

**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES****Department of ECE****Object Oriented Programming****SYLLABUS:****UNIT-1:**

**Introduction to OOPS:** Paradigms of Programming Languages, Basic concepts of Object Oriented Programming, Differences between Procedure Oriented Programming and Object Oriented Programming, Objects and Classes, Data abstraction and Encapsulation, Inheritance, Polymorphism, benefits of OOP , application of OOPs.

**Java :** History, Java features, Java Environment, JDK, API.

**Introduction to Java :** Types of java program, Creating and Executing a Java program, Java Tokens, Keywords, Character set, Identifiers, Literals, Separator, Java Virtual Machine (JVM), Command Line Arguments, Comments in Java program.

**UNIT -2:**

**Elements:** Constants, Variables, Data types, Scope of variables, Type casting, Operators: Arithmetic, Logical, Bit wise operator, Increment and Decrement, Relational, Assignment, Conditional, Special operator, Expressions – Evaluation of Expressions

**Decision making and Branching:** Simple if statement, if, else statement, Nesting if, else, else if Ladder, switch statement, Decision making and Looping: While loop, do-While loop, for loop, break, labelled loop, continue Statement, Simple programs

**Arrays:** One Dimensional Array, Creating an array, Array processing, Multidimensional Array, Vectors, Wrapper classes, Simple programs

**UNIT-3:**

**Strings:** Exploring String class, String Class Methods, String Buffer Class, Simple programs

**Class and objects:** Defining a class, Methods, Creating objects, Accessing class members, Constructors, Static members, Nesting of Methods, this keyword, Command line input.

**Polymorphism** – Static Polymorphism, Dynamic Polymorphism, Method overloading, Polymorphism with Static Methods, Private Methods and Final Methods.

**Inheritance:** Defining a sub class, Deriving a sub class, Single Inheritance, Multilevel Inheritance, Hierarchical Inheritance, Overriding methods, Final variables and methods, Final classes, Finalizer methods, Abstract methods and classes, Visibility Control: Public access, Private access, default and protected. Abstract classes, Interfaces - Interfaces vs Abstract classes, defining an interface, implementing interfaces, accessing implementations through interface references, extending interfaces. Inner classes - uses of inner classes, local inner classes, anonymous inner classes, static inner classes, examples.

**UNIT- 4:**

**Packages:** Java API Packages, System Packages, Naming Conventions, Creating & Accessing a Package, Adding Class to a Package, Hiding Classes, Programs

**Exception Handling:** Limitations of Error handling, Advantages of Exception Handling, Types of Errors, Basics of Exception Handling, try blocks, throwing an exception, catching an exception, finally statement

**Multi threading:** Creating Threads, Life of a Thread, Defining & Running Thread, Thread Methods, Thread Priority, Synchronization, Implementing runnable interface, Thread scheduling.

**UNIT-5:**

**AWT Components and Event Handlers:** Abstract window tool kit, Event Handlers, Event Listeners, AWT Controls and Event Handling: Labels, TextComponent, ActionEvent, Buttons, CheckBoxes, ItemEvent, Choice, Scrollbars, Layout Managers- Input Events, Menus, Programs

**GUI Programming with Java -** Introduction to Swing, limitations of AWT, Swing vs AWT, MVC architecture, Hierarchy for Swing components, Containers - JFrame, JApplet, JDialog, JPanel. Overview of some swing components JButton, JLabel, JTextField, JTextArea, simple swing applications.

**I/O Streams:** File, Streams, Advantages, The stream classes, Byte streams, Character streams.

**JDBC, ODBC Drivers:** JDBC ODBC Bridges, Seven Steps to JDBC, Importing java SQL Packages, Loading & Registering the drivers, Establishing connection. Creating & Executing the statement.

**TEXT BOOKS:**

1. Java; the complete reference, 7<sup>th</sup> edition, Herbert Schildt, TMH
2. Understanding OOP with Java, updated edition, T. Budd, Pearson Education.

**REFERENCE BOOKS:**

1. An Introduction to programming and OO design using Java, J.Nino and F.A. Hosch, John Wiley & sons
2. Introduction to Java programming, Y. Daniel Liang, Pearson Education
3. An Introduction to Java programming and Object Oriented Application Development, R.A. Johnson
4. Programming with Java - E. Balagurusamy
5. Object oriented Programming in Java - Dr. G.Thampi
6. Let us Java – Yashavant Kanetkar - BPB Publications, New Delhi - First Edition 2012
7. Core Java, An Integrated Approach, Dr. R. Nageswara Rao
8. An Introduction to Ooops with Java - C Thomas WU - TataMc-Graw Hill, New Delhi - 4th Edition
9. Object oriented Programming through Java - ISRD Group - TataMc-Graw Hill, New Delhi - Eight Reprint

**Expected Outcome:**

The Student is expected to have

- ✓ Understanding of OOP concepts and basics of Java Programming (Console and GUI based ).
- ✓ The skills to apply OOP and Java programming in Problem solving.
- ✓ Should have the ability to extend his/her knowledge of Java programming further on his/her own.

**Recommended System/Software Requirements :**

- Software : j2sdk1.7. , Eclipse or Netbeans      Operating System : Windows / Linux

**\*\*\* ALL THE GOOD LUCK ... Come Lets Start Programming with JAVA\*\*\***

**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES****Department of ECE****Object Oriented Programming****Course Objectives:**

- Covers software design, implementation, and testing using Java. Introduces object-oriented design techniques and problem solving.
- Emphasizes development of secure, well-designed software projects that solve practical real-world problems
- To understand Object oriented programming concepts, and apply them in Problem solving.
- To learn the basics of Java Console and GUI based programming

**Course Outcomes:**

On completion of the course the student should be able to:

- Use an integrated development environment to write, compile, run, and test simple object-oriented Java programs.
- Read and make elementary modifications to Java programs that solve real-world problems.
- Validate input in a Java program.
- Identify and fix defects and common security issues in code.
- Document a Java program using Javadoc.
- Use a version control system to track source code in a project.

**\*\*\* ALL THE GOOD LUCK ... Come Lets Start Programming with JAVA\*\*\***



RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES  
BASAR, NIRMAL, TELANGANA, INDIA

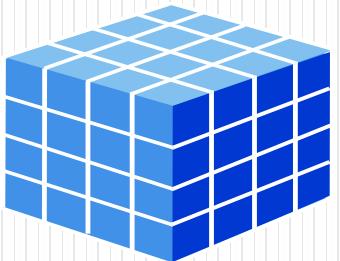


# OBJECT ORIENTED PROGRAMMING USING JAVA

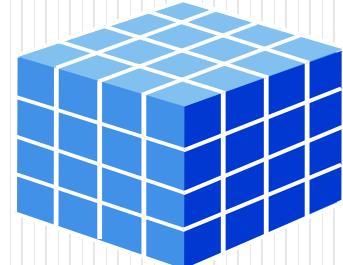
Every Week : Monday, Tuesday & Wednesday

Timings - 11:00 am to 12:00 pm

MOODLE LMS SESSION LINK: <http://lms.rgukt.ac.in/course>



**Mr. K. RAVIKANTH**  
Assistant Professor  
Department of CSE  
RGUKT-BASAR CAMPUS



- **Introduction to OOPS:**

- Paradigms of Programming Languages
- Basic concepts of Object Oriented Programming
- Differences between Procedure Oriented Programming and Object Oriented Programming
- Objects and Classes
- Data abstraction and Encapsulation
- Inheritance
- Polymorphism
- Benefits of OOP
- Application of OOPs.
- Summary

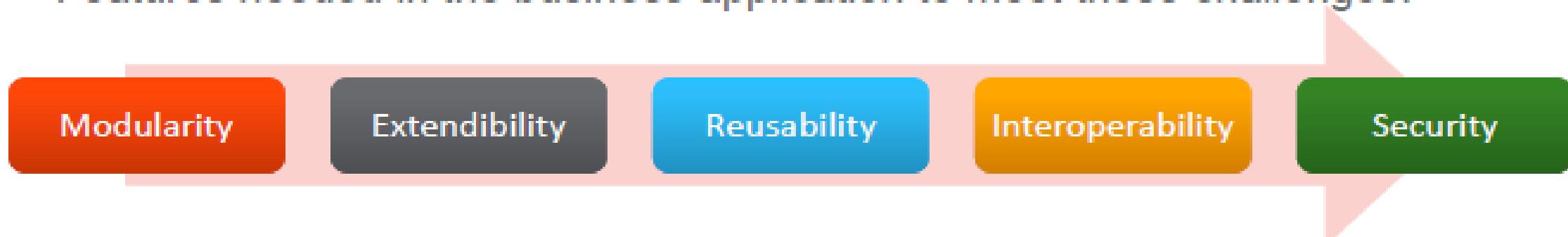
- **Need for Object Oriented Approach**
- To face the Challenges in Developing a business application
  - Integration of Modules/Applications
  - Extensibility of existing code
  - High level of flexibility
  - Illusion of simplicity
- If these challenges are not addressed it may lead to **SOFTWARE CRISIS**
- The term software crisis describes software failure in terms of
  - Exceeding Software Budget
  - Software not meeting client's requirement
  - Bugs in the software
- OOPS is a programming paradigm which deals with the concept of objects to build programs and software applications

## Need for Object Oriented Approach

- Challenges in developing a business application



- If these challenges are not addressed it may lead to **Software Crisis**
- Features needed in the business application to meet these challenges:



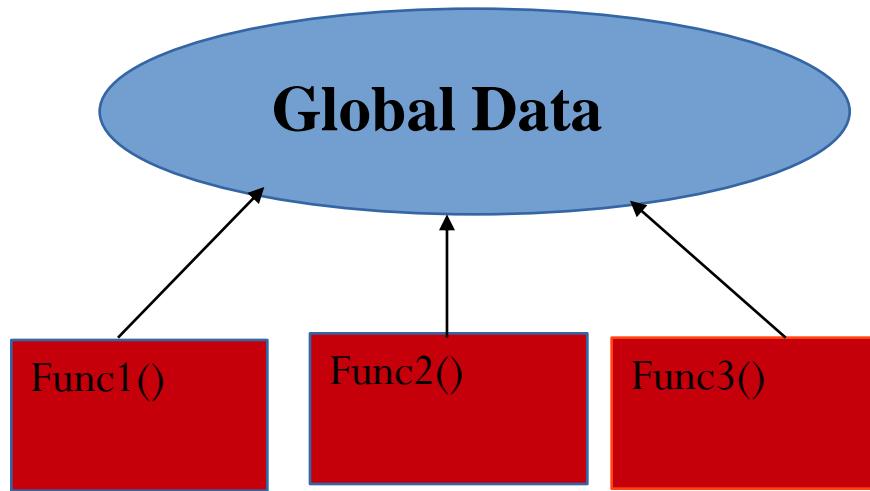
- Challenges can be addressed using object oriented approach

Every Object has a well-defined *identity, attributes and behavior*

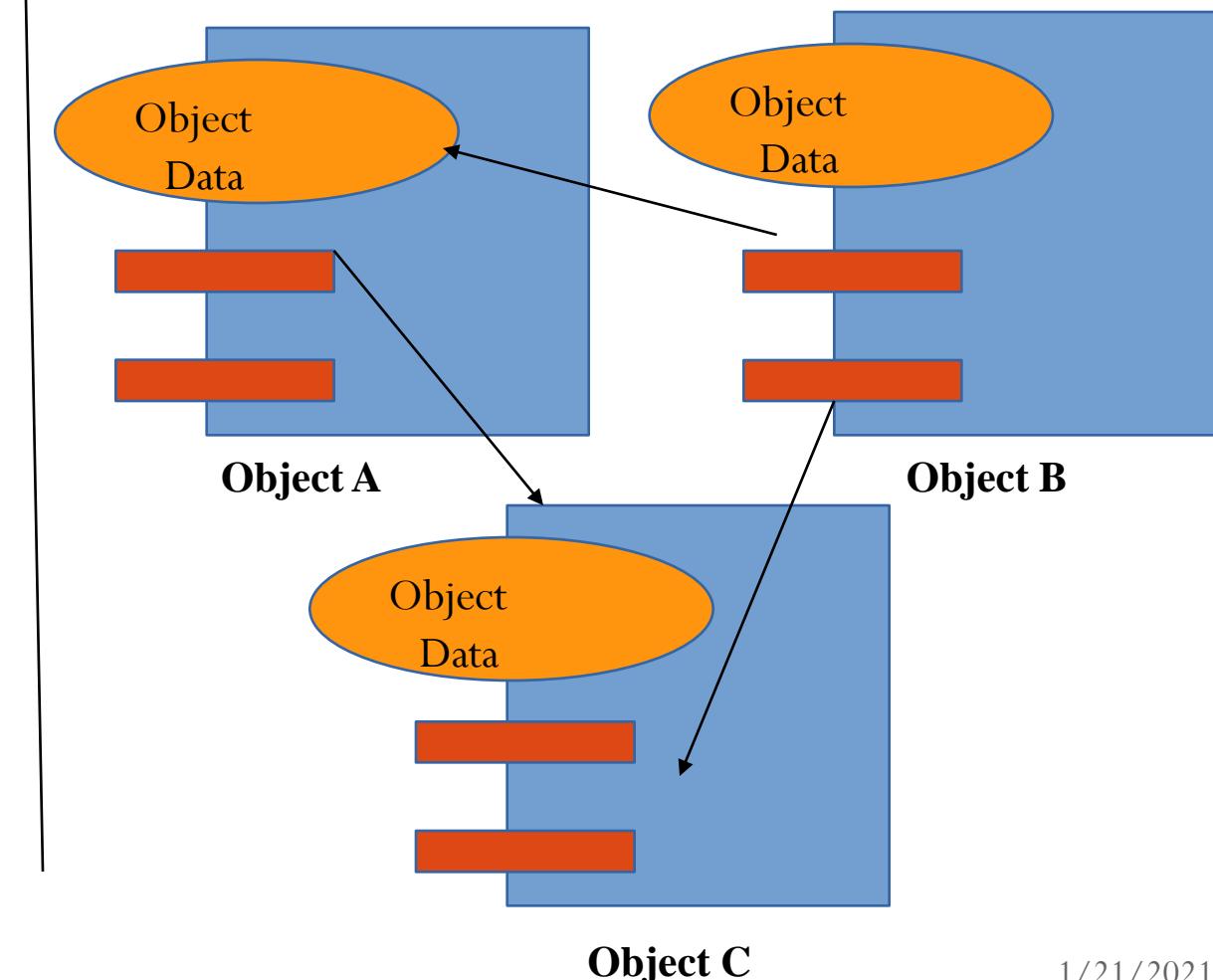
| Programming Style      | Abstraction Employed   |
|------------------------|--|
| Monolithic             | Emphasizes on finding a solution   |
| Procedure Oriented     | Lays stress on Algorithms ( Computation )  |
| Structured             | Focuses on Modules ( Functions or Sub-Programs )                                       |
| <b>Object Oriented</b> | <b>Classes &amp; Objects</b><br><b>(Wide range of Architecture framework together)</b> |
| Logic Oriented         | Goals ( Often expressed in Predicate Calculus)   |
| Rule Oriented          | If-Then-Else Rules ( Design of Knowledge Base )  |
| Constraint Oriented    | Utilizes Invariant Relationship ( Complex Systems )                                    |

OOP is much more similar to the way the real world works; it is analogous to the human brain. Each program is made up of many entities called objects.

## Procedural Programming



## Object Oriented Programming



## Our Analysis goes as...

- In **Procedural Language**, a program is a list of instructions, as the length of the program increases, its complexity increases & becomes difficult to maintain very large programs
- In **Structured Programming Language**, this problem can be overcome by dividing a large program into different functions or modules, in-turn it results in other problems.
- Hence, Large programs can still become increasingly complex
- The two major problems of Procedural Languages are:
  - The functions have unrestricted access to global data
  - They provide poor mapping to the real world
  - It is very difficult to create anew data type or a user defined data type

# Differences between Procedure Oriented Language and Object Oriented Programming

| Procedural Languages                              | Object Oriented Programming                   |
|---|---|
| Separate data from functions that operate on them | Encapsulate data and methods in a class       |
| Not suitable for defining abstract types          | Suitable for defining abstract types          |
| Debugging is difficult                            | Debugging is easier                           |
| Difficult to implement change                     | Easier to manage and implement change         |
| Not suitable for larger programs and applications | Suitable for larger programs and applications |
| Analysis and design not so easy                   | Analysis and design made easier               |
| Faster  | Slower  |

# Differences between Procedure Oriented Language and Object Oriented Programming

| Procedural Languages                    | Object Oriented Programming                                      |
|---|--|
| Less Flexible                           | Highly Flexible  |
| Data and procedure based                | Object Oriented  |
| Less Reusable                           | More Reusable  |
| Only data and procedures are there      | Inheritance, Encapsulation and Polymorphism are the key Features |
| Use Top-Down Approach                   | Use Bottom-up Approach   |
| Only a function calls another           | Object Communication is there                                    |
| Example: C, Basic, FORTRAN, PASCAL etc. | Example: JAVA, PYTHON, C++, C#.NET etc.                          |

## What is a Object-Oriented Programming Language ?

- **Object-oriented programming** is one of the most interesting and useful innovations of Software Development. It addresses the problems commonly known as the **Software Crisis**
- The main aim of object-oriented programming is to implement **real-world entities** for example object, classes, abstraction, inheritance, polymorphism, etc.
- In the **physical world** we deal with Objects like person, plane, car, pen, chair, table, computer, watch, etc.
- **Objects** are defined by their unique **identity, state and behavior**

- **Basic Concepts of Object Oriented Programming**
  - Objects
  - Classes
  - Attributes \ Variables
  - Behavior \ Method
  - Data Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Message Passing
  - Reusability

## Basic concepts of Object Oriented Programming

- **Object Oriented programming** is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.
- If we develop any **business application** according to OOPs principles in any programming Language then we will get the security, flexibility and reusability in those applications.

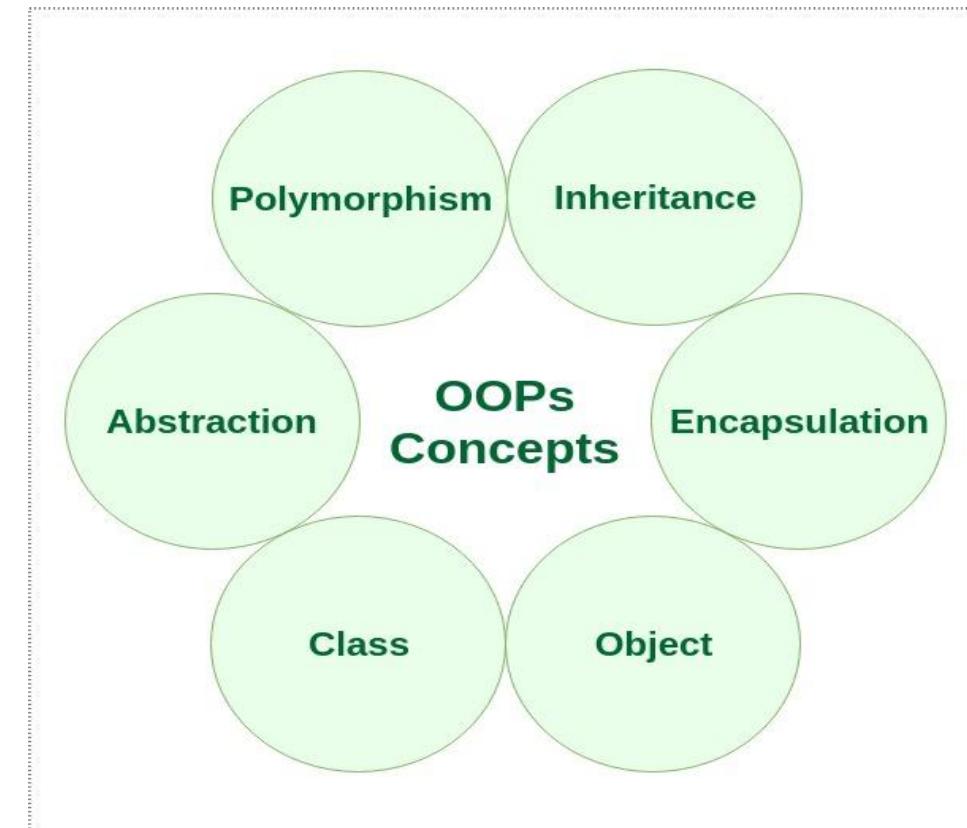


- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- **Object**
- **Class**
- **Data Abstraction**
- **Encapsulation**
- **Inheritance**
- **Polymorphism**

- Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- Coupling
- Cohesion
- Association
- Aggregation
- Composition



**► Object**

- It is a **basic unit** of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :
- **State** : It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity** : It gives a unique name to an object and enables one object to interact with other objects.
- Example of an object : dog



## What is an Object?

**In English:** Object is a material that can be seen and touched.

**In Real World:** Normally Every Object has two properties/ Characteristics which are:

**Properties: 1. State(s) and 2. Behavior(s).**

Examples of Objects: **chair, pen, table, keyboard, bike etc.**

**It can be Physical and Logical**

**Physical entity:** Car, Bus ,Plane

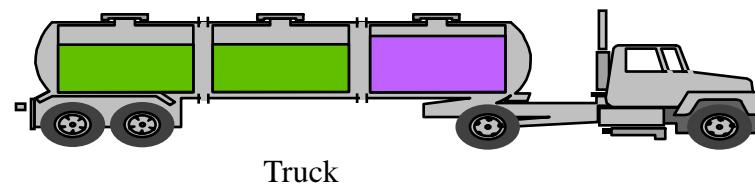
**Creator Objects:** Humans, Employees, Students, Animal

**Objects in Computer System:** Monitor, Keyboard, Mouse, CPU

Objects are defined as the instances of a class.

**Ex:** Table, chair are all instances of the class Furniture

**NOTE:** Objects have Unique identity, state and behavior



Truck

**State(s):** State(s) is a property, is(are) used to uniquely identify the object(s).

**Behavior(s):** is(are) an action(s) performed by the Object.....

**For Example:** Pen is a Object

**State(s):** color, length, name, weight etc.

**Behavior:** writing is an action done by pen.

**Note:** In a Programming, States are nothing but variables

and

Behavior is nothing but methods

## Other Definitions of Object –

Anything that provides a way to locate, access, modify and secure data

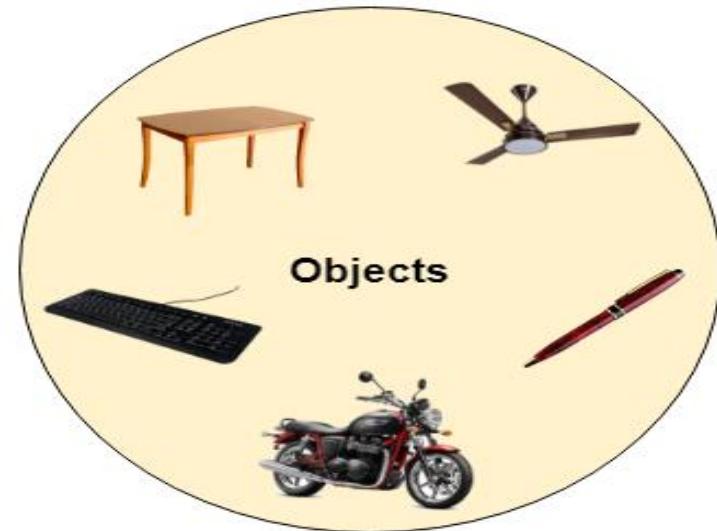
OR

An object is an instance of a class which can be uniquely identified by its name. Every object has a state which is given by its attributes at a particular time

OR

Object is similar to variable, where a variable contains a Small Area , whereas an Object contains a Large Area

- **Note: While a class is a model / logical structure, an object is a physical**



| Object   | Data or Attributes                                | Functions or Methods                          |
|----------|---|---|
| Person   | Name,Age,Sex,Color                                | Speak(),Walk(), Listen(), Write()             |
| Vehicle  | Name, Company, Model, Capacity,Color              | Start(),Stop(),accelerate()                   |
| Polygon  | Vertices, Border, Color                           | Draw(),Erase()                                |
| Account  | Type, Number, Balance                             | Deposit(),Withdraw(),Enquire()                |
| City     | Name, Population, Area, Literacy rate             | Analyse( ),Data( ),Display( )                 |
| Computer | Brand, Resolution, Price                          | Processing(),Display(),Printing()             |
| Customer | C_name,C_Addr,C_Bal,C_acc.no                      | Deposit(),Withdraw(),Transfer(),Bal_enquire() |
| Employee | E_id,E_name,E_add, E_sal                          | Da(),Ta(),Hra(),Pf(),tax(),T_sal()            |
| Bike     | boolean kickStart, boolean buttonStart, int gears | Accelerate( ), applyBreak( ), changeGear( )   |

## CLASS

- **Class** – A class is defined as the blueprint for an object. It serves as a plan or a template. The description of a number of similar objects is called a class

OR

- A description of what data is accessible through a particular kind of object, and how that data may be accessed

OR

Class define the Properties and Behavior of objects. Therefore a class is a collection of objects

OR

A class is also defined as a new data type, a user-defined type which contains two things: Data members and Methods

**Class is not REAL. Class cannot be executed. Class is a Skeleton of a Program**

A class can be executed by allocating some area called as HEAP.

