
- ▶ A common memory location is allocated for a static variable for all objects made of that class
- ▶ One of the major advantage of using static variable is that we can use it to count number of objects made of the class. Since each object will access the same memory location for static variable, we can increment its value in the constructor of class.
- ▶ Whenever the object is created, this constructor will be called and this static variable will be incremented.

```
import java.util.Scanner;
class Counter
{
private static int count;
Counter()
{
count++;
}
static void display()
{
System.out.println("Count="+count);
}
}
```

```
class Main{
public static void main(String args[])
{
Counter c1=new Counter();
Counter.display();
Counter c2=new Counter();
Counter.c3=new Counter();
Counter.display();
Counter c4=new Counter();
Counter.c5=new Counter();
Counter.display();
}
}
```

C:\Windows\System32\cmd.exe

F:\Java>javac Counter.java

F:\Java>java Main

Count=1

Count=3

Count=5

F:\Java>

“this” keyword

- ▶ **this** keyword refers to the current object.
- ▶ **this** can also be used to:
 - ▶ Invoke current class constructor
 - ▶ Invoke current class method
 - ▶ Return the current class object
 - ▶ Pass an argument in the method call
 - ▶ Pass an argument in the constructor call

Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01

this can be used to refer current class instance variable.

04

this can be passed as an argument in the method call.

02

this can be used to invoke current class method (implicitly)

05

this can be passed as argument in the constructor call.

03

this() can be used to invoke current class Constructor.

06

this can be used to return the current class instance from the method

Eg1 : Without this keyword

```
class Student1{
int rollno;
String name;
float fee;
Student1 (int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class Main{
public static void main(String args[]){
Student1 s1=new Student1 (111,"ankit",5000f);
Student1 s2=new Student1 (112,"sumit",6000f);
s1.display();
s2.display();
}}
```

```
F:\Java>javac Student1.java
```

```
F:\Java>java Main
0 null 0.0
0 null 0.0
```

```
F:\Java>
```

Eg2 : With this keyword

```
class Student1{
int rollno;
String name;
float fee;
Student1 (int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class Main {
public static void main(String args[]){
Student1 s1=new Student1 (111,"ankit",5000f);
Student1 s2=new Student1 (112,"sumit",6000f);
s1.display();
s2.display();
}}
```

```
F:\Java>javac Student1.java
```

```
F:\Java>java Main
111 ankit 5000.0
112 sumit 6000.0
```

```
F:\Java>
```

In the above eg1, parameters (formal arguments) and instance variables are same. So, we are using this keyword in eg2 to distinguish local variable and instance variable.

Programming Practice Questions

- ▶ Create a class Employee with data members *emp_id*, *name*, *designation* and *salary*. Write methods *get_details()* to take employee details from the user and *show_grade()* to display grade of employees based on salary range as given below and *show_details()* to display employee details.

Salary Range	Grade
<10000	D
10000-24999	C
25000-49999	B
>50000	A

- ▶ Create a class student with appropriate data members and methods to store and display Roll No, Name and marks stored by students in Physics, Chemistry and Maths.
- ▶ WAP to demonstrate Method Overloading by overloading the method for calculating sum. Consider taking different parameter list in terms of no. of parameters and type of parameters.
- ▶ WAP to demonstrate Constructor Overloading by overloading the constructor for taking radius input for calculating area of a circle.
- ▶ WAP to implement a class Stack using array with two methods push and pop to add and remove elements from the stack. Addition of elements should not be allowed if the stack is full and popping should not be allowed if the stack is empty.



JAVA PROGRAMMING

Chap 3 : Inheritance

- ▶ Inheritance is an important pillar of OOP(Object-Oriented Programming).
- ▶ It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- ▶ In Java, inheritance means creating new classes based on existing ones.
- ▶ A class that inherits from another class can reuse the methods and fields of that class.
- ▶ In addition, you can add new fields and methods to your current class as well.
- ▶ Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- ▶ Important terminologies used in Inheritance:
 - ▶ **Class:** Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
 - ▶ **Super Class/Parent Class/Base Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
 - ▶ **Sub Class/Child Class/Derived Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
 - ▶ **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

► Why do we need Inheritance in Java?

- **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which java achieves Run Time Polymorphism.
- **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

► Syntax for inheritance

- The **extends keyword** is used for inheritance in java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality.

- Syntax:

```
class derived_class extends base_class
{
    //methods and fields
}
```

► Types of Inheritance :

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

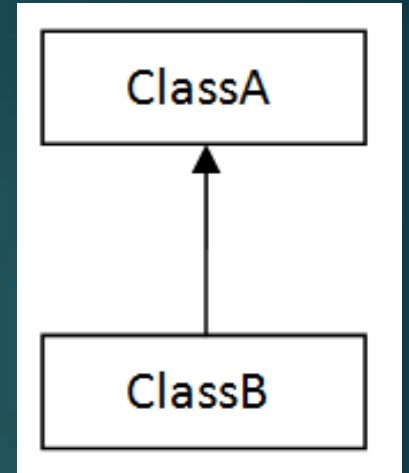
Single Inheritance

Here only one class is derived from another class.

```
import java.util.*;
class Data {           // Base class Data
    float r;
    public void read(float x)
    {
        r=x;
    }
}
class Area extends Data // creating child
class Area
{
    private float a;
    public void calculate()
    {
        a=3.14f*r*r;
    }
    public void display()
    {
        System.out.println("Area is : "+a);
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Radius : ");
        rad=sc.nextFloat();
        Area ar=new Area();
        ar.read(rad);
        ar.calculate();
        ar.display();
    }
}
```

```
Enter Radius :
10
Area is : 314.0
```



Multilevel Inheritance

Here one class is derived from a class which is in turn derived from another class.

```
import java.util.*;
class Data {           // Base class Data
    float r;
    public void read(float x)
    {
        r=x;
    }
}
```

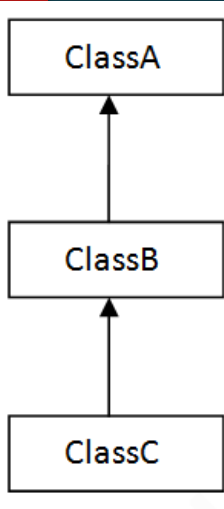
```
class Area extends Data    // creating
child class Area of parent class Data
{
    protected float a;
    public void calculate()
    {
        a=3.14f*r*r;
    }
    public void display()
    {
        System.out.println("Area is : "+a);
    }
}
```

```
class Volume extends Area  // creating child
class Volume of parent class Area
{
    private float vol;
    public void compute()
    {
        vol=a*r*4/3;
    }
    public void show()
    {
        System.out.println("Volume is : "+vol);
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Radius : ");
        rad=sc.nextFloat();
    }
}
```

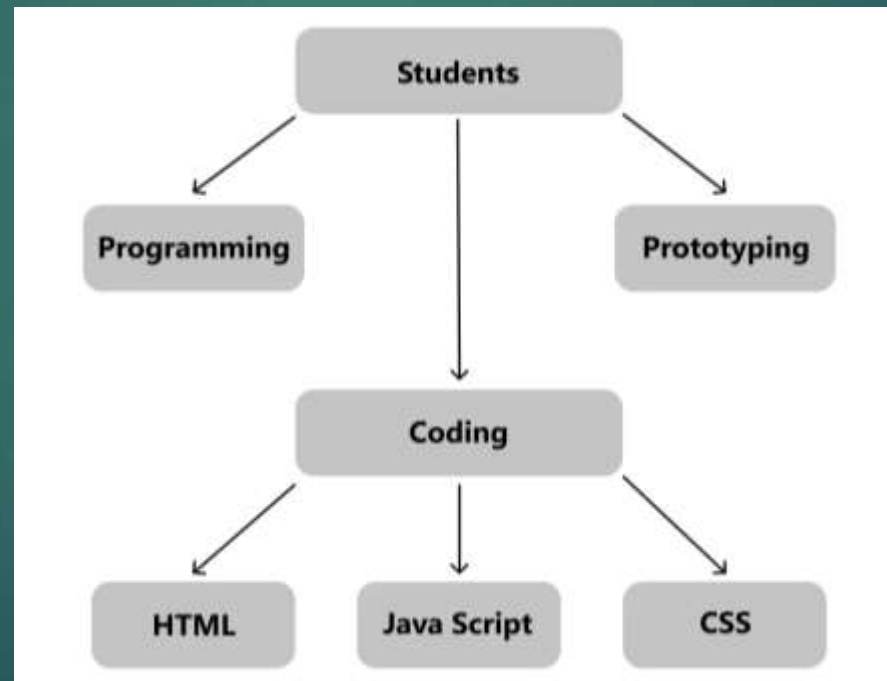
```
Volume v=new Volume();
v.read(rad);
v.calculate();
v.display();
v.compute();
v.show();
}
```

```
Enter Radius :
10
Area is : 314.0
Volume is : 4186.6665
```