**Ex:** Application Form is a class

# Class

CLASS NAME: E2\_ECE

Properties

| Object No | Name    | Age | Weight | Internal Marks | External Marks | Sex | Total Marks | Roll No |
|-----------|---------|-----|--------|----------------|----------------|-----|-------------|---------|
| Ob1       | Sai     | 22  | 59.5   | 23             | 47             | M   | 80          | ECE1    |
| Ob2       | Sandya  | 21  | 65.55  | 24             | 48             | F   | 89          | ECE2    |
| Ob3       | Gopi    | 22  | 57.65  | 22             | 56             | M   | 78          | ECE3    |
| Ob4       | Ramu    | 23  | 65.54  | 21             | 60             | M   | 77          | ECE4    |
| Ob5       | Vennela | 23  | 46.76  | 26             | 57             | F   | 98          | ECE5    |

Objects

Values

## Sample Class

A class is a description of a group of objects with common properties (attributes), behaviour (operations), relationships

### Properties

Name  
Location  
Days offered  
Credit hours  
Start time  
End time

### Class Course



### Behavior

Add a student  
Delete a student  
Get course roster  
Determine if it is full

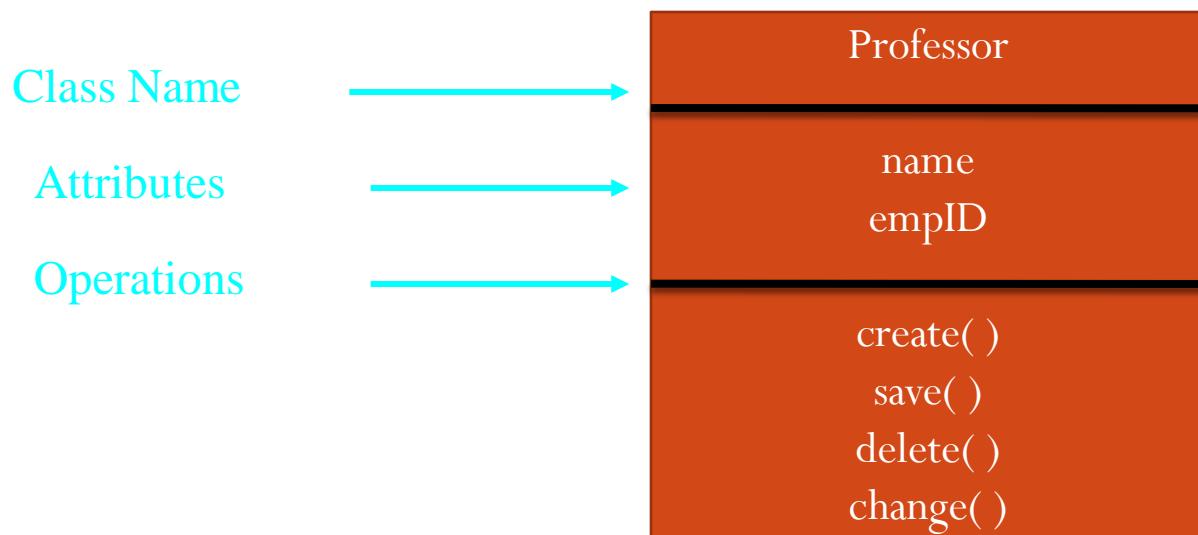
# Class Compartments

A class is comprised of three sections

The first section contains the class name

The second section shows the structure (attributes)

The third section shows the behaviour (operations)



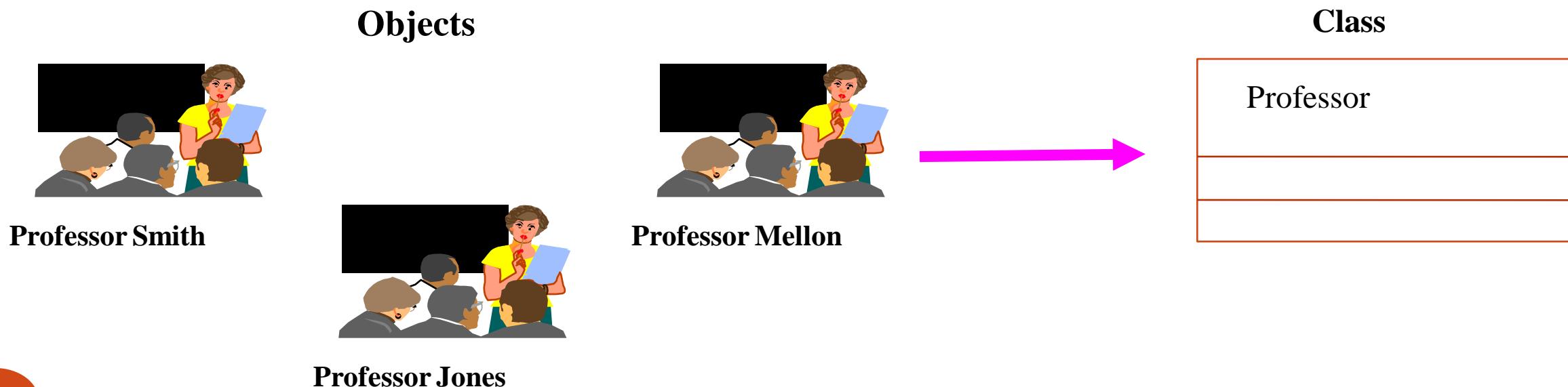
# The Relationship Between Classes and Objects

A class is an abstract definition of an object

It defines the structure and behaviour of each object in the class

It serves as a template for creating objects

Objects are grouped into classes



# What is an Attribute?

*Class*

*Attribute*

**CourseOffering**

number  
startTime  
endTime

*Attribute Value*

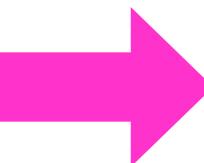
*Object*

**:CourseOffering**

number = 101  
startTime = 900  
endTime = 1100

**:CourseOffering**

number = 104  
startTime = 1300  
endTime = 1500



# Classes & Objects (1 of 2)

Let us look at some customers



## Classes & Objects (2 of 2)

- A class is a prototype / design that describes the common attributes (properties) and activities (behaviors) of objects

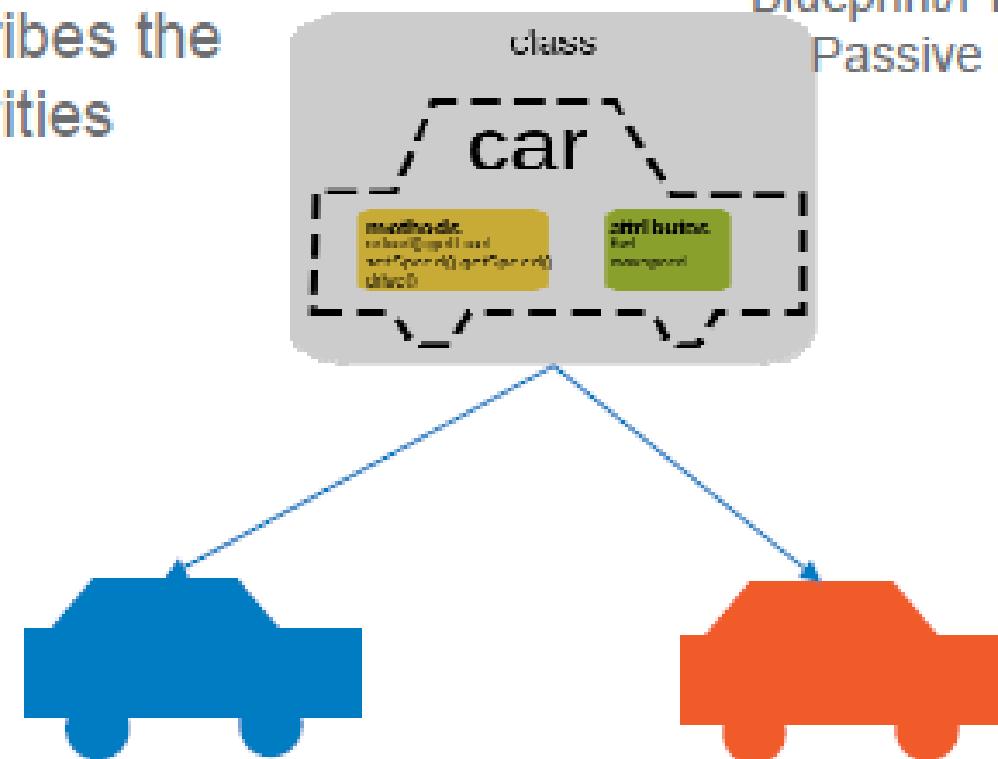
Attributes

- Example : Customer Id, Name, Telephone number and Address

Behavior/  
Activity

- Activities(behavior) exhibited by the class to external world
- Example: Purchasing items from the retail shop

Blueprint/Prototype  
Passive Entity



## ► Class

- **A class is a user defined blueprint or prototype from which objects are created.** It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:
  - 1. Modifiers :** A class can be public or has default access .
  - 2. Class name:** The name should begin with a initial letter (capitalized by convention).
  - 3. Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
  - 4. Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
  - 5. Body:** The class body surrounded by braces, { }.

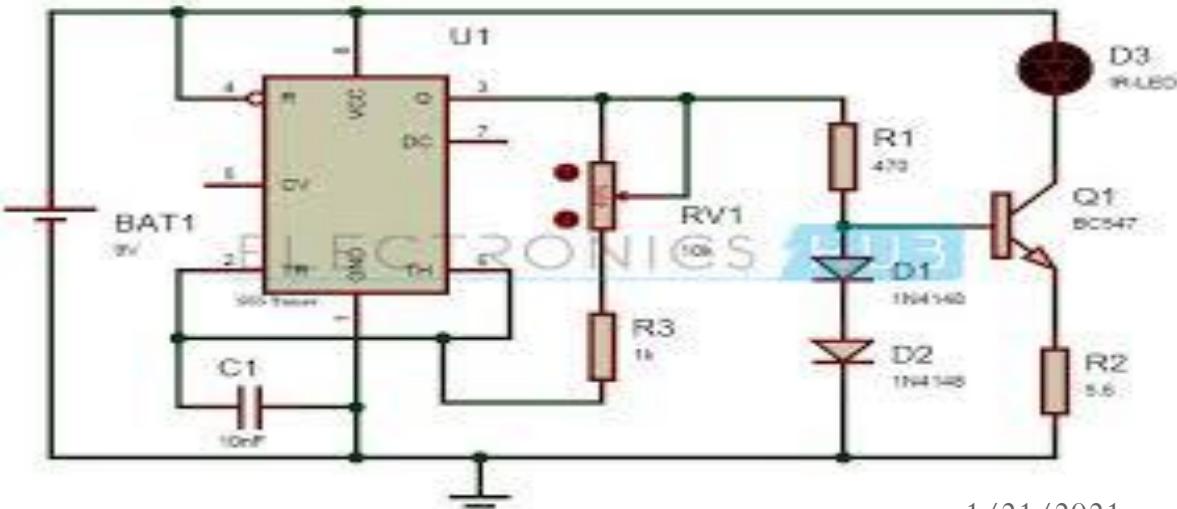
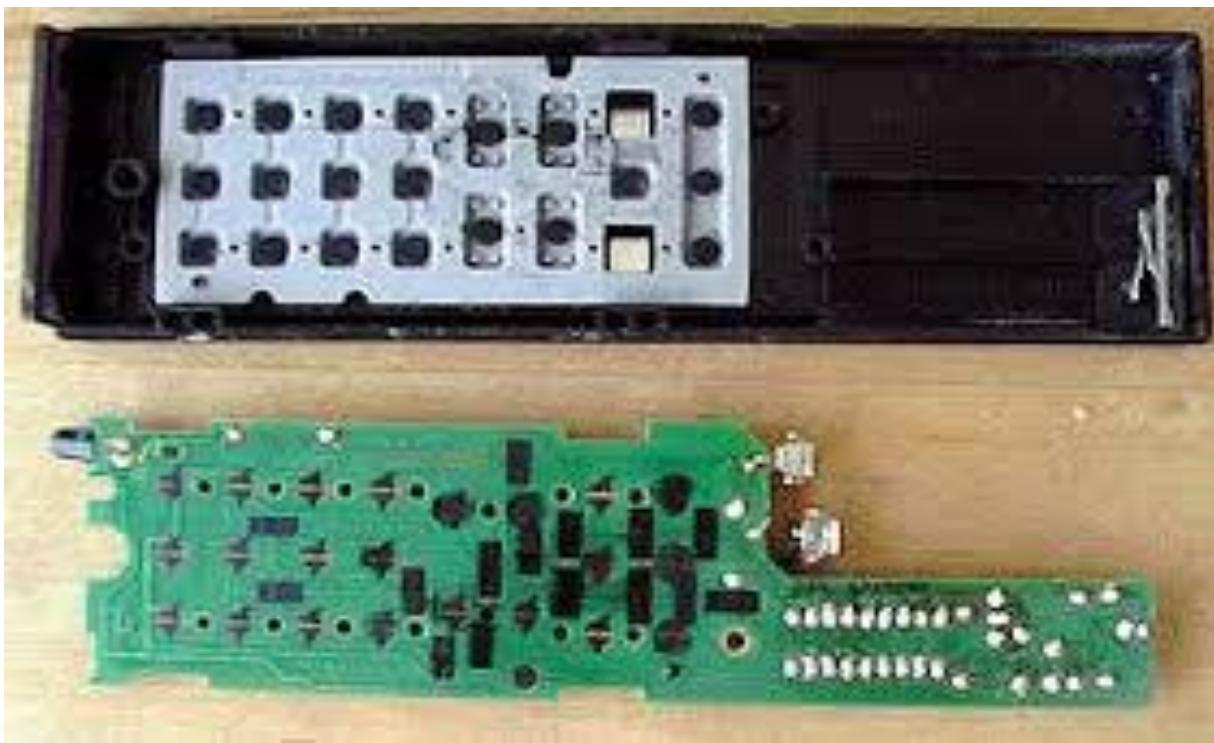
# Data Abstraction

- Abstraction is the mechanism in which Hiding internal details and showing functionality is known as abstraction
- Data Abstraction refers to the process by which data and functions are defined in such a way that only essential details are revealed and the implementation details are hidden
- The main focus of the data abstraction is to separate the interface and the implementation of a program
- **Ex:** A car is viewed as a car rather than its individual components.
- Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.
- In java, **abstraction is achieved by interfaces and abstract classes.** We can achieve 100% abstraction using interfaces.

- **Example:**

- As User of Television set, we can switch it on or off, change the channel, set the volume and add external devices such as speakers and CD or DVD players without knowing the details about how its functionality has been implemented.
- Therefore the internal implementation is completely hidden from the external world
- Similarly, in OOPS, classes provide PUBLIC methods to the outside world to provide the functionality of the object or to manipulate the object data and the implementation is hidden from the outside world
- It shows only necessary data and it hides unnecessary data that means it hides implementation
- Can you classify the following items?
  - **Elephant CDPlayer      Television      Chair   Table   Tiger**

# Abstraction

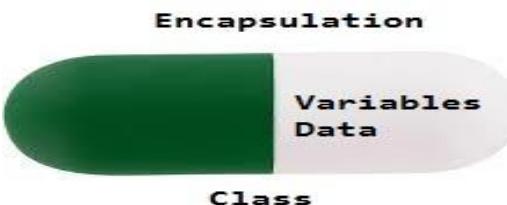


# Encapsulation



# Encapsulation

- Data encapsulation is also called as data hiding, is the technique of packing data and its functions into a single component (class) to hide implementation details of a class from the end user point of view
- Binding the (or wrapping) code and data together into a single unit is known as **encapsulation**
- Users are allowed to execute only a restricted set of operations ( class methods) on the data members of that class.
- Therefore encapsulation organizes the data and methods into a structure that prevents data access by any function(or method) that is not specific to class
- **For example:** capsule, it is wrapped with different medicines



- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.
- Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.
- **Encapsulation defines three access levels for data variables and member functions of the class:**
  - Any data or function with access level **public** can be accessed by any function belonging to any class. This is the lowest level of data protection
  - Any data or function with access level **protected** can be accessed only by that class or by any class that is inherited from it
  - Any data or function with access level **private** can be accessed only by the class in which it is declared. This is the highest level of data protection

# Polymorphism

- **Polymorphism** is one of the essential concepts of OOP, refers to having **several different forms**
- Polymorphism is a concept that enables the programmers to assign a different meaning or usage to a method in different contexts.
- Polymorphism exists when a number of subclasses is defined which have methods of same name
- **For example:** to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- **For example:** there are certain bacteria that exhibits in more than one morphological form
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

- There are **two types** of polymorphism in Java: **compile-time** polymorphism and **runtime** polymorphism
- We can perform polymorphism in java by **method overloading** and **method overriding**
- **Runtime Polymorphism**, also known as Dynamic Binding or Late Binding, is used to determine which method to invoke at runtime
- **Compile Time Polymorphism**, the compiler determines which method will be executed, hence called Early Binding
- CTP in java is implemented by **Overloading** and RTP by **Overriding**
- In **Overloading**, a method has the same name with different signatures [ Signature is the list arguments]
- In **Overriding**, a method is defined in subclass with the same name and signature as that of parent class

## Polymorphism:

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

## Polymorphism – Guided Activity

- Payment of bill - Two modes
  - Cash (Calculation includes VAT)
  - Credit card(Calculation includes processing charge and VAT)



What do you observe in this retail store scenario?

Total Amount = Purchase amount + VAT

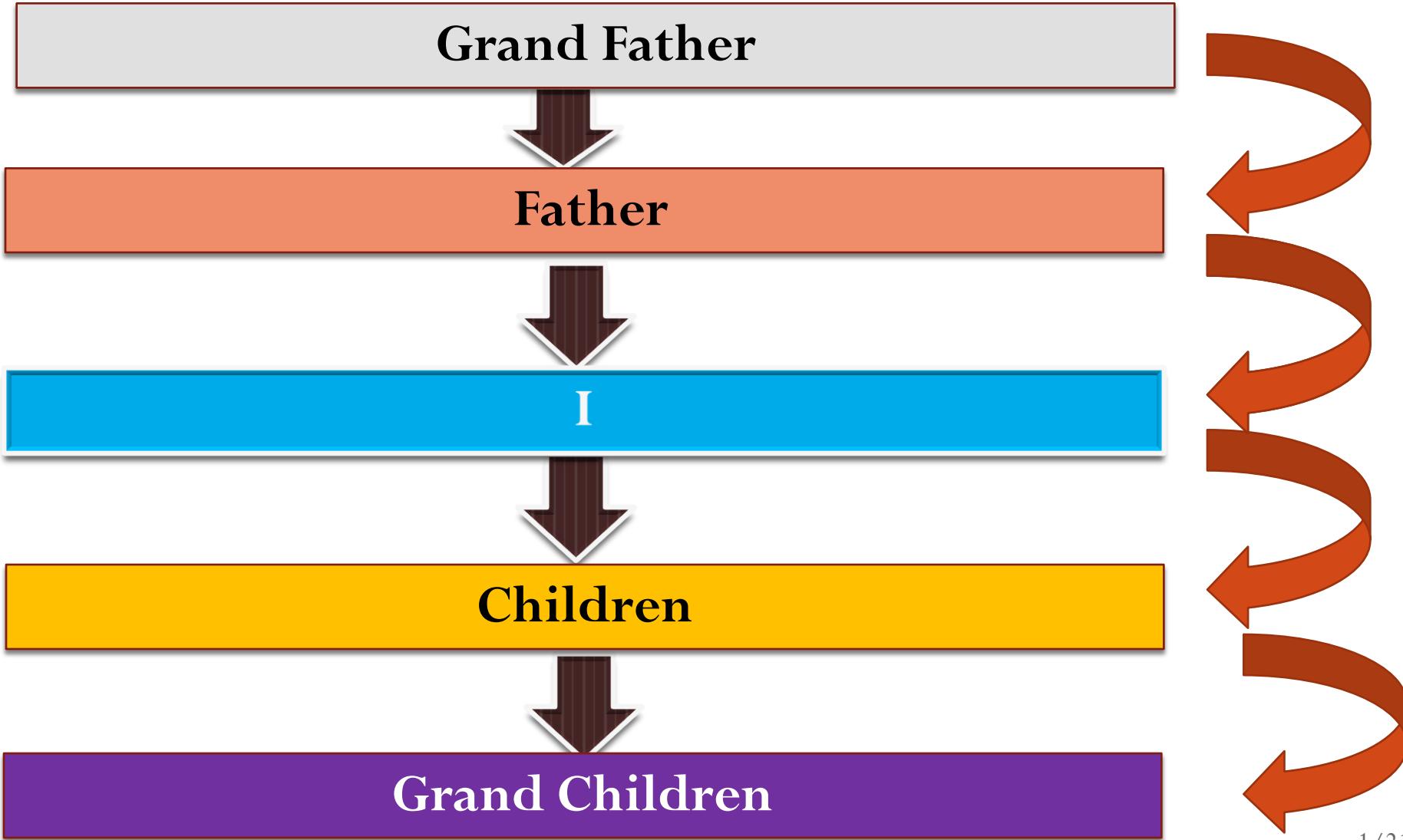
Total Amount = Purchase amount + VAT  
+ Processing charge

**POLYMORPHISM:** Refers to the ability of an object/operation to behave differently in different situations

# Inheritance

- Inheritance is a concept of OOP in which a new class is created from an existing class.
- The new class known as **Sub class** or **Derived class** or **Child class**, contains the attributes and methods of the **Super class** or **Base class** or **Parent class** [ the existing class from which the new class is created]
- The relationship followed from subclass to super class is refers as ‘**is-a**’ relation and we make use of **extends** keyword
- The subclass not only has all the states and behaviors associated with the super class but has other specialized features( additional data or methods)as well
- The main advantage of inheritance is the ability of **reuse** the code which ensures **Code Reusability**
- It is used to achieve Method Overriding [**runtime polymorphism**]

# Inheritance

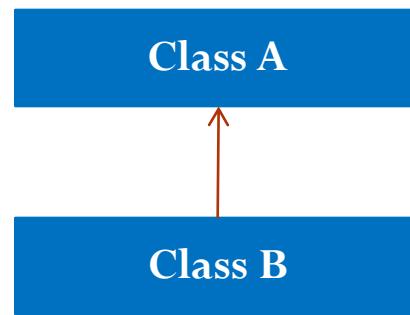


# Inheritance

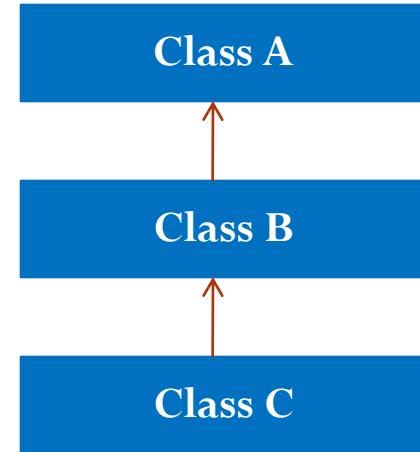
- When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
- It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.
- **Important terminology:**
- **Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as 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.
- The keyword used for inheritance is **extends**.
- **Syntax:**

```
class derived-class extends base-class
{
    //methods and fields
}
```

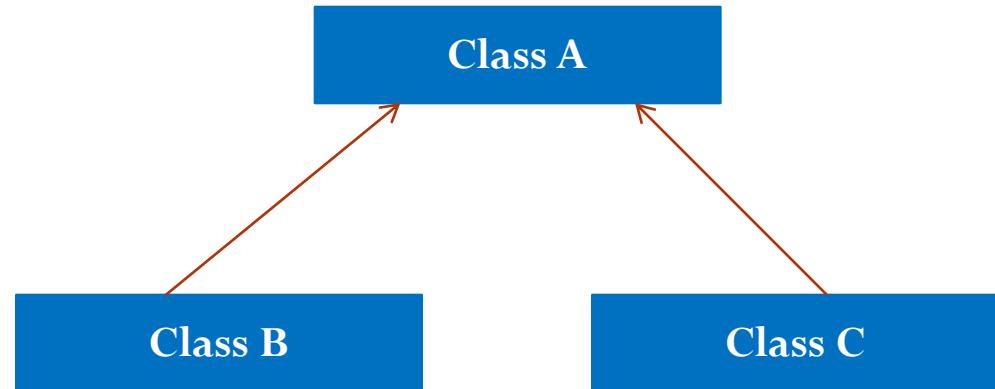
# Types of Inheritance



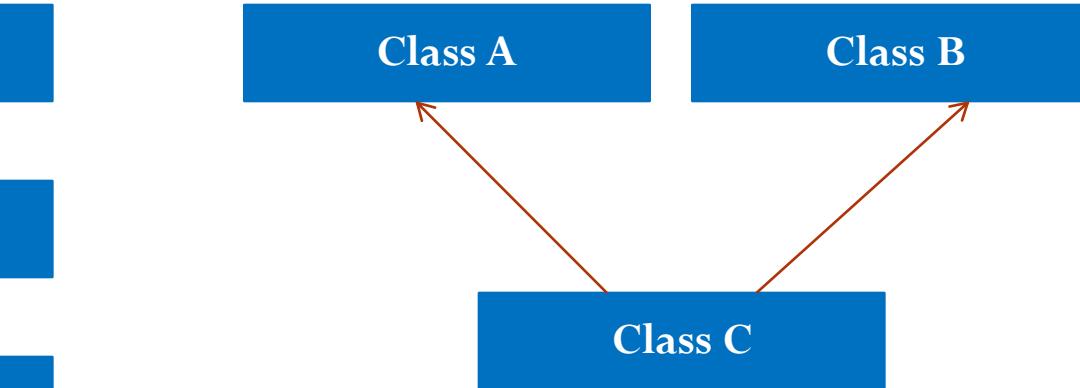
Single



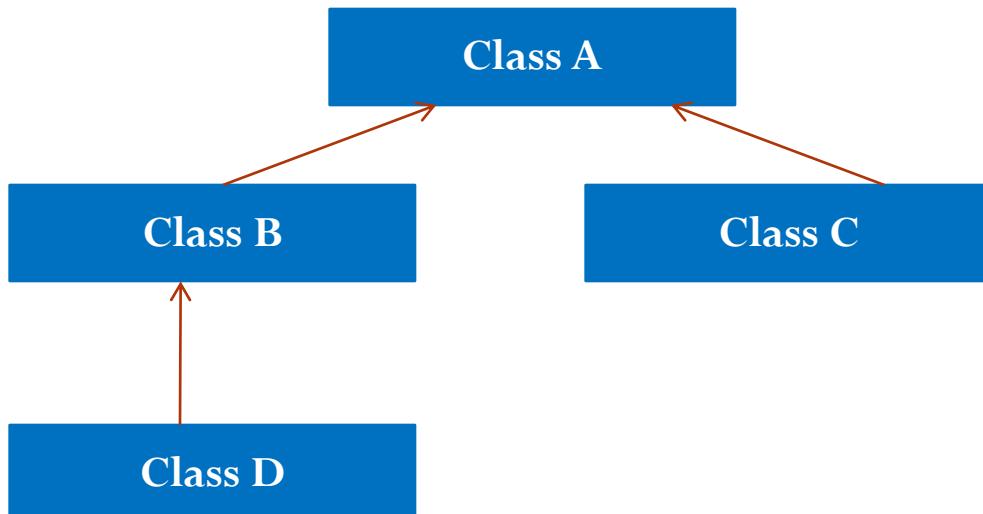
Multilevel



Hierarchical



Multiple

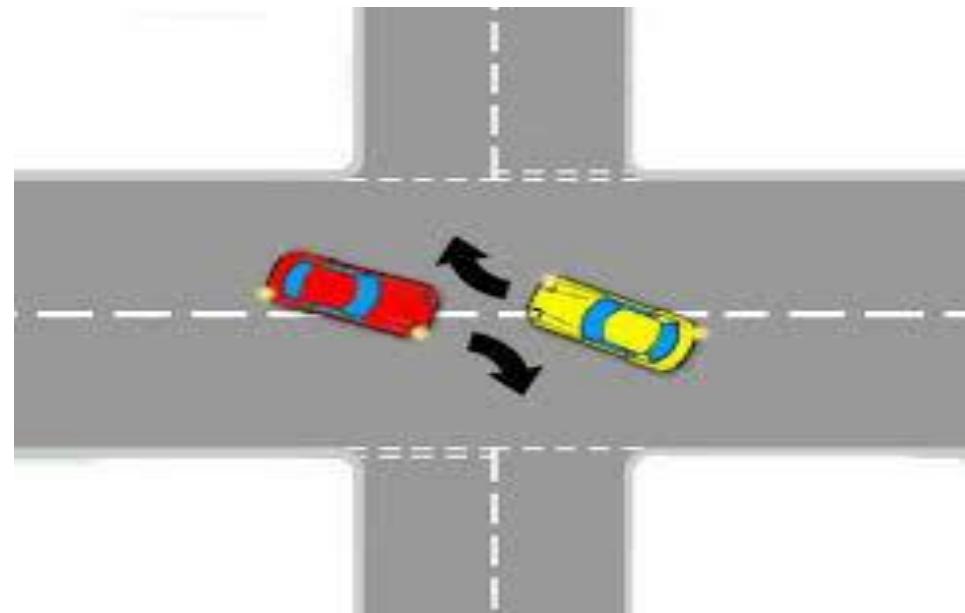


Hybrid

## ► Benefits of Inheritance

- One of the key benefits of inheritance is to **minimize the amount of duplicate code** in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilation units.
- Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass.
- **Reusability** -- facility to use public methods of base class without rewriting the same
- **Extensibility** -- extending the base class logic as per business logic of the derived class
- **Data hiding** -- base class can decide to keep some data private so that it cannot be altered by the derived class.
- **Overriding**--With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

- **Message Passing-** Two objects can communicate with each other through message passing mechanisms. An object asks another object to invoke one of its method by sending it a message



- **Method** – The means by which an object's data is accessed, modified or processed or A method is a function associated with a class. It defines the operation that the object can execute when it receives a message

## Benefits or Merits of OOP Language

- It Leads to development of Smaller but Stable subsystems
- The subsystems are flexible to change
- Reduces the risk factor in building large systems as they are build incrementally from subsystems which are stable
- Through inheritance, we can **eliminate redundant code** and extend the use of existing classes.
- The principle of data hiding helps the programmer to build **secure** programs.
- It is easy to **partition** the work in a project based on objects.
- Object oriented system easily **upgraded** from small to large systems.
- Software complexity can be easily managed
- OOPS support code reusability to a great extent
- OOPS are Realistic, where we can think of Real Time Objects

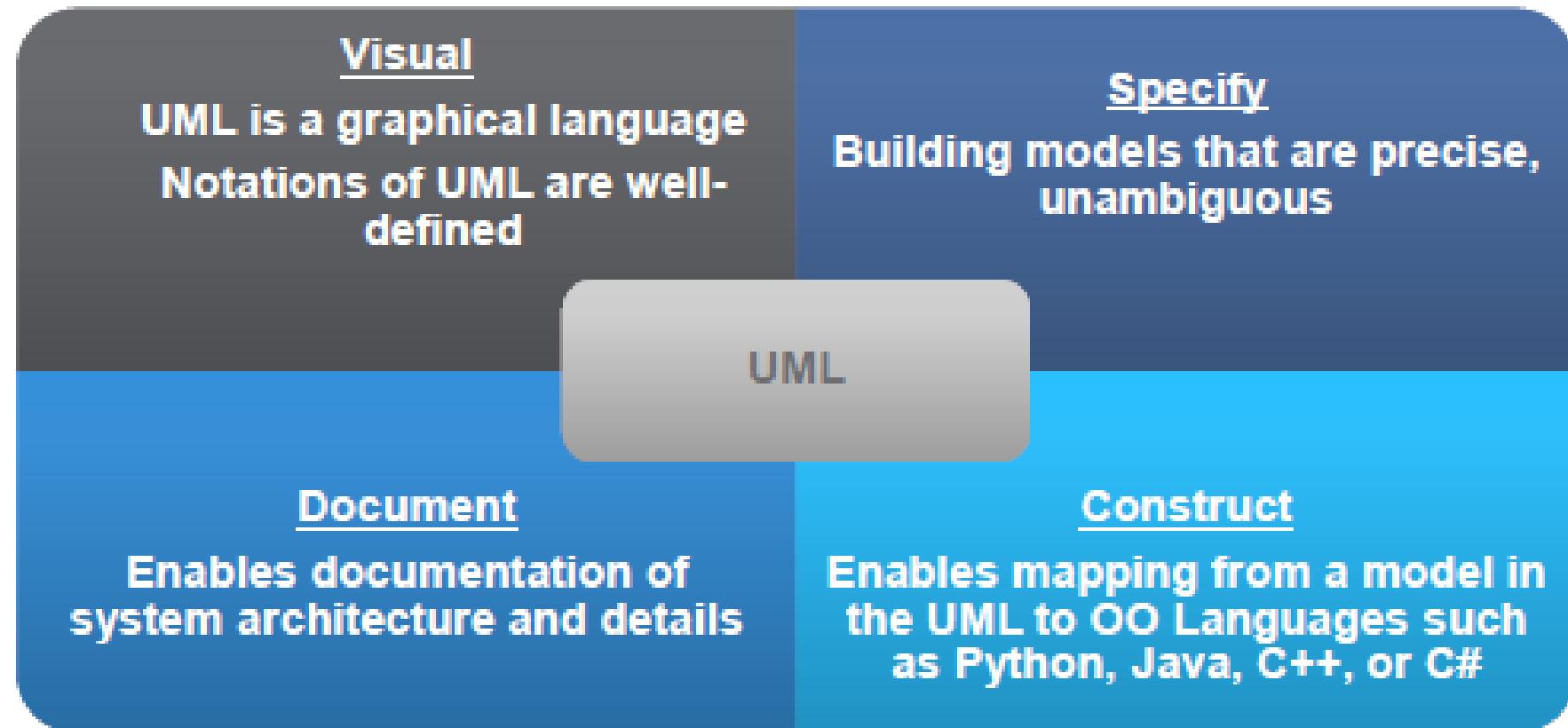
**Hence Object Orientation is suitable for development extremely complex business systems and enterprise applications**

# Applications of OOPS

- Object Oriented Technology has changed the way of thinking , analyzing, planning and implementing the software.
- Applications developed using OOT are not only efficient but also easy to upgrade.
- **Some of these areas includes the following:**
- Designing User interfaces such as work screens , menus, windows so on
- Real-time systems
- Simulation and modelling
- Compiler design
- Client Server system
- Object-oriented databases & Object Oriented distributed database management
- Artificial intelligence
- Neural networks and parallel programming
- Decision control systems and Office automation systems
- Computer-aided design (CAD)
- Computer-aided manufacturing (CAM) systems
- Computer Animations
- Developing Computer Games
- Hypertext and Hypermedia
- Network for programming routers, firewalls and other devices
- All Business Oriented Applications so on...

# What is UML?

“The Unified Modelling Language (UML) is a language for visualizing, specifying, constructing and documenting the software system and its components”  
[OMG03a]



# **Building Blocks of the UML**

**The vocabulary of the UML encompasses **3** kinds of building blocks**

## **1. Things**

Things are the abstractions that are first-class citizens in a model

## **2. Relationships**

Relationships tie these things together

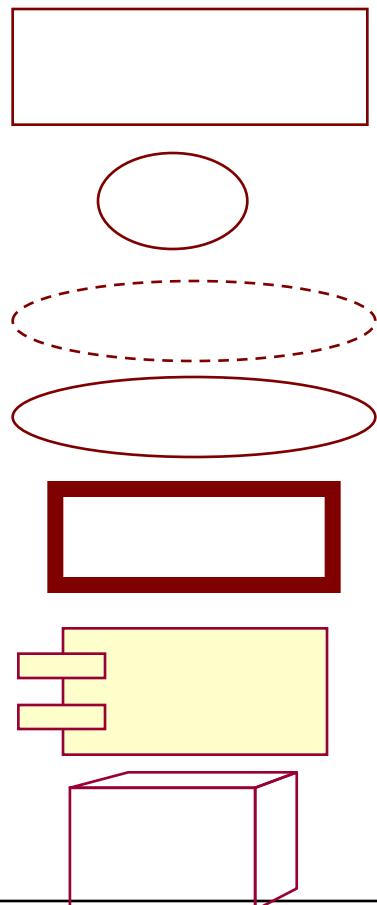
## **3. Diagrams**

Diagrams group interesting collections of things

# Structural Things

- These are the **nouns** of UML models
- mostly the **static** parts of a model
- represents **elements** that are either **conceptual** or **physical**
- There are **7** kinds of Structural Things

- |                         |                        |
|-------------------------|------------------------|
| <b>1. Class</b>         | rectangle              |
| <b>2. Interface</b>     | small circle           |
| <b>3. Collaboration</b> | dotted ellipse         |
| <b>4. Use Case</b>      | ellipse                |
| <b>5. Active Class</b>  | dark-bounded rectangle |
| <b>6. Component</b>     | tabbed rectangle       |
| <b>7. Node</b>          | cube                   |



# Behavioral Things

- These are the **verbs** of UML models
- mostly the **dynamic** parts of a model
- represents **behaviour** over **time & space**
- There are **2** kinds of Behavioral Things

## 1. Interaction

arrow



## 2. State/State Machine

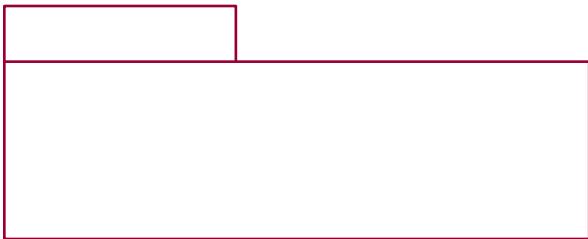
rounded rectangle



# Grouping Things

- These are the **organizational parts** of UML models
- the **boxes** into which a model can be decomposed
- There is **1** primary kind of Grouping Thing

**1. Package**      tabbed folder



# Annotational Things

- These are the **explanatory parts** of UML models, like comments & remarks
- There is **1** primary kind of Annotational Thing

**1. Note**      rectangle with a dog-eared corner



# Relationships

- A **relationship** is a connection among things
- In object oriented modeling (UML), the **4** most important relationships are:
  1. **Dependency**
  2. **Association**
  3. **Generalization**
  4. **Realization**
- Graphically, a relationship is rendered as a path, with **different kinds of lines** used to distinguish the kinds of relationships.

# UML Diagrams

The UML defines these **9** kinds of diagrams

- 1. Class Diagram**
- 2. Object Diagram**
- 3. Use case Diagram**
- 4. Sequence Diagram**
- 5. Collaboration Diagram**
- 6. State Chart Diagram**
- 7. Activity Diagram**
- 8. Component Diagram**
- 9. Deployment Diagram**

# UML Diagrams

Links: <http://www.uml-diagrams.org/>

- There are thirteen standard diagrams used in different phases

Use Case Diagram

Class Diagram

Object Diagram

Component Diagram

Composite Structure Diagram

Package Diagram

Deployment Diagram

State Machine Diagram

Activity Diagram

Sequence Diagram

Communication Diagram

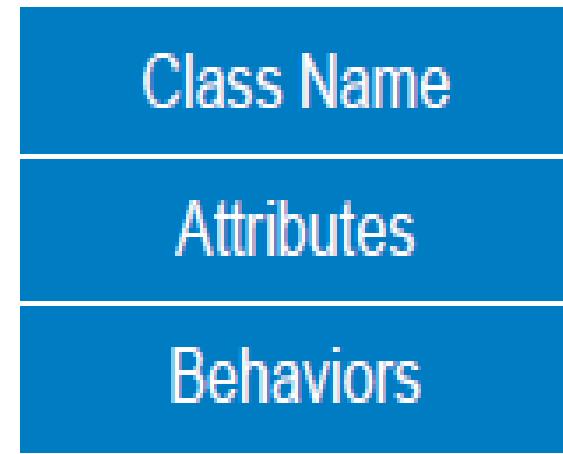
Timing Diagram

Interaction Overview Diagram

We will be focusing on class diagrams in this course

# Class Diagram

- Classes are the basic components of an object oriented system
- This diagram shows the collection of classes and the relationships among them
- In UML, any class is represented by a rectangular box divided with three compartments :



**Access Specifiers**  
public  
private  
protected

# Summary

- Object-oriented languages have become an ubiquitous standard for programming. They have been derived from the real world
- OOP revolves around objects and classes. A class is defined as a group of objects with similar attributes and behavior
- OOP is a programming paradigm which deals with the concepts of objects to develop software applications
- Certain principles have been laid down by OOP which are followed by every OOP language. These principles are: inheritance, abstraction, encapsulation, and polymorphism
- We have presented a detailed comparison of procedural and object-oriented languages. For building large projects, a technique known as OOAD is used.
- Object-oriented analysis and design deals with what the system should do and OOD deals with how the system achieves what has been specified by OOA
- OOAD is realized with the help of a language known as UML
- UML stands for Unified Modeling Language; it is a standard language used for visualizing the software
- An abstract model is created for the entire software using graphical notations provided by UML

A photograph of a person's hand holding a white rectangular card. The card contains the following text:

**For Details Contact Me @ :**  
**9247448766**  
**ravikanth.iiit@rgukt.ac.in**

The background of the slide shows a large, semi-transparent image of a hand holding a similar white card.

## UNIT-I

**Topic: Java : History, Java features, Java Environment, JDK, API.**

### History of Java



Creator of Java  
James Gosling

Java First named was "Oak" on Oak Tree  
Final name "Java" on Java Coffee



Oak Tree

### Java's Journey : From Embedded Systems to Middle-Tier Applications

- **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**
- Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called "**Greentalk**" by **James Gosling**, and file extension was .gt
- After that, it was called **Oak** and was developed as a part of the **Green project**.
- **Why Java named "Oak"?**
- **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.



JAVA LANGUAGE  
PROJECT

JAMES GOSLING



PATRICK NAUGHTON

MIKE SHERIDAN



1991 JUNE

- It was developed as an embedded programming language, which would enable embedded system application. It was not really created as web programming language. Java is available as *jdk* and it is an open source s/w.
- Primary motivation of Java was the need of Platform -independent which could be used to create s/w to be embedded in various consumer electronic devices like TVs, VCRs, Ovens, Remote Controls.
- Later **Oak** was renamed Java in 1995. Java has released its first version called JDK 1.0 on 23<sup>rd</sup> Jan, 1996.. Now Sun Microsystems of USA is a subsidiary of Oracle Corporation
- Introduction to OOPs using java: JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation. It was developed by James Gosling and Patrick Naughton. It is a simple programming language. Writing, compiling and debugging a program is easy in java. It helps to create modular programs and reusable code.
- The team wanted something that reflected the essence of the technology: revolutionary,dynamic,lively, cool, unique, and easy to spell and fun to say
- In 1995, Oak was renamed as Java- where java is an island of Indonesia where first coffee was produced ( called java coffee)
- JDK 1.0 got released

**Green Team Members:** James Gosling, Patrick Naughton, Chris Warth, Ed Frank, Mike Sheridan, Bill Joy, Arthur van Hoff, Janathan Payne, Frank Yellin, Tim Lindholm were key contributors in designing And developing JAVA Programming Language.



### Mile Stones of Java

**1990** – Sun Microsystems, headed by James Gosling decided to develop a Special. S/W that could be used to manipulate Consumer Electronic Devices.

**1991** – The team announced a new lang. Named “**OAK**”

**1992** – The Team demonstrated he appl. Of their new language to control a list of home appliances

**1993** – WWW appeared on the Internet & Transformed the text-based Internet into graphical-rich environment. The team came up with the idea of developing web applets ( tiny programs) that could run on all types of computers connected to Internet

**1994**- The team developed a web browser called “ **Hot Java**” to locate & run applet programs on Internet

**1995**- Oak was renamed “**Java**”. Many popular companies including Netscape, Microsoft announced their support

**1996**- Java established itself not only as a leader for Internet Programming but also as a general purpose OOP Lang.

### Versions of Java

Next Versions are:

**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)**

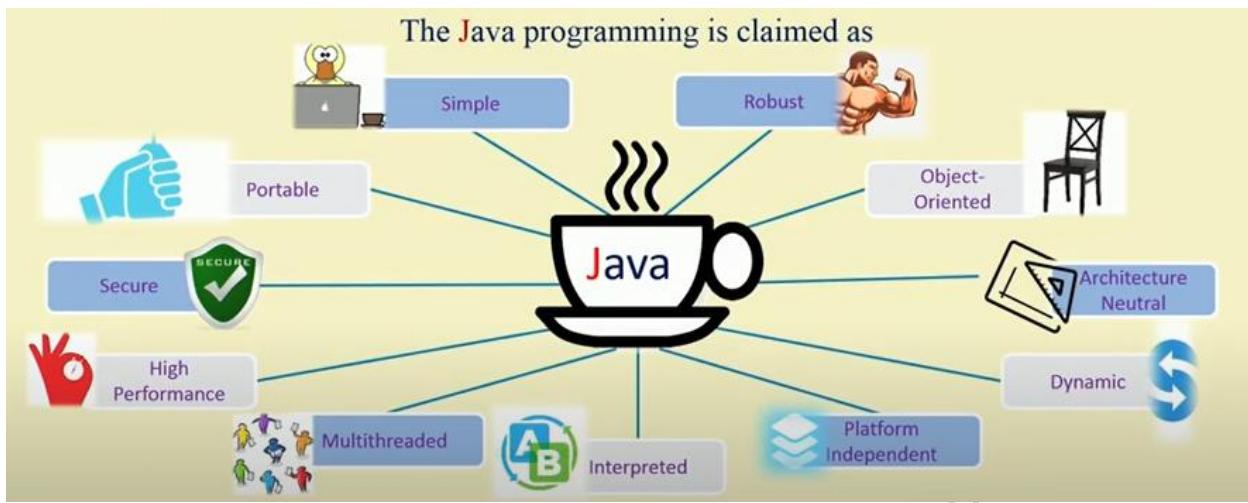
**Java SE 8 (March 18, 2014)**

**Java SE 9 (September 21, 2017)**

**Java SE 10 (March 20, 2018)**

**Java SE 11 (September 25, 2018)**

**Java SE 12 (March 19, 2019)**

**Features of JAVA [ Java Buzz Words]****Simple, Small & Familiar**

- Java is very easy to learn, and its syntax is simple, small and easy to understand. According to Sun, Java language is a simple programming language because:
- Java Syntax is based on C++
- Java does not have complex features like explicit pointers, Operator overloading, Multiple inheritance and Explicit memory allocation.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java

**Object Oriented**

- Java is an Object-Oriented programming language. Everything in Java is an object. Object-Oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPS) is a methodology that simplifies software development and maintenance by providing some rules.

- **Basic Concepts of OOPs are:** Object, Class, Inheritance, Polymorphism, Abstraction and Encapsulation

## Platform Independent language

- Java is platform independent because it is different from other languages like C,C++ etc. which are complied into platform specific machines while Java is a Write Once, Run Anywhere language. A platform is the hardware or software environment in which a program runs.
- There are two types of platforms software-based and hardware-based
- Java Provides a Software-based platform, which has two components:

### 1. Runtime Environment

### 2. API(Application Programming Interface)

- Compiler (javac) converts source code (.java file) to the byte code(.class file). As mentioned above, JVM executes the bytecode produced by compiler.
- This byte code is a platform-independent code which can run on any platform such as Windows, Linux, Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa.
- Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call java as platform independent language.

## Secure

- Java is best known for its security. With java we can develop virus-free systems. Java is secure because:
  - No explicit Pointers: We don't have pointers and we cannot access out of bound arrays (you get ArrayIndexOutOfBoundsException if you try to do so) in java. That's why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

- Java Programs run inside a virtual machine sandbox
- Classloader: Classloader in java is apart of the JRE which is used to load java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources
- Bytecode Verifier: It checks the code fragments for illegal code that can violate access right to objects
- Security Manager: It determines what resources a class can access such as reading and writing to the local disk

## **Robust Language**

- Robust means reliable Strong.
- Java programming language is developed in a way that puts a lot of importance on early checking for possible errors, that's why java compiler is able to detect errors that are not easy to detect in other programming languages.
- The main features of java that makes it robust are garbage collection, Exception Handling and memory allocation.

## **Architecture-neutral**

- Java is architectural neutral because there are no implementation dependent features, for example, the size of primitive data types is fixed
- Ex: In C Programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java

## **High Performance**

- Java is faster than other traditional interpreted programming languages because Java byte code is “close” to native code

- It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

## Java is Distributed

- Using java programming language we can create distributed applications.
- RMI(Remote Method Invocation) and EJB(Enterprise Java Beans) are used for creating distributed applications in java. In simple words: The java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (java virtual machine) can execute procedures on a remote JVM.

## Multithreading

- Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.
- The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, web applications etc.

## Portable

- As discussed above, java code that is written on one machine can run on another machine. The platform independent byte code can be carried to any platform for execution that makes java code portable. It doesn't require any implementation.