Hierarchical Inheritance

- ▶ When multiple classes are derived from a class and further more classes are derived from these derived classes
- ▶ one class serves as a superclass (base class) for more than one subclass
- ▶ Example from codes



Hierarchical Inheritance

// parent class

```
class Employee {  
    double salary = 50000;  
  
    void displaySalary() {  
        System.out.println("Employee Salary:  
Rs."+salary);  
    }  
}
```

// child class 1

```
class FullTimeEmployee extends  
Employee{  
    double hike = 0.50;  
  
    void incrementSalary() {  
        salary = salary + (salary * hike);  
        System.out.println("Full Time Employee  
Salary after increment : "+salary);  
    }  
}
```

// child class 2

```
class InternEmployee extends Employee{  
    double hike = 0.25;  
  
    void incrementSalary() {  
        salary = salary + (salary * hike);  
        System.out.println("Intern Salary after increment : "+salary);  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        // object created  
        FullTimeEmployee emp1 = new FullTimeEmployee();  
        InternEmployee emp2 = new InternEmployee();  
        System.out.println("Salary of a full-time employee before  
incrementing:");  
        emp1.displaySalary();  
        emp1.incrementSalary(); // salary incremented of Full time Employee  
        System.out.println("Salary of an intern before incrementing:");  
        emp2.displaySalary();  
        emp2.incrementSalary(); // salary incremented of Intern  
    }  
}
```


Hierarchical Inheritance

```
Salary of a full-time employee before incrementing:  
Employee Salary: Rs.50000.0  
Full Time Employee Salary after increement : 75000.0  
Salary of an intern before incrementing:  
Employee Salary: Rs.50000.0  
Intern Salary after increement : 62500.0
```

Method Overriding

- ▶ If a base class and derived class have a method with same name but different parameter list, then it is called as method overloading.
- ▶ But, **If a base class and derived class have a method with same name and same parameter list, then it is called as method overriding.**
- ▶ Usage of Java Method Overriding
 - ▶ Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
 - ▶ Method overriding is used for runtime polymorphism
- ▶ Rules for Java Method Overriding
 - ▶ The method must have the same name as in the parent class
 - ▶ The method must have the same parameter as in the parent class.
 - ▶ There must be an IS-A relationship (inheritance).

Method Overriding

```
class Bank{  
int getRateOfInterest()  
{  
return 0;  
}}
```

```
class SBI extends Bank{  
int getRateOfInterest()  
{  
return 8;  
}}
```

```
class ICICI extends Bank{  
int getRateOfInterest()  
{  
return 7;  
}}
```

```
class AXIS extends Bank{  
int getRateOfInterest()  
{  
return 9;  
}}
```

```
class Main{  
public static void main(String args[]){  
SBI s=new SBI();  
ICICI i=new ICICI();  
AXIS a=new AXIS();  
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());  
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());  
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());  
}}
```

```
SBI Rate of Interest: 8  
ICICI Rate of Interest: 7  
AXIS Rate of Interest: 9
```

Final class and “final” keyword

- ▶ The final keyword can be preceded to any member of a class or to the class itself.
- ▶ Making anything final has following implications –
 - ▶ If the field member is declared final then the variable value cannot be changed i.e. it becomes constant.
 - ▶ If a method is declared as final then that method cannot be overridden
 - ▶ If a class is declared as final then that class cannot have any sub class i.e. no class can be derived from a final class