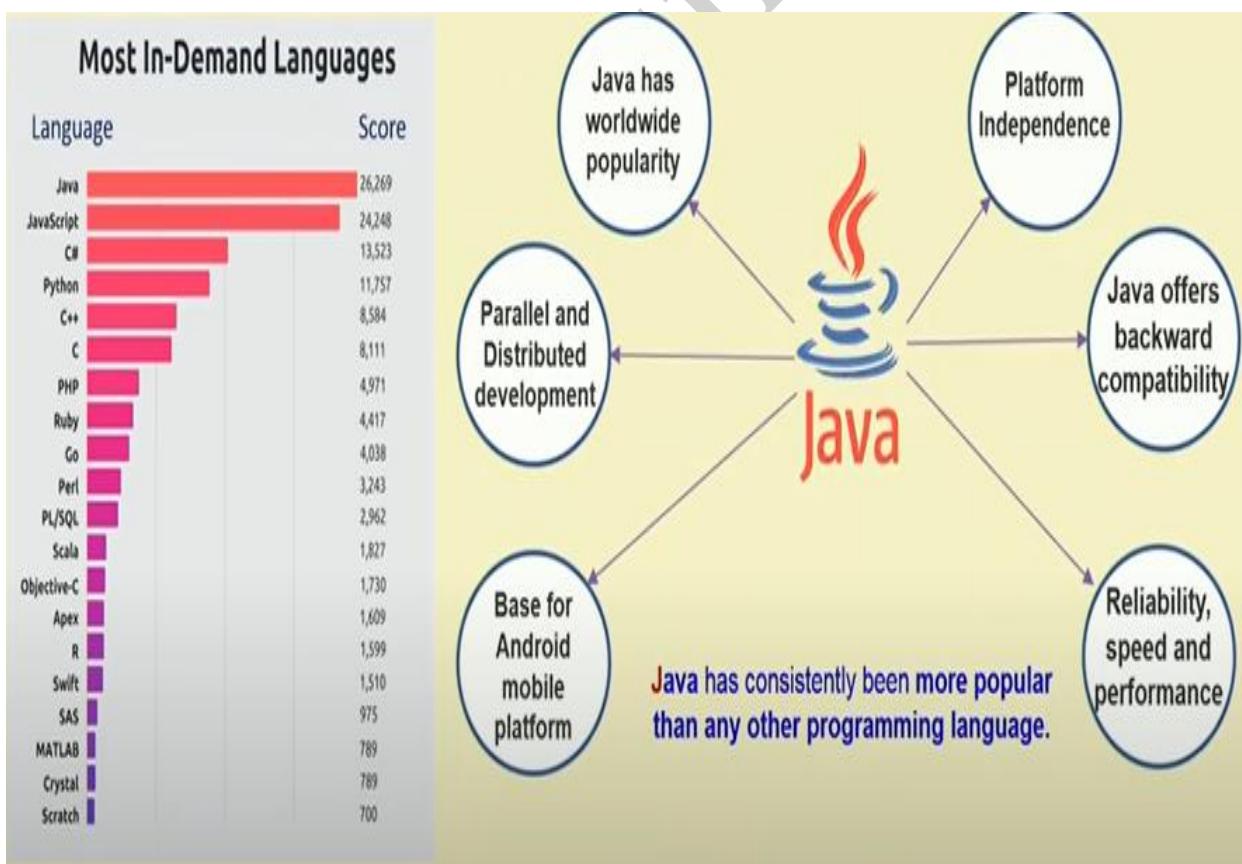
## Dynamic

- Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native language i.e., C and C++.
- Java supports dynamic compilation and automatic memory management(Garbage Collection)

### Interpreted

- Usually a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.
- **Compiled:** Java enables creation of a cross platform programs by compiling into an intermediate representation called Java Bytecode.
- **Interpreted:** Bytecode is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.

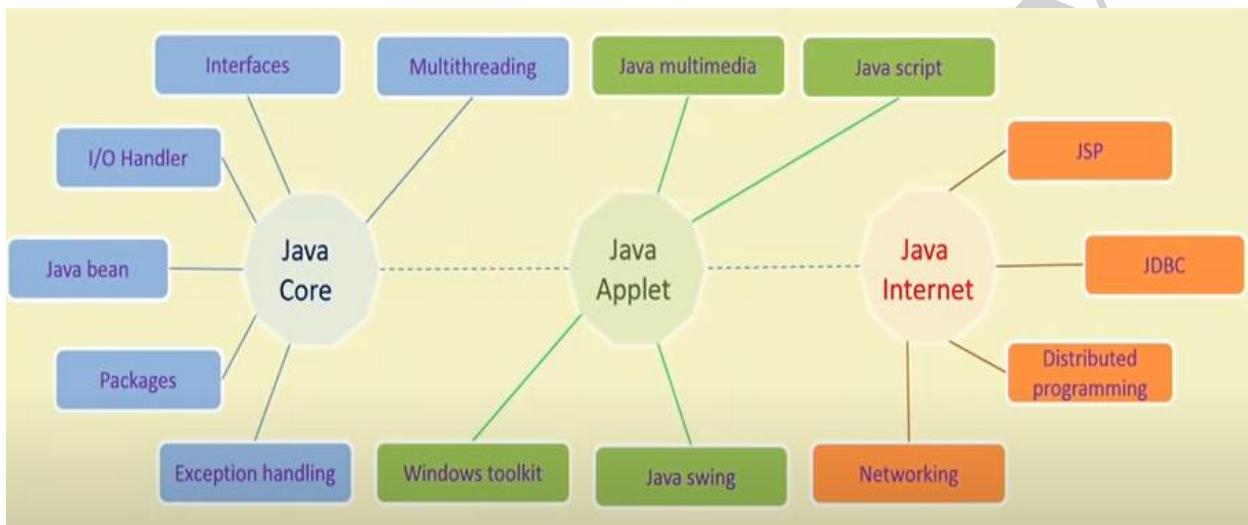
### Current Popular Programming Languages and their Demand



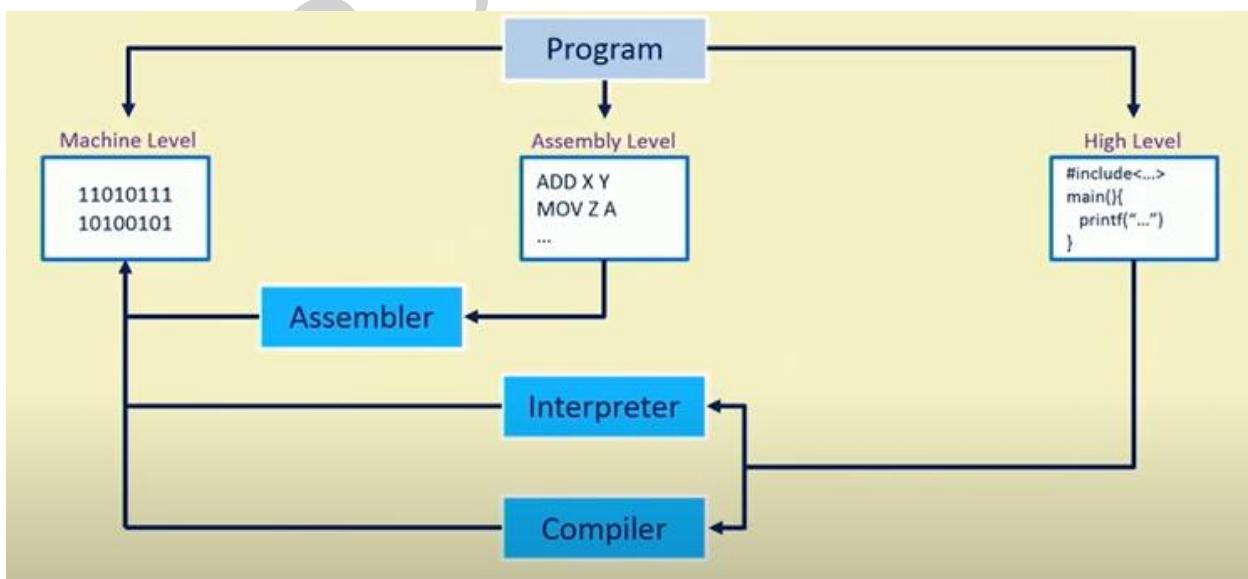
## Java Platforms

There are three main platforms for Java:

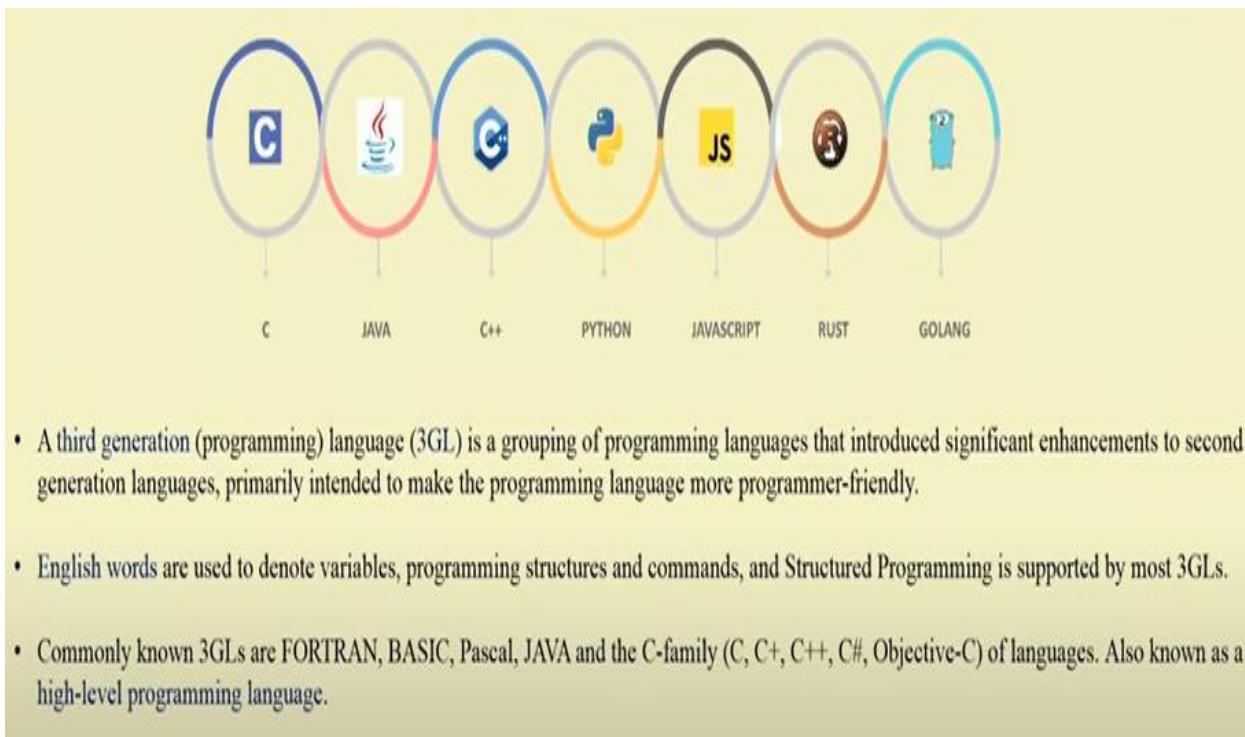
- Java SE (Java Platform, Standard Edition) – runs on desktops and laptops.
- Java EE (Java Platform, Enterprise Edition) – runs on servers.
- Java ME (Java Platform, Micro Edition) – runs on mobile devices such as cell phones.



## Programming languages



### Third Generation Programming Languages



## **Java Environment**

It includes a large number of **development tools** & hundreds of **classes & methods**.

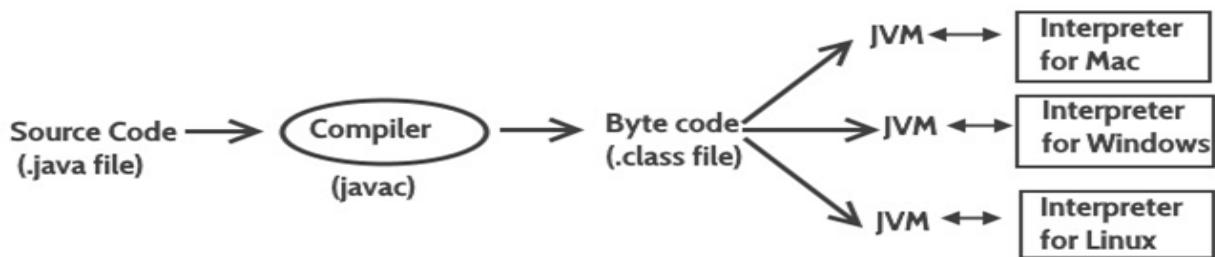
The development tools are part of the system called **Java Development Kit ( JDK)**

The Classes & Methods are part of the **Java Standard Library ( JSR)** also called as **Application Programming Interface ( API)**

**JDK** comes with a collection of Basic Tools that are used for developing & running Java Programs.

They include:

1. **javac ( Java Compiler)**-which translates Java Source code to bytecode files that the Interpreter can understand. **Java filename.java**
2. **java ( Java Interpreter)**-runs applets & applications by reading & interpreting bytecode files  
**Java filename**
3. **javap ( Java Disassembler)**-enables us to convert bytecode files into a pgm Description
4. **javah ( for C header files)** – produces header files for use with native methods
5. **javadoc** – Creates HTML format documentation from Java Source code files
6. **jdb ( Java Debugger)** – helps to find errors in program
7. **appletviewer** – enables us to run Java applets, without actually using a Java Compatable Brower
8. **jar** – used for creating and managing jar (similar to WinZip file) files
9. **Servletrunner**- A Simple WebServer to test Servlets
10. **extcheck** – It is used for detecting Jar conflicts



### **Java Terminology:-**

**1. Java Virtual Machine (JVM)** This is generally referred as JVM. Before, we discuss about JVM lets see the phases of program execution. Phases are as follows: we write the program, then we compile the program and at last we run the program.

- Writing of the program is of course done by java programmer like you and me.
- Compilation of program is done by javac compiler, javac is the primary java compiler included in java development kit (JDK). It takes java program as input and generates java bytecode as output.
- In third phase, JVM executes the bytecode generated by compiler. This is called program run phase.
- So, now that we understood that the primary function of JVM is to execute the bytecode produced by compiler.
- **Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems.**
- That is why we call java as platform independent language.
- JVM is platform dependent ( It uses the class libraries and other supporting files provided in JRE)

### **2. bytecode**

As discussed above, javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. The bytecode is saved in a .class file by compiler.

### **3. Java Development Kit(JDK)**

As the name suggests this is complete java development kit that includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc. In order to create, compile and run Java program you would need JDK installed on your computer.

## 4. Java Runtime Environment(JRE)

JRE is a part of JDK which means that JDK includes JRE. When you have JRE installed on your system, you can run a java program however you won't be able to compile it. JRE includes JVM, browser plugins and applets support. When you only need to run a java program on your computer, you would only need JRE.

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

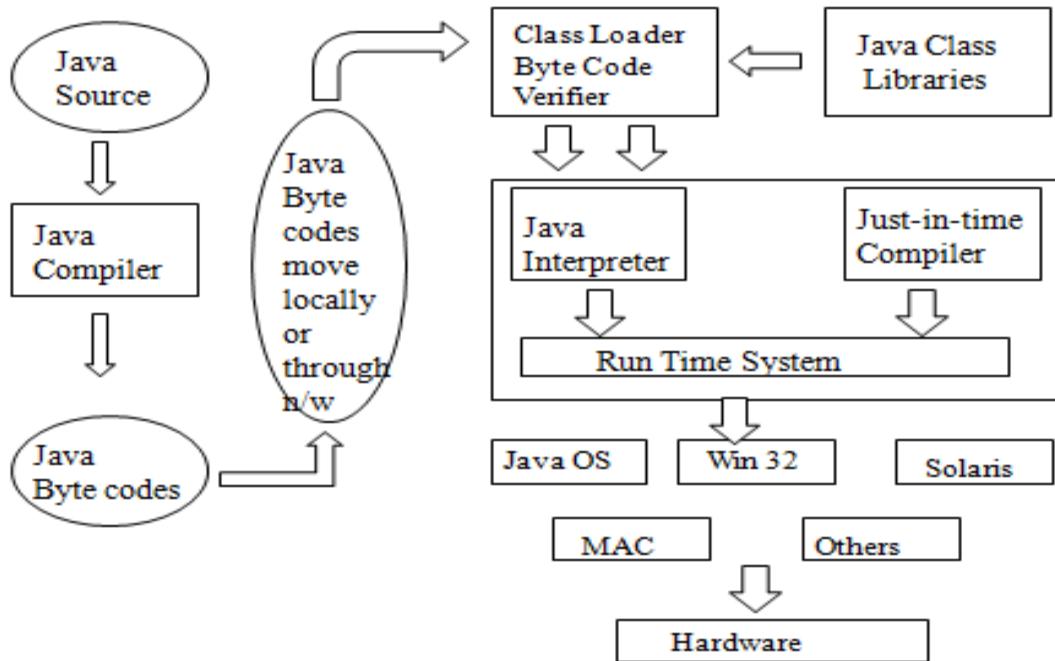
## Java Virtual Machine (JVM), Difference JDK, JRE & JVM – Core Java

Java is a high level programming language. A program written in high level language cannot be run on any machine directly. First, it needs to be translated into that particular machine language. The **javac compiler** does this thing, it takes java program (.java file containing source code) and translates it into machine code (referred as byte code or .class file).

Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the **machine language for JVM is byte code**. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output

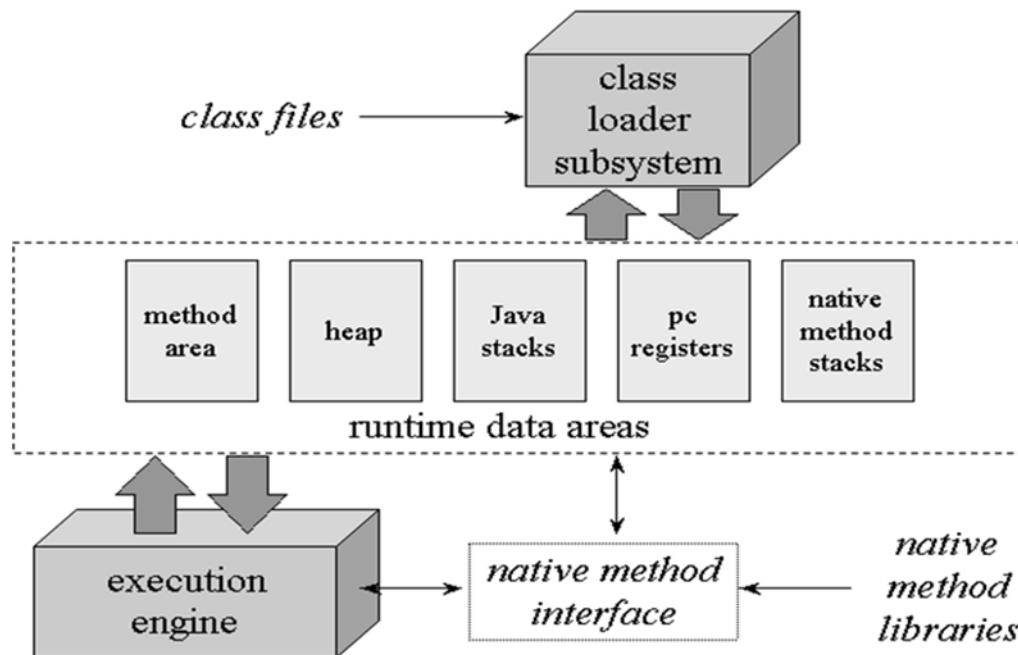
So, now we understood that the primary function of JVM is to execute the byte code produced by compiler. Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems. This means that the byte code generated on Windows can be run on Mac OS and vice versa. That is why we call java as platform independent language. The same thing can be seen in the diagram below:

## Java Execution Procedure



So to summarize everything: The Java Virtual machine (JVM) is the virtual machine that runs on actual machine (your computer) and executes Java byte code. The JVM doesn't understand Java source code, that's why we need to have javac compiler that compiles \*.java files to obtain \*.class files that contain the byte codes understood by the JVM. JVM makes java portable (write once, run anywhere). Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.

## The Architecture of the Java Virtual Machine



### How JVW works:

#### Class loader subsystem

- The Java virtual machine contains two kinds of class loaders: a *bootstrap class loader* and *user-defined class loaders*.
- The bootstrap class loader is a part of the virtual machine implementation, and user-defined class loaders are part of the running Java application.
- **Linking:** performing verification, preparation, and (optionally) resolution
  - Verification: ensuring the correctness of the imported type
  - Preparation: allocating memory for class variables and initializing the memory to default values
  - Resolution: transforming symbolic references from the type into direct references.

- **Initialization:** invoking Java code that initializes class variables to their proper starting values.

**Class Loader:** The class loader reads the .class file and save the byte code in the method area.

**Method Area:** There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.

**Heap:** Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file.

**Stack:** Stack is also a part of JVM memory but unlike Heap, it is used for storing temporary variables.

**PC Registers:** This keeps the track of which instruction has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.

**Native Method stack:** A native method can access the runtime data areas of the virtual machine.

**Native Method interface:** It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.

**Garbage collection:** A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.

### **Execution Engine**

It contains:

- 1) A virtual processor
- 2) Interpreter: Read bytecode stream then execute the instructions.
- 3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time

needed for compilation. Here the term compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

### **JDK Vs JRE Vs JVM**

**Java Development Kit (JDK):** is comprised of three basic components, as follows:

- Java compiler
- Java Virtual Machine (JVM)
- Java Application Programming Interface (API)

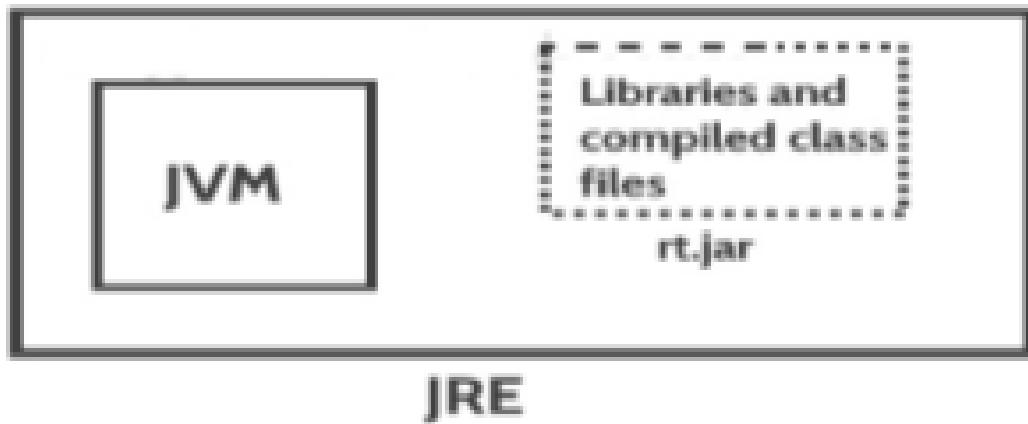
The Java API, included with the JDK, describes the function of each of its components. In Java programming, many of these components are pre-created and commonly used. Thus, the programmer is able to apply prewritten code via the Java API. After referring to the available API classes and packages, the programmer easily invokes the necessary code classes and packages for implementation.

An application programming interface (API), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction.

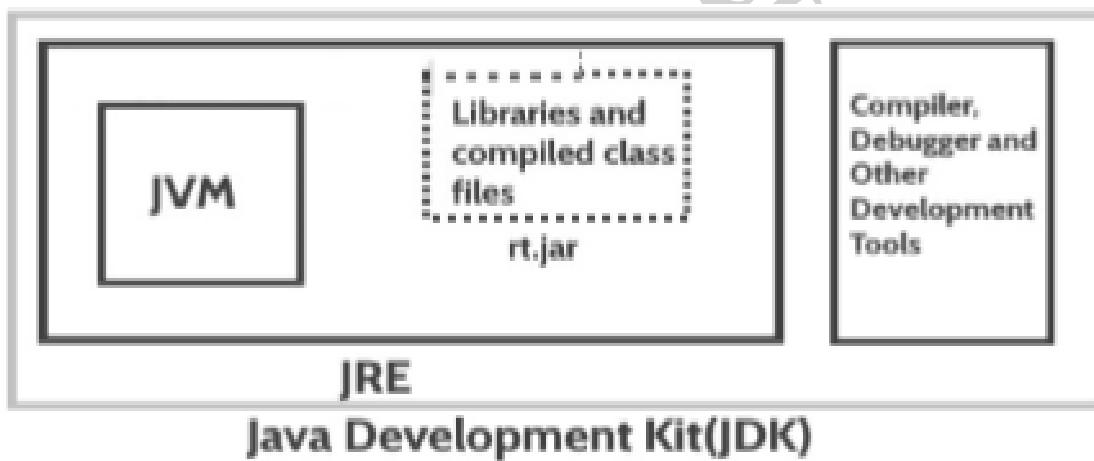
In Java, most basic programming tasks are performed by the API's classes and packages, which are helpful in minimizing the number of lines written within pieces of code.

**JRE:** JRE is the environment within which the java virtual machine runs. JRE contains Java virtual Machine(JVM), class libraries, and other files excluding development tools such as compiler and debugger. Which means you can run the code in JRE but you can't develop and compile the code in JRE.

**JVM:** As we discussed above, JVM runs the program by using class, libraries and files provided by JRE.



**JDK:** JDK is a superset of JRE, it contains everything that JRE has along with development tools such as compiler, debugger etc.



**Binary form of a .class file(partial)**

```

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

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

```

**JSL / API**

JSL/ API includes hundreds of classes & methods grouped into several functional Packages.

Most commonly used packages are:

1. **Language Support Package** – To implement basic features of java (**java.lang**)
2. **Utilities Package** – To provide utility functions like date & time (**java.util**)
3. **Input/Output Package** – For Input/Output manipulations (**java.io**)
4. **Networking Package** – For Communicating with other computer via Internet (**java.network**)
5. **AWT Package** – The Abstract window Tool Kit Package contains classes that implement platform-independent graphical user interface (**java.awt**)

6. **Applet Package** – Allows us to create Java Applets ( **java.applet**)

7. **Swing Package** – Provides classes for swing operations programming ( **java.swing**)

### **Just In Time ( JIT)**

The JVM is an interpreter that translates & runs each byte code instruction separately whenever it is needed by the computer program. In some cases, it is very slow. As an alternative, Java also provides local compiler for each system that will compile byte code file into executable code for faster running. Java calls these compilers as **JUST IN TIME COMPILERS**.

### **Integrated Development Environment ( IDE)**

IDE consists of tools specifically designed for writing Java codes. These tools offer a GUI environment to compile and debug Java program easily from the editor environment as well as browse through the classes.

**Some of IDE's are: Eclipse, NetBeans, JCreator, DrJava, Gel...so on.**

## UNIT-I

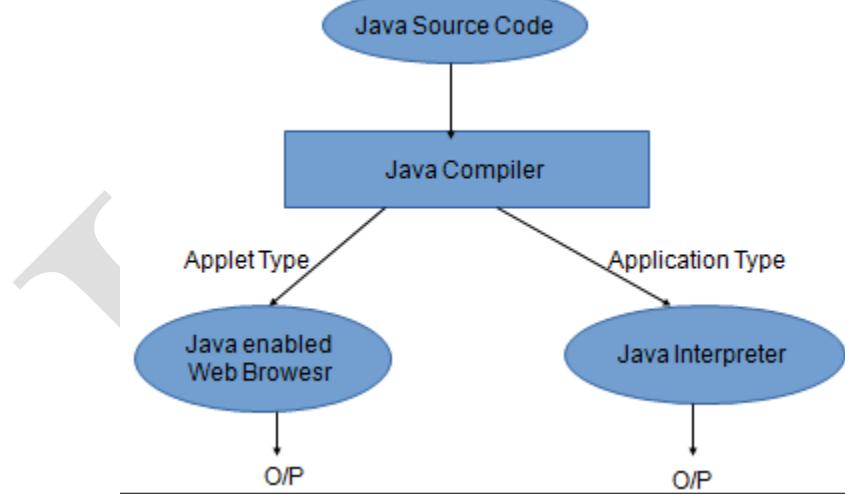
### **Topics to be covered:**

**Introduction to Java :** Types of java program, Creating and Executing a Java program, Java Tokens, Keywords, Character set, Identifiers, Literals, Separator, Command Line Arguments, Comments in Java program.

### **Types of java program:**

Java is a general-purpose, OOPL. We can develop 2 types of Java Programs.

1. Stand-alone Applications
2. Web Applications/Web Applets



### **Stand-alone Applications:**

- The Programs written in Java to carry out certain tasks on a standalone local computer
- Java can be used to development programs for all kinds of applications
- Executing a standalone Java Programs involves 2 steps:

1. Compiling source code into byte code using “javac” compiler

2. Executing the bytecode program using “java” interpreter

### **Web Applets:**

- Applets are small java programs developed for Internet applications
- An applet located on a distance computer (server) can be downloaded via Internet & Executed on a local computer (client) using a Java-capable Browser.
- We can develop applets for doing simple animated graphics to complex games
- Creating & running applets are more complex than applications
- JAR file (Java ARchive) - used to package Java files together into a single file (almost exactly like a .zip file).
- Servlet - runs on a web server and helps to generate web pages.
- Swing application - used to build an application that has a GUI (windows, buttons, menus, etc.).
- EJB - runs on a web server and is used to develop large, complex websites.

**Structure of a JAVA Program:**

- Documentation Section ----- Suggested
- Package Statements ----- Optional
- Import Statements ----- Optional
- Interface Statements ----- Optional
- Class Definitions ----- Optional
- Main Method Class ----- Essential
- {
- Main Method Def.
- }

**Documentation Section:**

- It comprises a set of comment lines, which would describe the program
- Java uses the third style of comment / \* \* .....\* / called documentation comment

**Package Section:**

- The 1<sup>st</sup> statement allowed in a java file is a '**Package**' statement
- This statement declares a package name & informs the compiler that the classes defined here belong to this package
- **Ex:** package student;

**Import Section:**

- Import Statement is similar to the #include stmt. In C. There may be any no. Of import stmts

- Ex: import student.Test;
- This stmt. Instructs the interpreter to load the “test” class contained in the package “student” i.e, using import stmt, we can have access to classes that are part of other named packages

### Interface Section:

- An interface is like a class, but includes a group of method decl.s, It is used when we wish to implement multiple inheritance feature in the pgm

### Class Definition Section:

- A java pgm may contain multiple class def.s,
- Classes are the primary & essential elements of a Java Pgm
- The no. Of classes used depends on the complexity of the problem

### Main Method Section:

- The main method class is the essential part of a Java Pgm
- Every java Stand-alone pgm requires it as its starting point
- It creates Objects of various classes & establishes connection between them
- On reaching the end of the main, the pgm terminates & the ctrl passes back to the Operating System
- A java pgm may contain many classes of which only one class defines a main method

## Creating and Executing a Java program

It involves a series of steps:

1. Creating the Program
2. Compiling the Program
3. Running the Program

Before we begin creating the pgm, the JDK must be properly installed on the system

We can create the pgm using any text editor

**Ex1: AIM: To display a message on the Screen**

### **Example 1- First Java Program**

**Line1:** /\* Call this file “Example.java”.\*/

**Line2:** public class Example

{

**Line3:** // Your program starts execution with a call to main()

**Line4:**       public static void main( String args[ ])

{

**Line5:**       System.out.println(“This is a simple Java Program”);

**Line6:**       } // End of main method

**Line7:** } // End of Example Class

**Expected Output: This is a simple Java Program**

**Explanation:**

- Java has the facility of command line arguments.
- Java main method consists of array of string objects. When we run the program, the array will fill with the values of any arguments it was given in the command line
- The keyword “**class**” is used to declare the new class being defined
- “**Example**” is an identifier i.e., name of the class
- “**public**” keyword is an access specifier, which allows the programmer to control the visibility of the class members
- The keyword “**static**” allows main( ) to be called without having to instantiate a particular instance of the class
- Therefore **main( )** is called by the java interpreter before any objects are made
- The keyword “**void**” is used to tell the compiler that main() doesnot return any value
- **main( )** is a method called when a java application begins
- **String args[ ]** declares a parameter named arguments, which is an array of instances of the class “**String**”
- **args** receives any command line arguments present when the pgm is executed
- Open a text editor, like Notepad on windows andTextEdit on Mac. Copy the above program and paste it in the text editor. You can also use IDE like NetBeans, Eclipse to run the java program
- We must save this pgm in a file with “**.java**” extension **Example.java**
- “**Example.java**” ensuring that the filename contains the class name containing the main method. This file is called the source file
- To compile the pgm, we must run the Java Compiler “ **javac** “ with the name of the source file on the command line

Or

- In this step, we will compile the program. For this, open **Terminal** in Linux To compile the program, type the following command and hit enter. javac Example.java
  - Set path in Linux: ex: export PATH=\$PATH:/home/jdk1.6.0/bin/
  - If everything is correct, the javac compiler creates a file called “**Example.class**” that contain byte code version of the pgm
  - To run a stand-alone pgm, we need to use the Java Interpreter “**java**” with the file name only at the command line
  - Now, the interpreter looks for the main method in the pgm & begin execution from there
  - The o/p of the pgm is the string in println. **This is a simple Java Program**
-

## UNIT-I

### Topics to be covered:

Ex1: AIM: To display a message on the Screen

#### **Example 1- First Java Program**

**Line1:** /\* Call this file “Example.java”.\*/

**Line2:** public class Example

{

**Line3:** // Your program starts execution with a call to main()

**Line4:**           public static void main( String args[ ])

{

**Line5:**           System.out.println(“This is a simple Java Program”);

**Line6:**           } // End of main method

**Line7:** } // End of Example Class

**Expected Output:** This is a simple Java Program

**Whether class contains main( ) method or not And Whether main( ) method is declared according to requirement or not these things will not be checked by compiler.**

**At runtime, JVM is responsible to check these things**

```
class Sample
```

```
{
```

```
}
```

### Java Sample.java

#### Java Sample

**Runtime Exception: NoSuchMethodError:main**

**Exp: At runtime if JVM is unable to find the required main() method then it give us runtime exception : NoSuchMethodError:main**

Now let us understand in detail about

**public static void main ( String args [ ] )**

**public:** To call by JVM from anywhere

**static:** without existing object also JVM has to call this method & main method nowhere related to any object

**void :** main() method will not return anything to JVM

**main :** this is the name which is configured inside JVM

**String [] args :** Command line arguments

The above syntax is very Strict but at the same time few changes are acceptable:

1. The order of modifiers is not important

Instead of “public static we can write “static public” also

2. We can declare “String[]” in any form

`main(Stirng [ ]args)`

`main(String args[])`

`main(String[] args)`

3. Instead of ‘args’ we can use any valid java identifier

`(String[] a)`

4. We can replace String [] with var arg parameter

`Main(String [] args) → main(String... args)`

5. We can declare main() method with following access modifiers such as :

`final`

`Synchronized`

`Strictfp- strict floating pt`

`class Sample`

`{`

`final static synchronized strictfp public void main(String... rgukt)`

```
{  
    SOPLN("VOID MAIN SYNTAX");  
}  
}
```

Javac Sample.java

Java Sample

OP: VOID MAIN SYNTAX

Which of the following are valid main method declarations

1. **public static void Main(String []args)**
2. **public static int main(String []args)**
3. **public static void main(String args)**
4. **public final strictfp synchronized void main(String []args)**
5. **public static final strictfp synchronized void main(String[] args)**
6. **public static void main(String... args)**

Note:

In which of the above cases we will get CTE

We will not get compile time error anywhere but at runtime we will get exception in all cases except last two cases

Now let us understand few special cases:

**Case1:**

**Overloading of the main method possible but JVM will always call String[] argument main method only**

The other overloading method we have to call explicitly then it will be executed just as a normal method call

**Class Sample**

```
{  
    psvm(String []a)  
    {  
        S.o.println("String[]")  
    }  
    psvm(int []a)  
    {  
        S.o.println("int[]");  
    }  
}
```

**OP:String[]**

**Case: 2**

Inheritance concept is applicable for the main method. Hence while executing child class if child class doesnot contain main method then parent class main method will be executed.

**Ex:**

**Class P**

```
{  
    Psvm(String []a)  
    {  
        Sopl("PARENT MAIN");  
    }  
}
```

**Class C extends P**

```
{  
}
```

**Execute:**

**java P----- O/p: PARENT MAIN**

**java C----- O/p: PARENT MAIN**

**Case: 3**

It seems overriding concept is applicable for main method but it is not overriding it is method hiding.

```
class p
{
    Psvm(String []args)
    {
        S.o.println("Parent Main");
    }
}

class c extends p
{
    Psvm(String []args)
    {
        S.o.println("Child Main");
    }
}
```

**Execute: java p----- O/p: Parent Main**

**Java c----- O/p: Child Main**

**Note: For main method inheritance and overloading concepts are applicable but overriding concepts is not applicable instead of overriding method hiding concept is applicable.**

**Q: Without writing main() method is it possible to print some statements to the console?**

**A: Yes we can print by using static block.**

**But this rule is applicable until 1.6 version only from 1.7 version onwards main() method is mandatory to print some statements to the console.**

**1.6 – static block**

**RE: NSEM:main**

**1.7- main method is not found in the classs**

**If the class contains main() method whether it is 1.6 or 1.7 version there is no change in execution sequence**

```
class Test  
{  
    static String s="OOPs";  
}  
  
Test.s.length();
```

- Test is a class name
- 's' is a static variable present in Test class of type String
- length( ) is method present in String class

Now lets try to understand

```
class System  
{  
    static PrintStream out;  
}  
  
System.out.println("Hello");
```

- System is class present in java.lang package
- 'out' is a static variable present in system class of type PrintStream
- println( ) is a method present in PrintStream class

## UNIT-I

### **Input & Output Statements in Java**

The package “java.lang” consists a class named '**System**' which contains console I/O methods.

If we want to take the info. from the keyboard & display the info on the screen, java uses Stream objects

The System provides 3 basic streams.

1. **System.in** ----- refers to the Std.i/p stream
2. **System.out** ----- refers to the Std.o/p stream
3. **System.err** ----- refers to the error stream

#### **Standard Input Stream**

- Accepting ( Reading) input is done by use of objects & methods of existing std. I/p stream
- Primitive datatypes are to be converted into object type by using wrapper classes

“Interger” class is used as wrapper class for primitive datatype “int”.

“parseInt( )” method converts string to an int

**Ex:** int a,b;

a=Integer.parseInt(args[0]);                    b=Integer.parseInt(args[1]);

We may also give values to var.s interactively through the keyboard using the

method “readLine ( )”

**Ex:** float f;

```
Int c;

f=Float.valueOf(in.readLine()).floatValue();

c=Interger.parseInt(in.readLine());
```

### **Basic Examples:**

#### **Java Program to read integer value from the Standard Input**

In this program we will see how to read an integer number entered by user. Scanner class is in **java.util package**. It is used for capturing the input of the primitive types like int, double etc. and strings.

#### **Program to read the number entered by user**

We have imported the package **java.util.Scanner** to use the Scanner. In order to read the input provided by user, we first create the object of Scanner by passing System.in as parameter. Then we are using **nextInt( )** method of Scanner class to read the integer.

```
import java.util.Scanner;
public class Demo
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter any number: ");
        // This method reads the number provided using keyboard
        int num = scan.nextInt();
        // Closing Scanner after the use
        scan.close();
        // Displaying the number
        System.out.println("The number entered by user: "+num);
    }
}
```

```

    }
}

```

**Output:**

```

Enter any number: 101
The number entered by user: 101

```

**Standard Output Stream**

System.out class consists of 2 methods to print the strings & values on the screen

1. print( )
2. println( )

**Difference between System.out.print( ) & System.out.println( ) :****1. System.out.print( )**

print( ) method- Cursor does not move to the Next Line it display on Same Line

**For Example:**

```
System.out.print("Java is a Platform-Independent");
```

```
System.out.print("Language");
```

**Output:**

Java is a Platform-Independent Language

**2. System.out.println( )**

println( ) method- after printing the cursor will move to the next line

**For Example:**

```
System.out.println("Java is a Platform-Independent");
```

```
System.out.println("Language");
```

**Output:**

Java is a Platform-Independent

Language

Java has a version of the '+' operator for string concatenation

This enables a string & a value of another datatype

```
int sum=56;
```

```
System.out.println( "sum is" + sum)    O/P : Sum is 56
```

Here, sum is automatically converted to a string & concatenated with the "sum is" ,

which will result in "**Sum is 56**"

## Now let us look into Example Program:

```

1  /* Write a Java Program to understand the difference between print() and println() method*/
2
3  class printTest
4  {
5      public static void main(String args[])
6      {
7          System.out.println("\nTest of print() Method...\n");
8          //test of print()
9          System.out.print("Java is Platform Independent");
10         System.out.print("Language");
11
12         System.out.println("\n\nTest of println() Method...\n");
13         // test of println()
14         System.out.println("Java is Platform Independent");
15         System.out.println("Language");
16     }
17
18
19 }

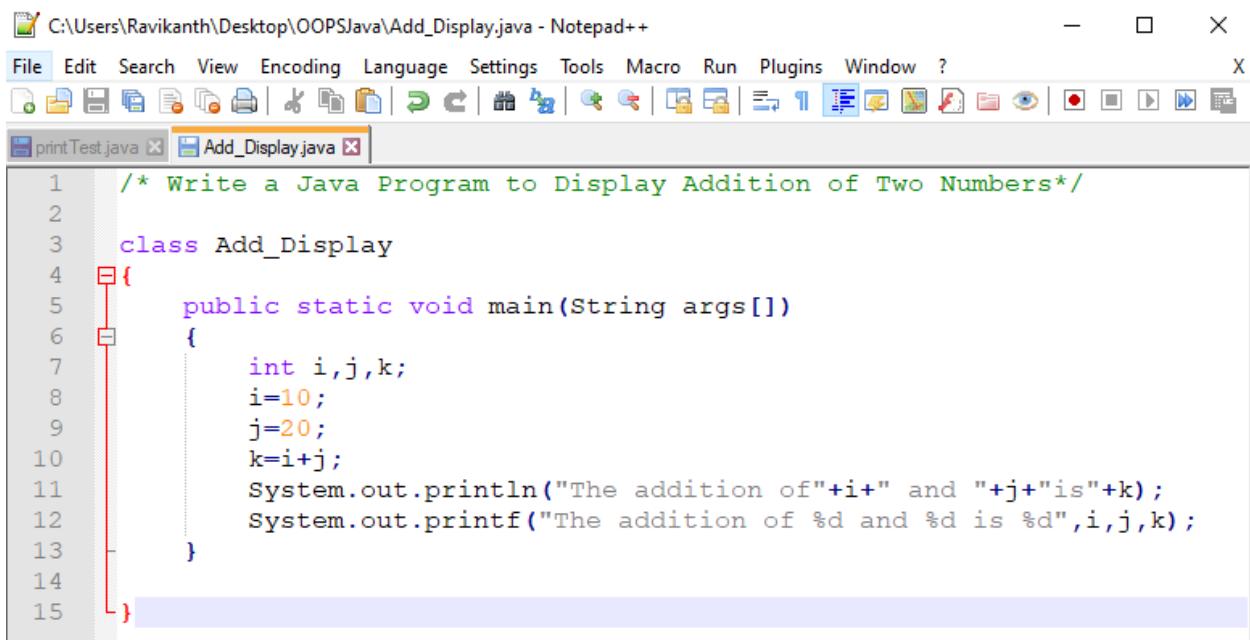
```

## Output Screen:

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac printTest.java
C:\Users\Ravikanth\Desktop\OOPSJava>java printTest
Test of print() Method...
Java is Platform IndependentLanguage
Test of println() Method...
Java is Platform Independent
Language
C:\Users\Ravikanth\Desktop\OOPSJava>

```



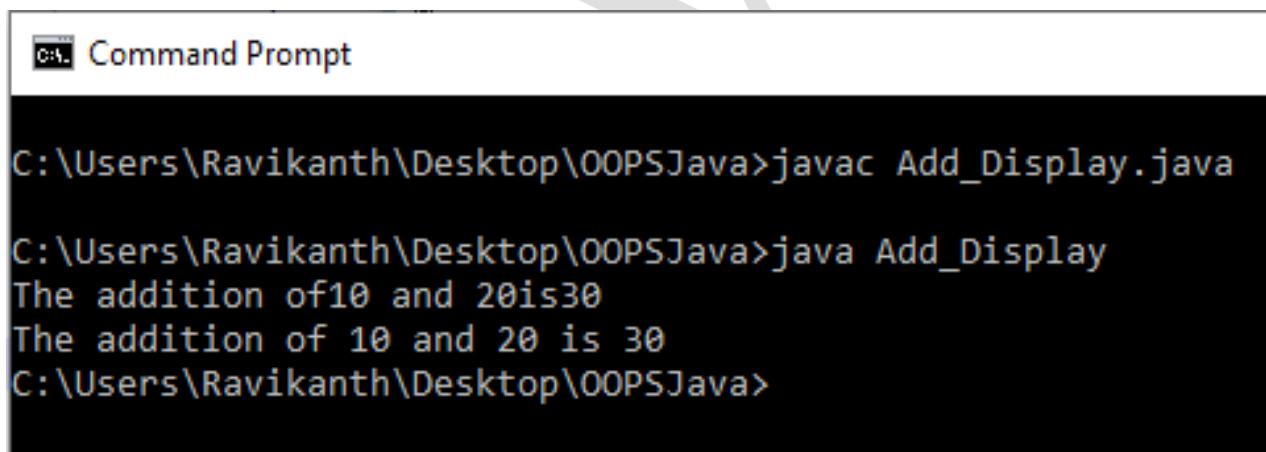
C:\Users\Ravikanth\Desktop\OOPSJava\Add\_Display.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

printTest.java Add\_Display.java

```
1  /* Write a Java Program to Display Addition of Two Numbers*/
2
3  class Add_Display
4  {
5      public static void main(String args[])
6      {
7          int i,j,k;
8          i=10;
9          j=20;
10         k=i+j;
11         System.out.println("The addition of "+i+" and "+j+" is "+k);
12         System.out.printf("The addition of %d and %d is %d",i,j,k);
13     }
14
15 }
```

## Output:



Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Add_Display.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Add_Display
The addition of 10 and 20 is 30
The addition of 10 and 20 is 30
C:\Users\Ravikanth\Desktop\OOPSJava>
```

## Comments in Java program

- The java comments are statements that are not executed by the compiler and interpreter.
- The comments can be used to provide information or explanation about the variable, method, class or any statement.
- It can also be used to hide program code for specific time.
- Comments are like helping text in your java program.

There are 3 types of comment sts. available in Java

### 1. Single line Comment ( // )

**The single line comment is used to comment only one line.**

```
// This is a Single Line Comment
```

### 2. Multi line Comment ( /\*      \*/ )

**The multi line comment is used to comment multiple lines of code**

```
/* This is a Multi Line
```

```
Comment*/
```

### 3. Documentation Comment ( /\*\*      \*/ )

**The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool**

```
/** This is a Documentation Comment
```

```
*/
```

## Literals

Sequence of characters ( digits, letters & characters) that represent constant values

### Types of Literals

- A literal is a fixed value that we assign to a variable in a Program.

```
int num=10;
```

Here value 10 is a Integer literal.

```
char ch = 'A';
```

Here A is a char literal

- **Integer Literal**

Integer literals are assigned to the variables of data type byte, short, int and long.

```
byte b = 100;
```

```
short s = 200;
```

```
int num = 13313131;
```

```
long l = 928389283L;
```

- **Float Literals**

Used for data type float and double.

```
double num1 = 22.4;
```

```
float num2 = 22.4f;
```

- Note: Always suffix float value with the “f” else compiler will consider it as double.

- **Char and String Literal**

Used for char and String type.

```
char ch = 'R';
```

```
String str = "Programming in Java";
```

## Separator

Separators help define the structure of a program.

The separators used in HelloWorld are parentheses, ( ), braces, { }, the period, ., and the semicolon, ;.

The table lists the six Java separators (nine if you count opening and closing separators as two).

**Parentheses ( )** - Encloses arguments in method definitions and calling; adjusts precedence in arithmetic expressions; surrounds cast types and delimits test expressions in flow control statements

Ex:

Method argument - **area(5,10)**

Precedence in arithmetic expressions - **a\*(b+c)**

Type casting - **int(5.32)=5 & float(4)=4.0**

**CurlyBraces { }** - Defines blocks of code/statements and it also automatically initializes arrays

Defie blocks of codes - **void area(int h, int w)**

{

Int are =w\*h;

}

Initializes arrays - **int a{ 1,2,3,4};**

**Square Bracket [ ]** - Declares array types and dereferences array values

**Semicolon ;** - Used to terminates statements, Used for initialization in for loop

Terminates statements - **int x=10;**

**Single Dot/Period .** – It Selects a field or a method from an object, separates package names from sub-package and class names

Selects field or method from object – **obj1.height;**

Separate package from subpackage - **import java.util.\*;**

**Colon :-** Used after loops labels

**Single coma , -** separates successive identifiers in variable declarations; chains statements in the

test, expression of a for loop

```
int x,y;
```

**Angle Brackets <> -** We define Wrapper classes, Collections and Generics

## Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values.