Final class and “final” keyword

- ▶ **Final variable** : If you make any variable as final, you cannot change the value of final variable(It will be constant).
- ▶ Example of final variable : There is a final variable speedlimit, we are going to change the value of this variable, but it can't be changed because final variable once assigned a value can never be changed.

```
class Bike{
    final int speedlimit=90; //final variable
    void run(){
        speedlimit=400;      // changing value of speedlimit
    }
}
class Main{
    public static void main(String args[]){
        Bike b=new Bike();
        b.run();
    }
}
```

```
Main.java:4: error: cannot assign a value to final variable speedlimit
    speedlimit=400;    // changing value of speedlimit
    ^
1 error
```

Final class and “final” keyword

- **Final method** : If you make any method as final, you cannot override it.

```
class Bike{  
    final void run()  
    {  
        System.out.println("running");  
    }  
}
```

```
class Honda extends Bike{  
    void run()          // overriding final method run()  
    {  
        System.out.println("running safely with 100kmph");  
    }  
}
```

```
class Main{  
    public static void main(String args[]){  
        Honda h= new Honda();  
        h.run();  
    }  
}
```

```
Main.java:10: error: run() in Honda cannot override run() in Bike  
    void run()          // overriding final method run()  
        ^  
    overridden method is final  
1 error
```

Final class and “final” keyword

- **Final class** : If you make any class as final, you cannot extend it.

```
final class Bike{
}

class Honda extends Bike{           // inheriting from final class bike
    void run()
    {
        System.out.println("running safely with 100kmph");
    }
}

class Main{
    public static void main(String args[]){
        Honda h= new Honda();
        h.run();
    }
}
```

```
Main.java:4: error: cannot inherit from final Bike
    class Honda extends Bike{
                        ^
1 error
```


Java Abstract class and method

- ▶ Abstract classes are used to declare common characteristics of subclasses.
- ▶ Abstract classes are declared with keyword ***abstract*** followed by class definition. They provide template for subclasses.
- ▶ ***No object can be made of abstract class. It can be used as a base class for other classes that are derived from the abstract class.***
- ▶ An abstract class can contain fields and methods. It can have abstract and non-abstract methods (method with the body).
- ▶ ***Methods of abstract class that has only declaration and no definition are known as Abstract Methods.***
- ▶ ***Abstract methods must be overridden.***
- ▶ If a class has any abstract method, the class becomes abstract and must be declared as abstract.

Rules for Java Abstract class



Java Abstract class and method

```
import java.util.*;

abstract class Abst {    //abstract class
    protected float r,vol;
    public void read(float x) // non-abstract method
    {
        r=x;
    }
    public abstract void calculate(); // abstract method
    public void display()
    {
        System.out.println("Volume = "+vol);
    }
}

class Sphere extends Abst {
    public void calculate() // overriding abstract
    method
    {
        vol=3.14f*r*r*r*4/3;
    }
}

class Hemisphere extends Abst {
    public void calculate() // overriding abstract method
    {
        vol=3.14f*r*r*r*2/3;
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Radius : ");
        rad=sc.nextFloat();
        Sphere s=new Sphere();
        s.read(rad);
        s.calculate();
        System.out.println("For Sphere : ");
        s.display();
        Hemisphere h=new Hemisphere();
        h.read(rad);
        h.calculate();
        System.out.println("For Hemisphere : ");
        h.display();
    }
}
```

```
Enter Radius :
10
For Sphere :
Volume = 4186.6665
For Hemisphere :
Volume = 2093.3333
```

The *super* keyword

- ▶ The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- ▶ If you want to access a member of a base class from the derived class, then the **super** keyword is used.
- ▶ This is especially to access the constructors and methods of base class.
- ▶ Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Super Keyword

1

Super can be used to refer immediate parent class instance variable.

2

Super can be used to invoke immediate parent class method.

3

super() can be used to invoke immediate parent class constructor.