### Example:

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
class CommandLineExample
{
    public static void main(String args[ ])
    {
        System.out.println("Your first argument is: "+args[0]);
    }
}
```

compile > javac CommandLineExample.java

run > java CommandLineExample sonoo

The screenshot shows the Notepad++ interface with a file named 'CMD.java'. The code is a simple Java program that prints "WE are working wth CLA" and "Your first argument is: <empty>". The code is as follows:

```
1 class CommandLineExample
2 {
3     public static void main(String args[ ])
4     {
5         System.out.println("WE are working wth CLA");
6         System.out.println("Your first argument is: "+args[0]);
7     }
8 }
9
```

The screenshot shows a Command Prompt window with the following session:

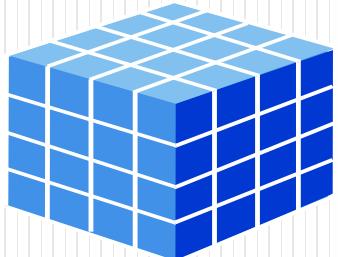
```
C:\Users\Ravikanth\Desktop\OOPSJava>javac CMD.java
C:\Users\Ravikanth\Desktop\OOPSJava>java CommandLineExample
WE are working wth CLA
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
at CommandLineExample.main(CMD.java:6)

C:\Users\Ravikanth\Desktop\OOPSJava>java CommandLineExample RGUKT-BASAR
WE are working wth CLA
Your first argument is: RGUKT-BASAR

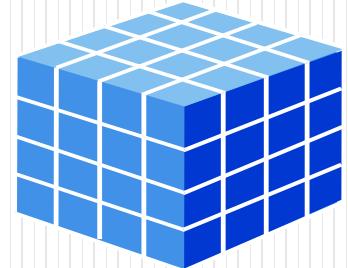
C:\Users\Ravikanth\Desktop\OOPSJava>
```

# OOOPS Using JAVA

## UNIT-II



Mr. K. RAVIKANTH  
Assistant Professor  
Department of CSE  
Rajiv Gandhi University of Knowledge Technologies, Basar.



- **Elements:** Type casting
- **Operators:** Arithmetic, Logical, Bit wise operator, Increment and Decrement, Relational, Assignment, Conditional, Special operator, Expressions – Evaluation of Expressions

## ► Type conversion and type casting:

- In Java, there are two kinds of data types – primitives and references.
  - Primitives include int, float, long, double etc.
  - References can be of type classes, interfaces, arrays.
- A value can change its type either implicitly or explicitly.
- **Implicit and Explicit changes:**
- There are situations in which the system itself changes the type of an expression implicitly based on some requirement.
- **Example:**
  - int num1 = 4;
  - double num2;
  - double d = num1 / num2; //num1 is also converted to double
- This kind of conversion is called **automatic type conversion**.

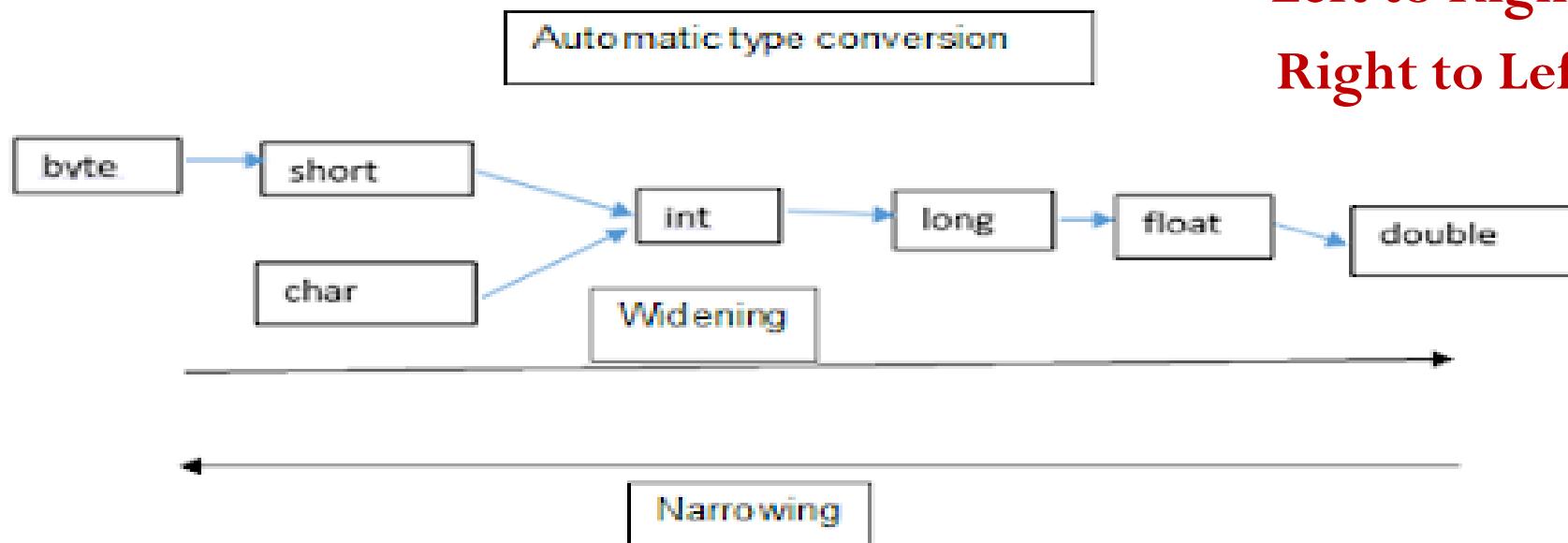
**► Type conversion and type casting:**

- When you assign value of one data type to another, the two types might not be compatible with each other.
- If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly.
- For example, assigning an int value to a long variable.
- **Two types of Conversions**
  - Widening or Automatic Type Conversion [ Implicit ]
  - Narrowing or Explicit Conversion

► Type conversion and type casting:

► Widening or Automatic Type Conversion

- Compiler is responsible to perform Implicit Type casting
- Smaller Datatype value to Bigger Datatype value
- Also known as Widening [ Keeping small value in Big value ] or also known as Upcasting
- There is no loss of information in ITC



Left to Right => Implicit Type Casting  
Right to Left => Explicit Type Casting

► Type conversion and type casting:

► Widening or Automatic Type Conversion

```

1 //Java program to illustrate Implicit type conversion
2 class Implicit
3 { public static void main(String[] args)
4 {
5     int x='a';
6     System.out.println(" Print x value is : "+x);
7 }
8 }
9

```

Command Prompt

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac Implicit.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Implicit
Print x value is : 97

```

Here compiler converted char to int automatically by ITC

```

1 //Java program to illustrate Implicit type conversion
2 class Implicit_Double
3 { public static void main(String[] args)
4 {
5     double d=10;
6     System.out.println(" Print d value is : "+d);
7 }
8 }
9

```

Command Prompt

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac Implicit_Double.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Implicit_Double
Print d value is : 10.0

```

Here compiler converted int to double automatically by ITC

► Type conversion and type casting:

► Widening or Automatic Type Conversion

The screenshot shows a Notepad++ interface with two tabs: "Nested\_Variables.java" and "Widening.java". The "Widening.java" tab is active, displaying the following code:

```
1  class Widening
2  {    public static void main(String[] args)
3  {      int i = 100;
4
5      //automatic type conversion
6      long l = i;
7      //automatic type conversion
8      float f = l;
9      System.out.println("Int value "+i);
10     System.out.println("Long value "+l);
11     System.out.println("Float value "+f);
12
13   }
14 }
```

A red arrow points from the closing brace of the main method back to the opening brace of the class definition.

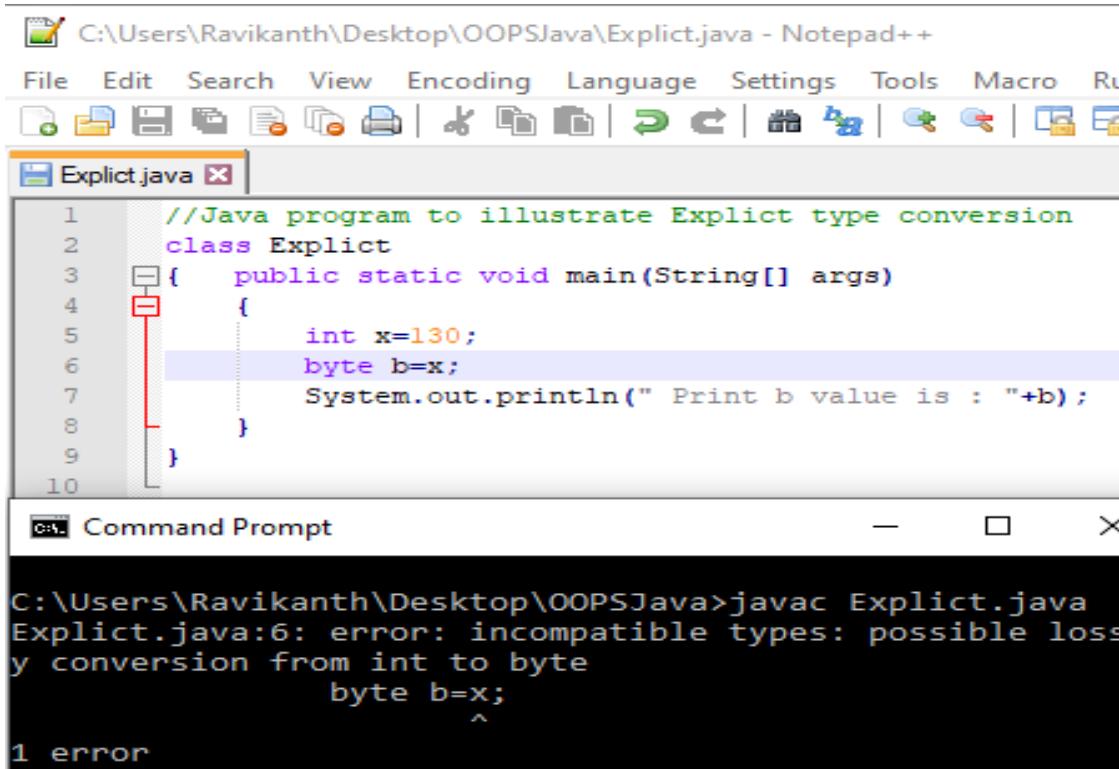
To the right of the editor is a terminal window titled "Command Prompt" showing the execution of the code:

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Widening.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Widening
Int value 100
Long value 100
Float value 100.0
```

## ► Type conversion and type casting:

### ► Narrowing or Explicit Conversion

- If we want to assign a value of larger data type to a smaller data type
- we perform explicit type casting or narrowing.



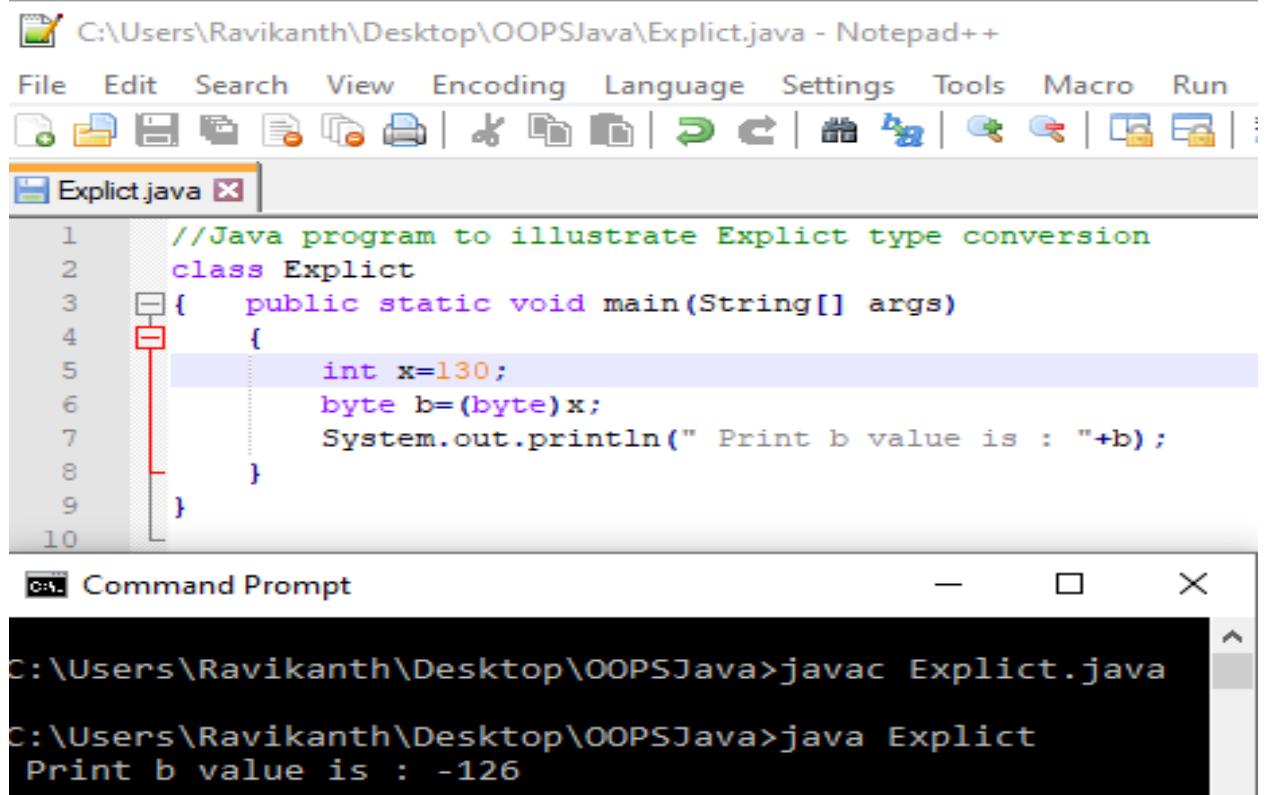
```

C:\Users\Ravikanth\Desktop\OOPSJava\Explict.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run
Explit.java x
1 //Java program to illustrate Explict type conversion
2 class Explict
3 {   public static void main(String[] args)
4     {
5       int x=130;
6       byte b=x;
7       System.out.println(" Print b value is : "+b);
8     }
9 }
10

Command Prompt
C:\Users\Ravikanth\Desktop\OOPSJava>javac Explict.java
Explict.java:6: error: incompatible types: possible loss
y conversion from int to byte
    byte b=x;
               ^
1 error

```

**Note: Here x is of 4 bytes and b is of 1 byte. There is a chance of loss of info. So compiler is not accepting**



The screenshot shows a Java development environment with the following components:

- Notepad++ Editor:** Displays the Java code for `Explict.java`. The line `byte b=x;` is highlighted in purple, indicating a syntax error.
- Command Prompt:** Shows the compilation command `C:\Users\Ravikanth\Desktop\OOPSJava>javac Explict.java` and the resulting error message: `Explict.java:6: error: incompatible types: possible loss`.
- Command Prompt:** Shows the execution command `C:\Users\Ravikanth\Desktop\OOPSJava>java Explict` and its output: `Print b value is : -126`.

**Note: There is a chance of Loss of Data**

## ► Narrowing or Explicit Conversion

- Programmer is responsible to perform ETC
- Bigger Datatype value to Smaller Datatype value
- Hence called Narrowing or DownCasting
- There may be loss of information

Double → Float → Long → Int → Short → Byte

Narrowing or Explicit Conversion

The screenshot shows a Notepad++ window with the file 'Narrowing.java' open. The code demonstrates narrowing from a double to a long, and then from a long to an int. The output in the Command Prompt shows that the fractional part of the double is lost during these conversions.

```

C:\Users\Ravikanth\Desktop\OOPSJava\Narrowing.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run
Narrowing.java x
1 //Java program to illustrate explicit type conversion
2 class Narrowing
3 {
4     public static void main(String[] args)
5     {
6         double d = 100.04;
7         //explicit type casting
8         long l = (long)d;
9         //explicit type casting
10        int i = (int)l;
11        System.out.println("Double value "+d);
12        //fractional part lost
13        System.out.println("Long value "+l);
14        //fractional part lost
15        System.out.println("Int value "+i);
16    }
}

```

Command Prompt

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac Narrowing.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Narrowing
Double value 100.04
Long value 100
Int value 100

```

## ► Trace the Program

```
class Simple{  
    public static void main(String[] args){  
        int a=10;  
        float f=a;  
        System.out.println(a);  
        System.out.println(f);  
    }  
}
```

```
class Simple{  
    public static void main(String[] args){  
        float f=10.5f;  
        //int a=f;//Compile time error  
        int a=(int)f;  
        System.out.println(f);  
        System.out.println(a);  
    }  
}
```

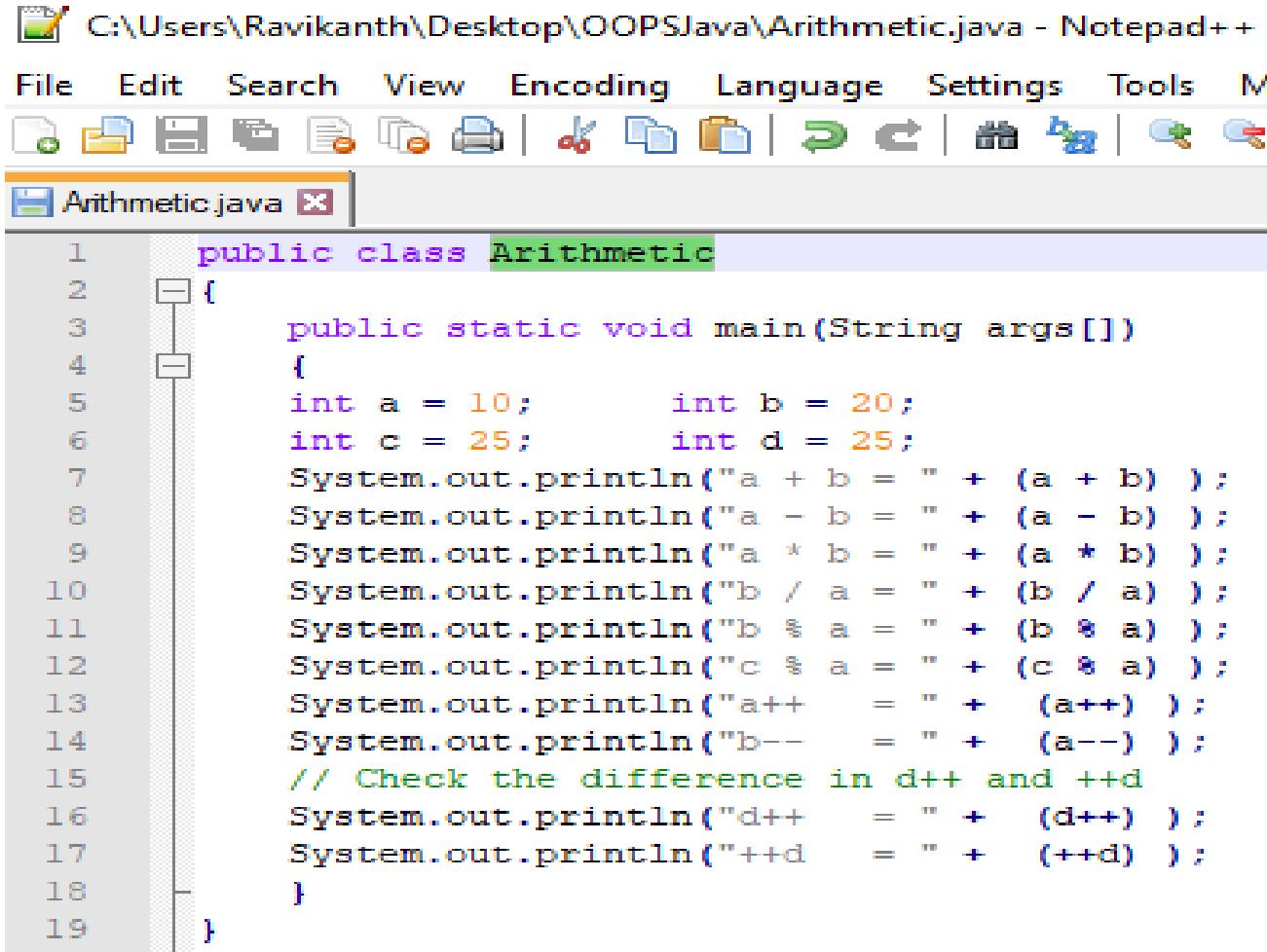
► **Operators, Operator hierarchy, expressions:**

► Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Unary Operators
- Shift Operator
- Ternary Operators

## ► Operators, Operator hierarchy, expressions:

### ► Arithmetic Operators (+,-,\*,/,%)

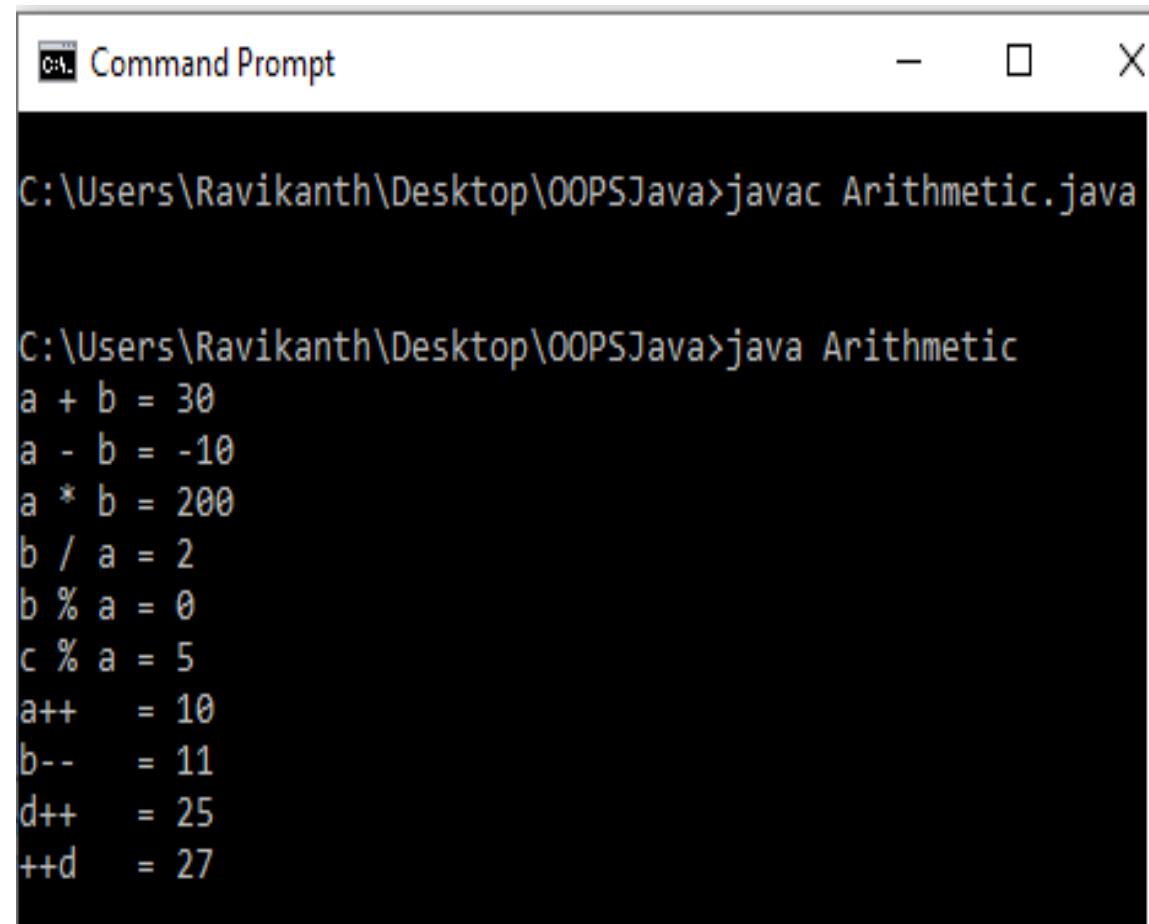


C:\Users\Ravikanth\Desktop\OOPSJava\Arithmetic.java - Notepad++

File Edit Search View Encoding Language Settings Tools M

Arithmetic.java

```
1 public class Arithmetic
2 {
3     public static void main(String args[])
4     {
5         int a = 10;      int b = 20;
6         int c = 25;      int d = 25;
7         System.out.println("a + b = " + (a + b));
8         System.out.println("a - b = " + (a - b));
9         System.out.println("a * b = " + (a * b));
10        System.out.println("b / a = " + (b / a));
11        System.out.println("b % a = " + (b % a));
12        System.out.println("c % a = " + (c % a));
13        System.out.println("a++ = " + (a++));
14        System.out.println("b-- = " + (b--));
15        // Check the difference in d++ and ++d
16        System.out.println("d++ = " + (d++));
17        System.out.println("++d = " + (++d));
18    }
19 }
```



Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Arithmetic.java
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Arithmetic
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++ = 10
b-- = 11
d++ = 25
++d = 27
```

► Operators, Operator hierarchy, expressions:

► Relational Operators(==,!=,>,<,>=,<=)

C:\Users\Ravikanth\Desktop\OOPSJava\RelOP.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro

RelOP.java

```
1 public class RelOP
2 {
3     public static void main(String args[])
4     {
5         int a = 10;
6         int b = 20;
7         System.out.println("a == b = " + (a == b) );
8         System.out.println("a != b = " + (a != b) );
9         System.out.println("a > b = " + (a > b) );
10        System.out.println("a < b = " + (a < b) );
11        System.out.println("b >= a = " + (b >= a) );
12        System.out.println("b <= a = " + (b <= a) );
13    }
14}
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac RelOP.java
C:\Users\Ravikanth\Desktop\OOPSJava>java RelOP
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

## ► Operators, Operator hierarchy, expressions:

### ► Bitwise Operators (&, |, ^, ~ (Binary Ones Complement Operator), <<, >>, <<< (Shift right zero fill operator))

```

1 public class BitOp
2 {
3     public static void main(String args[])
4     {
5         int a = 60; /* 60 = 0011 1100 */
6         int b = 13; /* 13 = 0000 1101 */
7         int c = 0;
8         c = a & b; /* 12 = 0000 1100 */
9         System.out.println("a & b = " + c );
10        c = a | b; /* 61 = 0011 1101 */
11        System.out.println("a | b = " + c );
12        c = a ^ b; /* 49 = 0011 0001 */
13        System.out.println("a ^ b = " + c );
14        c = ~a; /* -61 = 1100 0011 */
15        System.out.println("~a = " + c );
16        c = a << 2; /* 240 = 1111 0000 */
17        System.out.println("a << 2 = " + c );
18        c = a >> 2; /* 215 = 1111 */
19        System.out.println("a >> 2 = " + c );
20    }
21 }
```

### Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac BitOp.java
C:\Users\Ravikanth\Desktop\OOPSJava>java BitOp
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2 = 15
```

► Operators, Operator hierarchy, expressions:

► Logical Operators(&&, || ,!)

The screenshot shows a Notepad++ interface with two windows. The left window displays a Java program named LogicOPs.java. The right window is a Command Prompt window showing the execution and output of the Java code.

**Notepad++ Window (Left):**

File Edit Search View Encoding Language Settings Tools Macro Run

LogicOPs.java

```
1 public class LogicOPs
2 {
3     public static void main(String args[])
4     {
5         boolean a = true;
6         boolean b = false;
7         System.out.println("a && b = " + (a&&b));
8         System.out.println("a || b = " + (a||b) );
9         System.out.println("! (a && b) = " + !(a && b));
10    }
11 }
```

**Command Prompt Window (Right):**

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac LogicOPs.java
C:\Users\Ravikanth\Desktop\OOPSJava>java LogicOPs
a && b = false
a || b = true
!(a && b) = true
```

► Operators, Operator hierarchy, expressions:

► Assignment Operators ( $=, +=, -=, *=, /=, \% =, <<=, >>=, \&=, \wedge=, | =$ )

- $=$  Simple assignment operator, Assigns values from right side operands to left side operand
- $+=$  Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand  
**C += A is equivalent to C = C + A**
- $-=$  Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand  
**C -= A is equivalent to C = C - A**
- $*=$  Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand  
**C \*= A is equivalent to C = C \* A**
- $/=$  Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand  
**C /= A is equivalent to C = C / A**
- $\% =$  Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand  
**C \% = A is equivalent to C = C \% A**
- $<<=$  Left shift AND assignment operator  
**C <<= 2 is same as C = C << 2**
- $>>=$  Right shift AND assignment operator  
**C >>= 2 is same as C = C >> 2**
- $\&=$  Bitwise AND assignment operator  
**C \&= 2 is same as C = C \& 2**
- $\wedge=$  bitwise exclusive OR and assignment operator  
**C \wedge= 2 is same as C = C \wedge 2**
- $| =$  bitwise inclusive OR and assignment operator  
**C |= 2 is same as C = C | 2**

## ► Operators, Operator hierarchy, expressions:

# ► Assignment Operators(=,+=,-=,\*=,/=%,<<=,>>=,&=,&^=, |=)

C:\Users\Ravikanth\Desktop\OOPSJava\AssignOp.java - Notepad++

File Edit Search View Encoding Language Settings Tools Mi



AssignOp.java x

```

1  public class AssignOp
2  {
3      public static void main(String args[])
4      {
5          int a = 10;
6          int b = 20;
7          int c = 0;
8          c = a + b;
9          System.out.println("c = a + b = " + c );
10         c += a ;
11         System.out.println("c += a = " + c );
12         c -= a ;
13         System.out.println("c -= a = " + c );
14         c *= a ;
15         System.out.println("c *= a = " + c );
16         a = 10;
17         c = 15;
18         c /= a ;
19         System.out.println("c /= a = " + c );

```

```

20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
}
```

```

a = 10;
c = 15;
c %= a ;
System.out.println("c %= a = " + c );
c <= 2 ;
System.out.println("c <= 2 = " + c );
c >= 2 ;
System.out.println("c >= 2 = " + c );
c >>= 2 ;
System.out.println("c >>= 2 = " + c );
c &= a ;
System.out.println("c &= 2 = " + c );
c ^= a ;
System.out.println("c ^= a = " + c );
c |= a ;
System.out.println("c |= a = " + c );

```

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac AssignOp.java
C:\Users\Ravikanth\Desktop\OOPSJava>java AssignOp
c = a + b = 30
c += a = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a = 5
c <= 2 = 20
c >= 2 = 5
c >>= 2 = 1
c &= 2 = 0
c ^= a = 10
c |= a = 10

```

Command Prompt

► Operators, Operator hierarchy, expressions:

► Misc Operators

► Conditional Operator ( ?: )

- Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable.
- The operator is written as: **variable x = (expression) ? value if true : value if false**

```
1 public class T0
2 {
3     public static void main(String args[ ])
4     {
5         int a , b;
6         a = 10;
7         b = (a == 1) ? 20: 30;
8         System.out.println( "Value of b is : " + b );
9         b = (a == 10) ? 20: 30;
10        System.out.println( "Value of b is : " + b );
11    }
12 }
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac T0.java
C:\Users\Ravikanth\Desktop\OOPSJava>java T0
Value of b is : 30
Value of b is : 20
```

► Operators, Operator hierarchy, expressions:

► Misc Operators

► instanceof Operator

- This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type).

The **instanceOf** operator is written as:

**( Object reference variable ) instanceof (class/interface type)**

- If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true.

```
class Vehicle {  
    public class Car extends Vehicle {  
        public static void main(String args[]){  
            Vehicle a = new Car();  
            boolean result = a instanceof Car;  
            System.out.println( result);  
        }  
    }  
}
```

**Output:  
true**

## ► **Operators, Operator hierarchy, expressions:**

### ► **Operator hierarchy**

- Operator precedence/hierarchy determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.
- Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

## ► Operators, Operator hierarchy, expressions:

### ► Operator hierarchy

| Category       | Operator   | Associativity |
|----------------|--|---------------|
| Postfix        | <code>o [] . (dot operator)</code>                             | Left to right |
| Unary          | <code>++ -- ! ~</code>   | Right to left |
| Multiplicative | <code>* / %</code>   | Left to right |
| Additive       | <code>+ -</code>   | Left to right |
| Shift          | <code>&gt;&gt; &gt;&gt;&gt; &lt;&lt;</code>                    | Left to right |
| Relational     | <code>&gt; &gt;= &lt; &lt;=</code>                             | Left to right |
| Equality       | <code>== !=</code>   | Left to right |
| Bitwise AND    | <code>&amp;</code>   | Left to right |
| Bitwise XOR    | <code>^</code>   | Left to right |
| Bitwise OR     | <code> </code>   | Left to right |
| Logical AND    | <code>&amp;&amp;</code>  | Left to right |
| Logical OR     | <code>  </code>  | Left to right |
| Conditional    | <code>? :</code>   | Right to left |
| Assignment     | <code>= += -= *= /= %= &gt;&gt;= &lt;&lt;= &amp;= ^=  =</code> | Right to left |
| Comma          | <code>,</code>   | Left to right |

## ► Operators, Operator hierarchy, expressions:

### ► Expressions:

- Expressions consist of variables, operators, literals and method calls that evaluates to a single value.
- Expressions are the basic way to create values. Expressions are created by combining literals (constants), variables, and method calls by using operators. Parentheses can be used to control the order of evaluation.
- The order in which expressions are evaluated is basically left to right, with the exception of the assignment operators. It may be changed by the use of parentheses. See the expression summary for the precedence.

## ► Trace the Program

```
class OperatorExample{  
public static void main(String args[]){  
int x=10;  
System.out.println(x++);//10 (11)  
System.out.println(++x);//12  
System.out.println(x--);//12 (11)  
System.out.println(--x);//10  
}}
```

```
class OperatorExample{  
public static void main(String args[]){  
int a=10;  
int b=10;  
System.out.println(a++ + ++a);//10+12=22  
System.out.println(b++ + b++);//10+11=21  
}}
```

## ► Trace the Program

```
class OperatorExample{  
public static void main(String args[]){  
int a=10;  
int b=-10;  
boolean c=true;  
boolean d=false;  
System.out.println(~a);//-11 (minus of total positive value whi  
|  
System.out.println(~b);//9 (positive of total minus, positive starts from 0)  
System.out.println(!c);//false (opposite of boolean value)  
System.out.println(!d)//true  
}}
```

```
class OperatorExample{  
public static void main(String args[]){  
int a=10;  
int b=5;  
System.out.println(a+b);//15  
System.out.println(a-b);//5  
System.out.println(a*b);//50  
System.out.println(a/b);//2  
System.out.println(a%b);//0  
}}
```

## ► Trace the Program

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

```
class OperatorExample{
public static void main(String args[]){
//For positive number, >> and >>> works same
System.out.println(20>>2);
System.out.println(20>>>2);
//For negative number, >>> changes parity bit (MSB) to 0
System.out.println(-20>>2);
System.out.println(-20>>>2);
}}
```

## ► Trace the Program

```
class OperatorExample{  
public static void main(String args[]){  
int a=10;  
int b=5;  
int c=20;  
System.out.println(a<b&&a<c);//false && true = false  
System.out.println(a<b&a<c);//false & true = false  
}}
```

```
class OperatorExample{  
public static void main(String args[]){  
int a=10;  
int b=5;  
int c=20;  
System.out.println(a<b&&a++<c);//false && true = false  
System.out.println(a);//10 because second condition is not checked  
System.out.println(a<b&a++<c);//false && true = false  
System.out.println(a);//11 because second condition is checked  
}}
```

## ► Trace the Program

```
class OperatorExample{  
public static void main(String args[]){  
int a=10;  
int b=5;  
int c=20;  
System.out.println(a>b||a<c);//true || true = true  
System.out.println(a>b|a<c);//true | true = true  
//|| vs |  
System.out.println(a>b||a++<c);//true || true = true  
System.out.println(a);//10 because second condition is not checked  
System.out.println(a>b|a++<c);//true | true = true  
System.out.println(a);//11 because second condition is checked  
}}
```

```
class OperatorExample{  
public static void main(String args[]){  
int a=2;  
int b=5;  
int min=(a<b)?a:b;  
System.out.println(min);  
}}
```

## ► Trace the Program

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

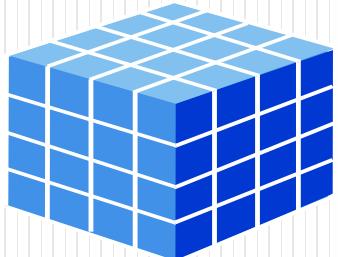
A photograph of a person's hand holding a white rectangular card. The card contains the following text:

**For Details Contact Me @ :**  
**9247448766**  
**ravikanth.iiit@rgukt.ac.in**

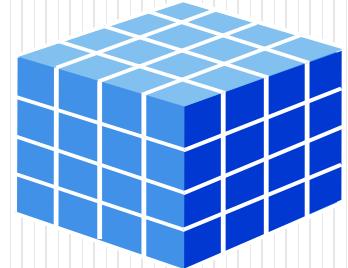
The background of the slide shows a large, semi-transparent image of a hand holding a similar white card.

# OOOPS Using JAVA

## UNIT-II



Mr. K. RAVIKANTH  
Assistant Professor  
Department of CSE  
Rajiv Gandhi University of Knowledge Technologies, Basar.



- **Elements:** Constants, Variables, Data types, Scope of variables, Type casting
- **Operators:** Arithmetic, Logical, Bit wise operator, Increment and Decrement, Relational, Assignment, Conditional, Special operator, Expressions – Evaluation of Expressions
- **Decision making and Branching:** Simple if statement, if, else statement, Nesting if, else, else if Ladder, switch statement
- **Decision making and Looping:** While loop, do-While loop, for loop, break, labelled loop, continue Statement
- **Simple programs Arrays:** One Dimensional Array, Creating an array, Array processing, Multidimensional Array, Vectors, Wrapper classes, Simple programs

# Variables

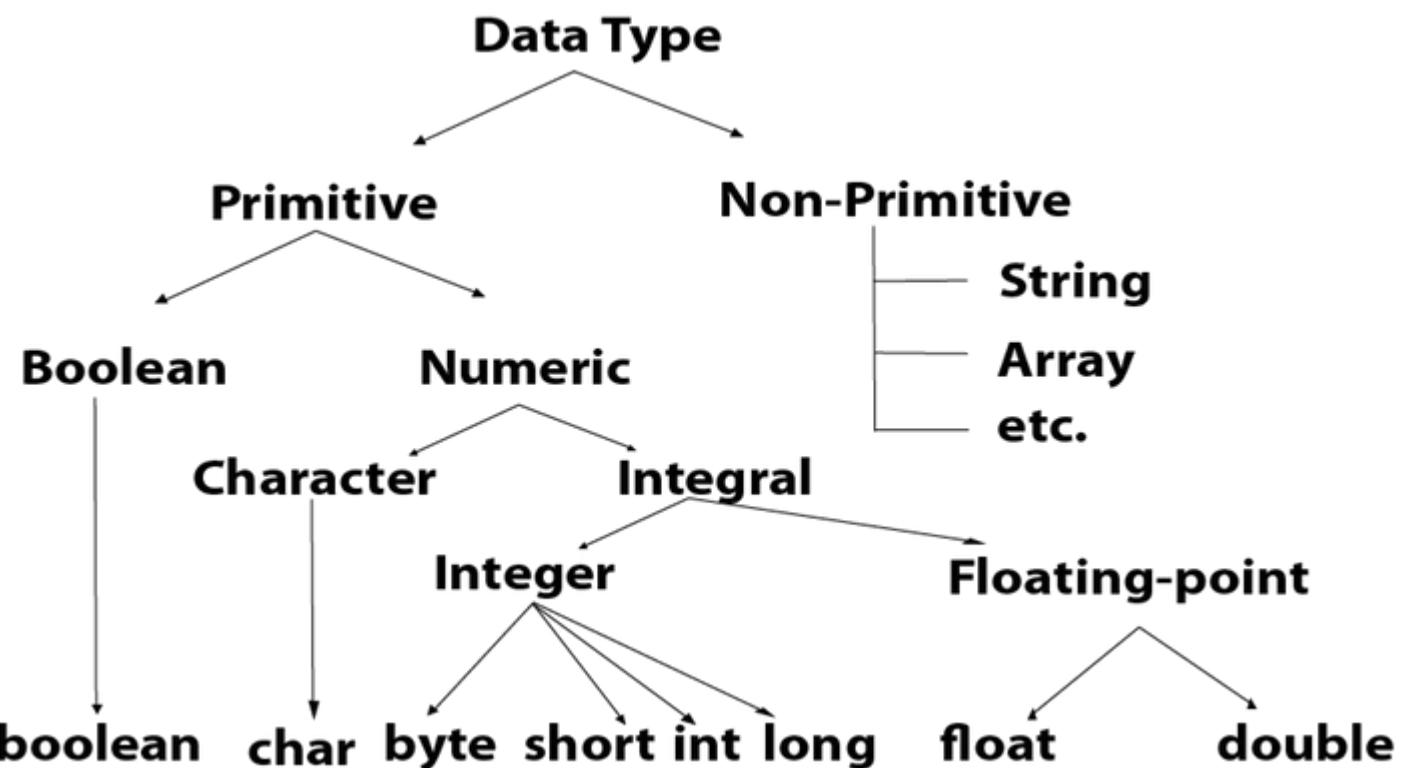
- Variable is a symbolic name refer to a memory location used to store values that can change during the execution of a program
- Java declares its variable in the following manner

```
int           student_strength = 192; // variable declaration  
[ Data Type] [ Identifier]      [Literal]
```

- A variable declaration involves specifying the type (data type), name (identifier), and value(literal) according to the type of the variable.
- All variables in Java have to be declared before they can be used, that is why Java is termed as a **Strongly Typed Language**

## ► Data Types

- 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 String, Array, Classes and Interfaces.



## ► Data Types

- Primitive data types:
- A primitive data type specifies the size and type of variable values, and it has no additional methods.
- There are eight primitive data types in Java:

| Data Type      | Size    | Description   |
|----------------|---------|---|
| <b>byte</b>    | 1 byte  | Stores whole numbers from -128 to 127   |
| <b>short</b>   | 2 bytes | Stores whole numbers from -32,768 to 32,767                                       |
| <b>int</b>     | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647                         |
| <b>long</b>    | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| <b>float</b>   | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits           |
| <b>double</b>  | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits               |
| <b>boolean</b> | 1 bit   | Stores true or false values   |
| <b>char</b>    | 2 bytes | Stores a single character/letter or ASCII values                                  |

## ► Data Types

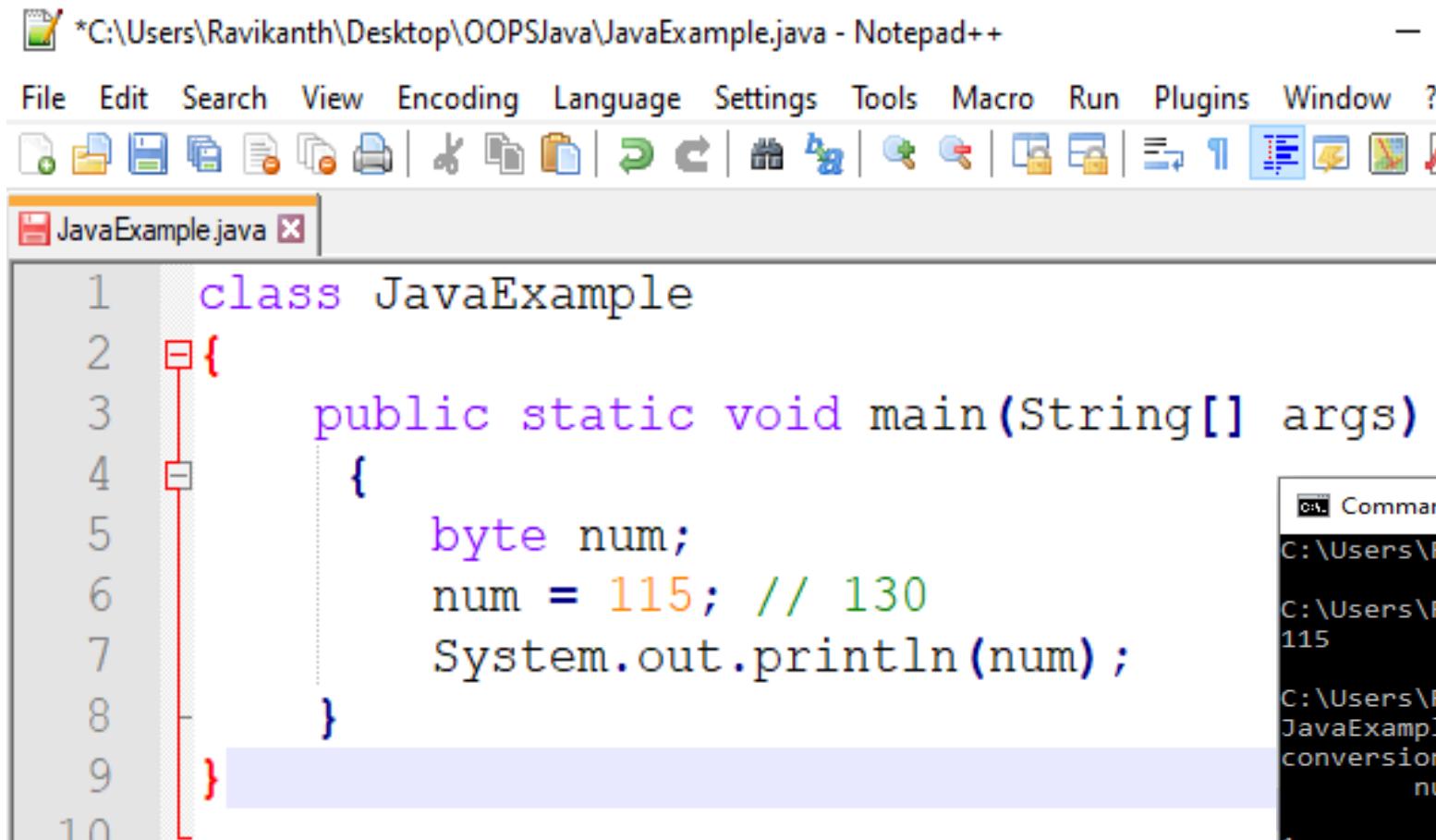
### ► Primitive data types:

► Primitive datatypes are predefined by the language and named by a keyword.

## ► **byte**

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

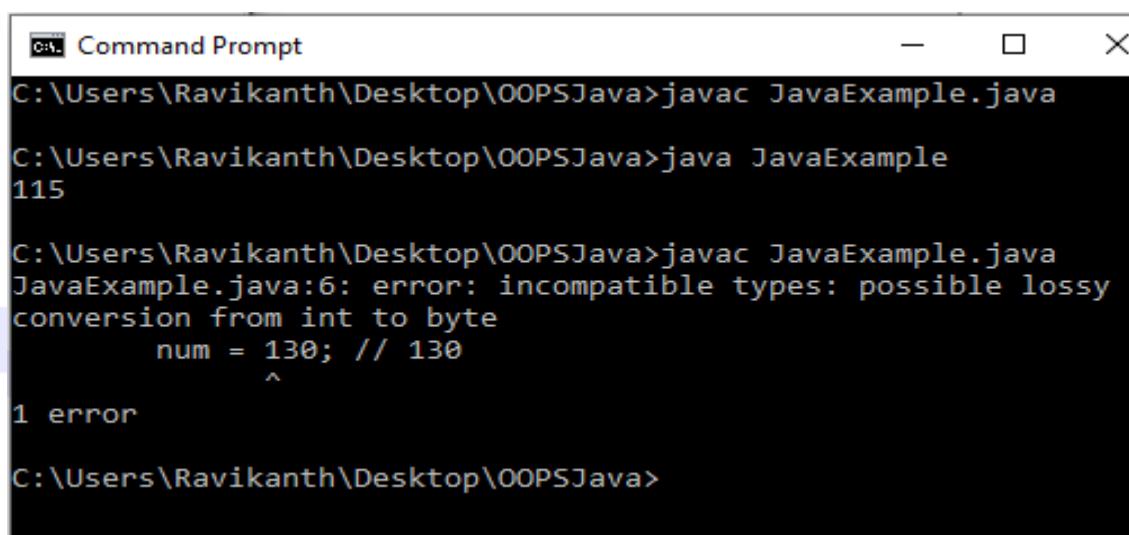
## Example:



The screenshot shows a Notepad++ window with the file 'JavaExample.java' open. The code is as follows:

```
1 class JavaExample
2 {
3     public static void main(String[] args)
4     {
5         byte num;
6         num = 115; // 130
7         System.out.println(num);
8     }
9 }
10
```

Try the same program by assigning value assigning 130 value to variable num, you would get **type mismatch** error because the value 130 is out of the range of byte data type. The range of byte as I mentioned above is -128 to 127.



The screenshot shows a Command Prompt window with the following output:

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac JavaExample.java
C:\Users\Ravikanth\Desktop\OOPSJava>java JavaExample
115

C:\Users\Ravikanth\Desktop\OOPSJava>javac JavaExample.java
JavaExample.java:6: error: incompatible types: possible lossy conversion from int to byte
        num = 130; // 130
                           ^
1 error

C:\Users\Ravikanth\Desktop\OOPSJava>
```

## ► Data Types

### ► Primitive data types:

#### ► short

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

## ► Data Types

### ► Primitive data types:

#### ► int

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

## ► Data Types

### ► Primitive data types:

#### ► long

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

► Primitive data types:

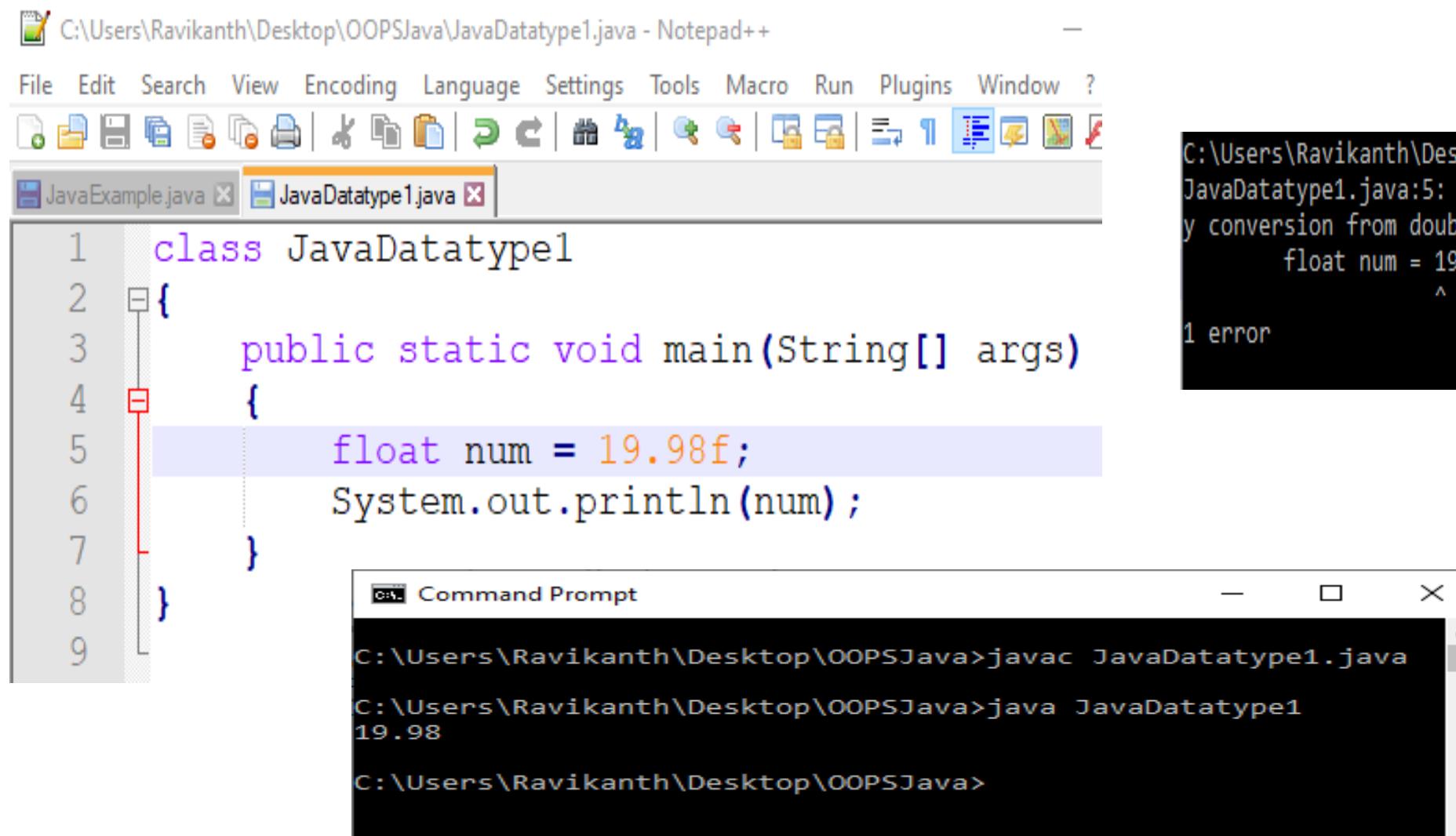
► **float**

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

► **double**

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

## Example:JavaDatatype1



```
C:\Users\Ravikanth\Desktop\OOPSJava\JavaDatatype1.java - Notepad++  
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
JavaExample.java JavaDatatype1.java  
  
1 class JavaDatatype1  
2 {  
3     public static void main(String[] args)  
4     {  
5         float num = 19.98f;  
6         System.out.println(num);  
7     }  
8 }  
  
Command Prompt  
C:\Users\Ravikanth\Desktop\OOPSJava>javac JavaDatatype1.java  
C:\Users\Ravikanth\Desktop\OOPSJava>java JavaDatatype1  
19.98  
C:\Users\Ravikanth\Desktop\OOPSJava>
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac JavaDatatype1.java  
JavaDatatype1.java:5: error: incompatible types: possible lossy conversion from double to float  
    float num = 19.98;  
                           ^  
1 error
```

► Primitive data types:

## ►boolean

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

## ►char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive) [It is a unification of dozens of character sets, such as Latin, Greek, Arabic,]
- Char data type is used to store any character
- Example: char letterA = 'A'

## Example: JavaDataType2

C:\Users\Ravikanth\Desktop\OOPSJava\JavaDatatype2.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

JavaExample.java JavaDatatype1.java JavaDatatype2.java

```
1 class JavaDataType2
2 {
3     public static void main(String[] args)
4     {
5         boolean b = false;
6         System.out.println(b);
7     }
8 }
9
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac JavaDataType2.java
C:\Users\Ravikanth\Desktop\OOPSJava>java JavaDataType2
false
```

C:\Users\Ravikanth\Desktop\OOPSJava\JavaDataType3.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

JavaExample.java JavaDatatype1.java JavaDatatype2.java JavaDatatype3.java

```
1 class JavaDataType3
2 {
3     public static void main(String[] args)
4     {
5         char ch = 'Z';
6         System.out.println(ch);
7     }
8 }
9
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac JavaDataType3.java
C:\Users\Ravikanth\Desktop\OOPSJava>java JavaDataType3
Z
```

## Example: DataTypes

The screenshot shows a Notepad++ window with the file 'DataTypes.java' open. The code defines a class 'DataTypes' with a main method that prints various data types to the console. The code is color-coded: purple for keywords like 'public', 'class', 'int', etc.; green for strings and comments; and blue for floating-point numbers.