The *super* keyword

```
class Parent{
public Parent(int a)
{
    System.out.println("Inside the
constructor of Parent class : "+a);
}

public void display(int x)
{
    System.out.println("x = "+x);
}
}
```

```
class Child extends Parent{
public Child(int a)
{
    super(a);
    System.out.println("Inside the
constructor of Child class : "+a);
}

public void display(int y)
{
    super.display(y);
    System.out.println("y = "+y);
}
}
```

```
class Main{
public static void main(String args[])
{
    Child c = new Child(3);
    c.display(5);
}
}
```

```
Inside the constructor of Parent class : 3
Inside the constructor of Child class : 3
x = 5
y = 5
```


Programming Practice Questions

Design a class hierarchy for a library system. Create a base class *Book* with attributes *title* and *author*, and a constructor that initializes these attributes. Then, create a derived class *Ebook* that extends *Book* with additional attributes *file_size* and *format* and *Printed_book* that extends *Book* with attributes *edition* and *no of copies sold*. Ensure that the constructor of *Ebook* and *Printed_book* properly initializes attributes from both the base class and the derived class. (use *super* keyword)

Create an abstract class *Appliance* with abstract methods *switch_on()* and *switch_off()*. Derive two classes *WashingMachine* and *Refrigerator* from *Appliance* and implement the abstract methods with specific behaviors for each appliance. Ensure that the abstract class cannot be instantiated directly and that the derived classes provide concrete implementations for the abstract methods.

Design a class hierarchy for a university system. Create a base class *Person* with attributes *name* and *age*, and methods like *getDetails()* and *display_details()*. Then, create derived classes *Student* and *Professor* that inherit from *Person* and add specific attributes and methods. For example, *Student* might have a *studentID* and *major* field, while *Professor* might have a *facultyID*, *salary* and *department*. Override *getDetails()* and *display_details()* methods to input and display all the details of student and professor.

Implement a class hierarchy for a banking system. Create a base class *Account* with attributes *accountNumber* and *balance*, a constructor to initialize these attributes and methods *deposit()*, *withdraw()* and *getBalance()*. Create a derived class *SavingsAccount* that adds an *interestRate* attribute and provides its own constructor that initializes all attributes. Also calculate the interest and add it to the balance using deposit method().

Programming Practice Questions

- ▶ You are tasked with designing a system for a university that includes different types of people: *Professor*, *Student*, and *Staff*. All these types of people share some common attributes but also have their unique attributes. You need to use *hierarchical inheritance* to model this system, ensuring that constructors are properly used to initialize the base and derived classes. Create a Base Class *Person* having Fields *name* and *id* and a constructor to initialize them. It will also have *getdetails()* and *display()* methods to input and display the details. Create Derived Classes *Professor*, *Student* and *Staff* that inherits from *Person*. *Professor* has an additional field *department*. Initialize *name*, *id*, and *department* using the base class constructor. Class *Student* has additional field *major* and *gpa*. Initialize *name*, *id*, *major*, and *gpa* using the base class constructor. Class *Staff* has additional field *designation*. Initialize *name*, *id*, and position using the base class constructor. Create instances of *Professor*, *Student*, and *Staff* classes and print their details using *display()* method.
- ▶ Design a class Hierarchy for a zoo. Create a base class *Animal* which has data members *Name* and *Species*. It has methods *getHabitat()* to return default habitat description and *display()* to display the details of the animal. Class *Mammal* is derived from *Animal* which has field *hasFur* of type *Boolean*. A class *Lion* inherits class *Mammal* and has data member *endangered* of type *Boolean*. Override *display()* and *getHabitat()* methods to display animal details and Habitat description of *Mammal* as well as *Lion*. Use appropriate constructors to initialize all the data members.
- ▶ Design an abstract class *Character* for a game to serve as the base class for various types of characters. Each character type has specific actions but shares common functionality such as *name*, *attacking* and *defending*. This class accepts the *character_name* through constructor and has abstract methods *attack()* and *defend()* that defines how the character attacks or defends. Class *Warrior* and *Mage* inherits class *Character* and overrides *attack()* method to display what weapon is used by the character to attack (eg: sword, fireball, axe, javelin, etc.) and *defend()* method to display the technique used by the character to defend (eg: uses shield, uses magic spell, counter attack, etc.).