```
1 public class DataTypes
2 {
3     public static void main(String[] args)
4     {
5         int myNum = 10;           // integer (whole number)
6         float myFloatNum = 5.992f; // floating point number
7         char myLetter = 'R';      // character
8         boolean myBool = true;    // boolean
9         String myText = "RGUKT";   // String
10        System.out.println(myNum);
11        System.out.println(myFloatNum);
12        System.out.println(myLetter);
13        System.out.println(myBool);
14        System.out.println(myText);
15    }
16 }
```

The screenshot shows a Command Prompt window titled 'Command Prompt'. It displays the command 'javac DataTypes.java' followed by the output of the program, which prints the values of the variables defined in the Java code.

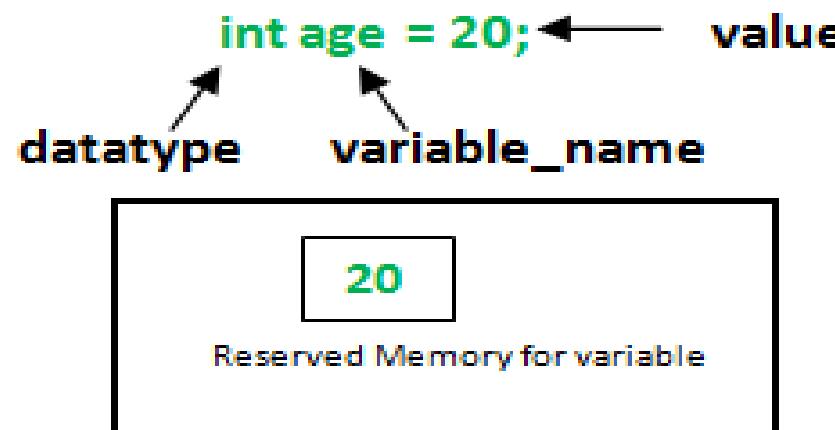
```
C:\Users\Ravikanth\Desktop\OOPSJava>javac DataTypes.java
C:\Users\Ravikanth\Desktop\OOPSJava>java DataTypes
10
5.992
R
true
RGUKT
```

## ► Data Types

- The **main difference between** primitive and **non-primitive** data types are:
- Primitive types are **predefined (already defined) in Java**. Non-primitive types are **created by the programmer and is not defined by Java (except for String)**.
- Non-primitive types can be **used to call methods to perform certain operations**, while primitive types **cannot**.
- A primitive type has always **a value**, while non-primitive types can be **null**.
- A primitive type **starts with a lowercase letter**, while non-primitive types starts with an **uppercase letter**.
- The size of a primitive type **depends on the data type**, while non-primitive types have **all the same size**.
- Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

## ► Variables

- A variable is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.
  - We can declare variables in java as follows:



- **datatype**: Type of data that can be stored in this variable.
- **variable\_name**: Name given to the variable.
- **value**: It is the initial value stored in the variable.

## ► Types of Variables:

► There are three types of variables in Java:

► **Local Variables**

► **Instance Variables**

► **Static Variables**

## ► Local Variables:

► A variable defined **within a block or method or constructor** is called local variable.

► Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.

► Access modifiers cannot be used for local variables.

► The **scope of these variables exists only within the block in which the variable is declared**. i.e. we can access these variable only within that block.

► **Initialization of Local Variable is Mandatory.** Local variables are not given initial default values. Thus, you must assign a value before you use a **local variable**.

## Example Program

The screenshot shows a Notepad++ interface with a Java file named `StudentDetails.java`. The code defines a class `StudentDetails` with a `main` method that creates an object of `StudentDetails` and calls its `StudentAge` method. The `StudentAge` method prints "Student age is : 5" to the console. Below the editor is a terminal window titled "Command Prompt" showing the execution of `javac StudentDetails.java` and the resulting output "Student age is : 5".

```
C:\Users\Ravikanth\Desktop\OOPSJava\StudentDetails.java - Notepad++  
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
StudentDetails.java  
1 public class StudentDetails  
2 {  
3     public void StudentAge()  
4     {  
5         // local variable age  
6         int age = 0;  
7         age = age + 5;  
8         System.out.println("Student age is : " + age);  
9     }  
10    public static void main(String args[])  
11    {  
12        StudentDetails obj = new StudentDetails();  
13        obj.StudentAge();  
14    }  
15 }  
C:\Users\Ravikanth\Desktop\OOPSJava>javac StudentDetails.java  
C:\Users\Ravikanth\Desktop\OOPSJava>java StudentDetails  
Student age is : 5
```

## ► Instance Variables

- Instance variables are **non-static variables and are declared in a class outside any method, constructor or any block.**
- As instance variables are **declared in a class, these variables are created when an object of the class is created with the use of keyword ‘new’ and destroyed when the object is destroyed.**
- Access modifiers can be given for instance variables.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
- **Initialization of Instance Variable is not Mandatory. Its default value is 0**
- Instance Variable can be **accessed only by creating objects.**
- However within static methods and different class ( when instance variables are given accessibility) that should be called using the fully qualified name **ObjectReference.VariableName**

**Example**

```

11  class MarksDemo
12  {
13      public static void main(String args[])
14      {
15          // first object
16          Marks obj1 = new Marks();
17          obj1.engMarks = 50;
18          obj1.mathsMarks = 80;
19          obj1.phyMarks = 90;
20          // second object
21          Marks obj2 = new Marks();
22          obj2.engMarks = 80;
23          obj2.mathsMarks = 60;
24          obj2.phyMarks = 85;
25          // displaying marks for first object
26          System.out.println("Marks for first object:");
27          System.out.println(obj1.engMarks);
28          System.out.println(obj1.mathsMarks);
29          System.out.println(obj1.phyMarks);
23          // displaying marks for second object
31          System.out.println("Marks for second object:");
32          System.out.println(obj2.engMarks);
33          System.out.println(obj2.mathsMarks);
34          System.out.println(obj2.phyMarks);
35      }
36  }

```

The screenshot shows the Notepad++ interface with two tabs: 'MarksDemo.java' and 'MarksDemo.java - Notepad++'. The code in 'MarksDemo.java' defines a class 'Marks' with three instance variables: 'engMarks', 'mathsMarks', and 'phyMarks'. The code in 'MarksDemo.java' creates two objects of the 'Marks' class and prints their respective marks. The output in the Command Prompt window shows the expected results.

```

C:\Users\Ravikanth\Desktop\OOPSJava\MarksDemo.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Wi
MarksDemo.java x
1 import java.io.*;
2 class Marks
3 {
4     // These variables are instance variables.
5     // These variables are in a class
6     // and are not inside any function
7     int engMarks;
8     int mathsMarks;
9     int phyMarks;
10 }

C:\Users\Ravikanth\Desktop\OOPSJava>javac MarksDemo.java
C:\Users\Ravikanth\Desktop\OOPSJava>java MarksDemo
Marks for first object:
50
80
90
Marks for second object:
80
60
85

```

## ► **Class/Static Variables**

- Static variables are also known as **Class variables**.
- These variables are **declared similarly as instance variables**, the **difference** is that **static variables are declared using the static keyword within a class outside any method constructor or block**.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are **created at the start of program execution** and **destroyed automatically when execution ends**.

## ► **Initialization of Static Variable is not Mandatory. Its default value is 0**

- If we access the static variable like Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name to class name automatically.
- If we access the static variable without the class name, Compiler will automatically append the class name.
- To access static variables, we need not create an object of that class, we can simply access the variable as **class\_name.variable\_name**;

## Example

The screenshot shows two windows: Notepad++ and a Command Prompt.

**Notepad++ Content:**

```
1 import java.io.*;
2 class Emp
3 {
4     // static variable salary
5     public static double salary;
6     public static String name = "EMP_Salary";
7 }
8 public class EmpDemo
9 {
10    public static void main(String args[])
11    {
12        // accessing static variable without object
13        Emp.salary = 1000;
14        System.out.println(Emp.name + "'s average salary:" + Emp.salary);
15    }
16 }
17 }
```

**Command Prompt Output:**

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac EmpDemo.java
C:\Users\Ravikanth\Desktop\OOPSJava>java EmpDemo
EMP_Salary's average salary:1000.0
```

## ► **Instance variable Vs Static variable**

- Each object **will have its own copy of instance variable** whereas We can only have **one copy of a static variable per class irrespective of how many objects we create.**
- Changes made in **an instance variable using one object will not be reflected in other objects** as each object has **its own copy of instance variable**. In case of static, **changes will be reflected in other objects** as static variables are common to all object of a class.
- We can **access instance variables through object references** and **Static Variables can be accessed directly using class name.**
- Syntax for static and instance variables:

### **class Example**

```
{  
    static int a; //static variable  
    int b;      //instance variable  
}
```

## ► Constants

- A constant is **an identifier** that is **similar to a variable** except that it holds the same value during its entire existence.
- As the name implies, it is constant, not variable.
- The compiler will issue an error if you try to change the value of a constant.
- In Java, we use the final modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

- Constants are **useful for three important reasons**
- **First**, they give meaning to otherwise unclear literal values
  - For example, MAX\_LOAD means more than the literal 250
- **Second**, they facilitate program maintenance
  - If a constant is used in multiple places, its value need only be updated in one place
- **Third**, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

### ► Scope and Lifetime of Variables:

- Scope of a variable refers to in which areas or sections of a program can the variable be accessed and lifetime of a variable refers to how long the variable stays alive in memory.
- General convention for a variable's scope is, it is accessible only within the block in which it is declared. A block begins with a left curly brace { and ends with a right curly brace }.
- As we know there are three types of variables:
  - 1) instance variables,
  - 2) class/static variables and
  - 3) local variables,
- we will look at the scope and lifetime of each of them now.
- General scope of an instance variable is throughout the class except in static methods. Lifetime of an instance variable is until the object stays in memory.
- General scope of a static/class variable is throughout the class and the lifetime of a static/class variable is until the end of the program or as long as the class is loaded in memory.
- Scope of a local variable is within the block in which it is declared and the lifetime of a local variable is until the control leaves the block in which it is declared.

| Variable Type     | Scope   | Lifetime   |
|-------------------|---|--|
| Instance variable | Throughout the class except in static methods | Until the object is available in the memory                |
| Class variable    | Throughout the class                          | Until the end of the program                               |
| Local variable    | Within the block in which it is declared      | Until the control leaves the block in which it is declared |

## ►Scope and Lifetime of Variables:

- Nested Scope
- In Java, we can create nested blocks – a block inside another block
- All the local variables in the outer block are accessible within the inner block but vice versa is not true i.e., local variables within the inner block are not accessible in the outer block. Consider the following example:

The screenshot shows a Java code editor and a terminal window. The code editor displays a file named 'Nested\_Variables.java' with the following content:

```
1  class Nested_Variables
2  {
3      public static void main(String[] args)
4      {
5          int x;
6          //Beginning of inner block
7          {
8              int y = 100;
9              x = 200;
10             System.out.println("x = "+x);
11         }
12         //End of inner block
13         System.out.println("x = "+x);
14         y = 200; //Error as y is not accessible in the outer block
15     }
16 }
```

The terminal window titled 'Command Prompt' shows the output of the compilation and execution of the code. It first shows the compilation error:

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Nested_Variables.java
Nested_Variables.java:14: error: cannot find symbol
                      y = 200; //Error as y is not accessible in the outer block
                                         ^
symbol:   variable y
location: class Nested_Variables
1 error
```

Then it shows the execution output:

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Nested_Variables
x = 200
x = 200
```

**► Trace the Program:**

```
class A{  
    int data=50;//instance variable  
    static int m=100;//static variable  
    void method(){  
        int n=90;//local variable  
    }  
}//end of class
```

```
class Simple{  
    public static void main(String[] args){  
        int a=10;  
        int b=10;  
        int c=a+b;  
        System.out.println(c);  
    }  
}
```

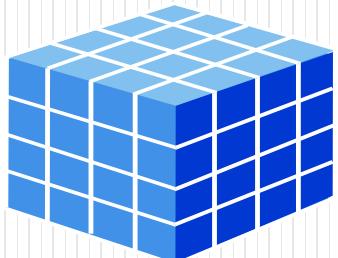


**For Details Contact Me @ :**  
**9247448766**  
**ravikanth.iiit@rgukt.ac.in**

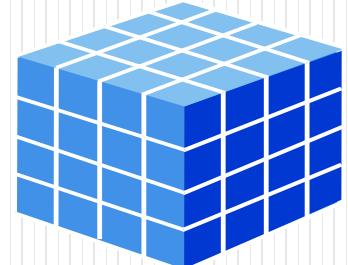
# OOOPS Using JAVA



## UNIT-II



Mr. K. RAVIKANTH  
Assistant Professor  
Department of CSE  
Rajiv Gandhi University of Knowledge Technologies, Basar.



- **Decision making and Branching:** Simple if statement, if, else statement, Nesting if, else, else if Ladder, switch statement,
- **Decision making and Looping:** While loop, do-While loop, for loop, break, labelled loop, continue Statement, Simple programs

## ► Decision making and Branching

- The program is a set of statements, which are stored into a file. The interpreter executes these statements in a sequential manner, i.e., in the order in which they appear in the file.
- But there are situations where programmers want to alter this normal sequential flow of control.
- For example, if one wants to repeatedly execute a block of statements or one wants to conditionally execute statements. Therefore' one more category of statements called control flow statements is provided.

| Statement | Type Keyword  |
|-----------|---|
| Selection | <code>if, if-else, nested if-else , else-if, switch-case</code> |
| Iteration | <code>while, do-while, for</code>                               |
| Jump      | <code>break, continue, label, return</code>                     |

## ► Decision making and Branching

- **if Statement** : The Java if statement tests the condition. It executes the *if block*, if condition is true.

- The *if statement* has the following syntax:

### Syntax:

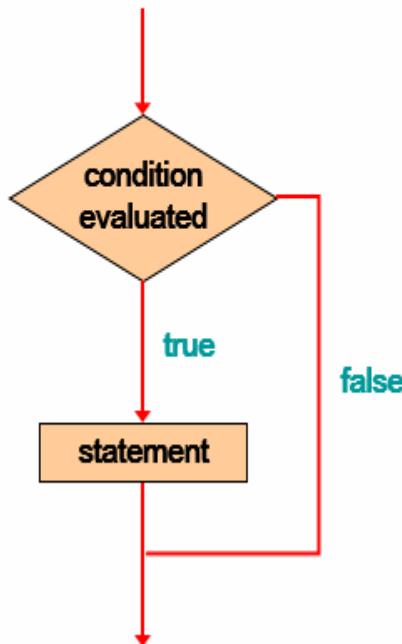
```
if (expression)
{
    // statements
}
```

- *if* is a Java reserved word

- The *expression* must be a boolean expression. It must evaluate to either true or false.

- If the *condition* is true, the *statement* is executed. If it is false, the *statement* is skipped.

### Logic of an if statement



## ► Decision making and Branching

### ► if Statement Example:

The screenshot shows a Notepad++ interface with the file `IfStatement.java` open. The code contains a simple if statement that prints "Number is positive." if the variable `number` is greater than 0, and "This statement is always executed." regardless of the value of `number`. The code is as follows:

```
1 class IfStatement
2 {
3     public static void main(String[] args)
4     {
5         int number = 10;
6         if (number > 0)
7         {
8             System.out.println("Number is positive.");
9         }
10        System.out.println("This statement is always executed.");
11    }
12 }
13
```

Below the editor is a Command Prompt window showing the execution of the Java code. The user navigates to the directory `C:\Users\Ravikanth\Desktop\OOPSJava`, compiles the Java file with `javac IfStatement.java`, and then runs it with `java IfStatement`. The output shows the expected results: "Number is positive." followed by "This statement is always executed."

```
C:\Users\Ravikanth>cd Desktop
C:\Users\Ravikanth\Desktop>cd OOPSJava
C:\Users\Ravikanth\Desktop\OOPSJava>javac IfStatement.java
C:\Users\Ravikanth\Desktop\OOPSJava>java IfStatement
Number is positive.
This statement is always executed.
```

## ► Decision making and Branching

### ► if Statement Example:

```
1 import java.util.Scanner;
2 class Age
3 {
4     public static void main (String[] args)
5     {
6         final int MINOR = 21;
7         Scanner scan = new Scanner (System.in);
8         System.out.print ("Enter your age: ");
9         int age = scan.nextInt();
10        System.out.println ("You entered: " + age);
11        if (age < MINOR)
12            System.out.println ("Youth is a wonderful thing. "+ "Enjoy.");
13        System.out.println ("Age is a state of mind.");
14    }
15 }
16 }
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Age.java
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Age
Enter your age: 19
You entered: 19
Youth is a wonderful thing. Enjoy.
Age is a state of mind.
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Age
Enter your age: 32
You entered: 32
Age is a state of mind.
```

## ► Decision making and Branching

### ► if...else Statement

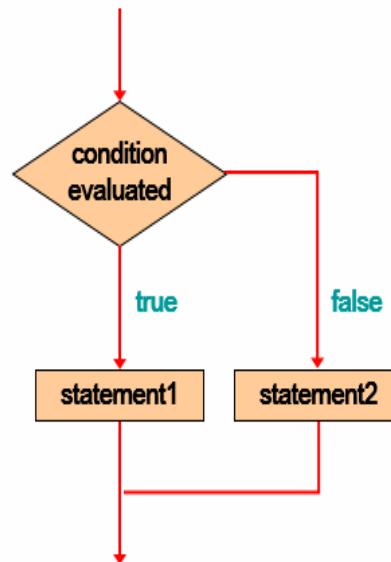
The if statement executes a certain section of code if the test expression is evaluated to true. The if statement can have optional else statement. Codes inside the body of else statement are executed if the test expression is false.

#### Syntax:

```
if (expression)
{
    // codes
}
else
{
    // some other code
}
```

- If the *expression* is true, *statement1* is executed; if the condition is false, *statement2* is executed
- One or the other will be executed, but not both

#### Logic of an if-else statement



## ► Decision making and Branching

### ► if...else Statement Example

```

1  class IfElse
2  {
3      public static void main(String[] args)
4      {
5          int number = 10;
6          if (number > 0)
7          {
8              System.out.println("Number is positive.");
9          }
10         else
11         {
12             System.out.println("Number is not positive.");
13         }
14         System.out.println("This statement is always executed.");
15     }
16 }

```

Command Prompt

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac IfElse.java

C:\Users\Ravikanth\Desktop\OOPSJava>java IfElse
Number is positive.
This statement is always executed.

```

```

1  import java.util.*;
2  class IfelseEx
3  {
4      public static void main(String args[])
5      {
6          int num1, num2, num3, min = 0;
7          Scanner scan = new Scanner (System.in);
8          System.out.println ("Enter three integers: ");
9          num1 = scan.nextInt();
10         num2 = scan.nextInt();
11         num3 = scan.nextInt();
12         if (num1 < num2)
13             if (num1 < num3)
14                 min = num1;
15             else
16                 min = num3;
17         else
18             if (num2 < num3)
19                 min = num2;
20             else
21                 min = num3;
22         System.out.println ("Minimum value: " + min);
23     }
24 }

```

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac IfelseEx.java

C:\Users\Ravikanth\Desktop\OOPSJava>java IfelseEx
Enter three integers:
10
20
30
Minimum value: 10

```

## ► Decision making and Branching

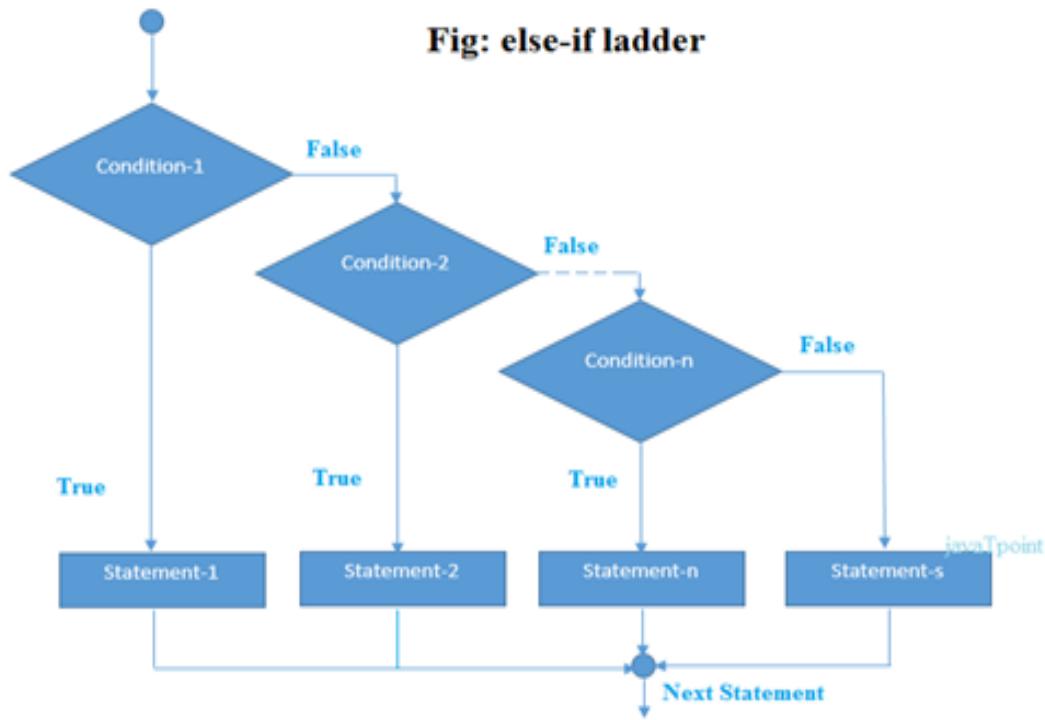
### ► else-if ladder Statement

► The if-else-if ladder statement executes one condition from multiple statements.

#### Syntax:

```
if (expression1)
{
    // codes
}
else if(expression2)
{
    // codes
}
else if (expression3)
{
    // codes
}
.....
else
{
    // codes
}
```

#### Flowchart:



- The if statements are executed from the top towards the bottom. As soon as the test expression is true, codes inside the body of that if statement is executed.
- Then, the control of program jumps outside if-else-if ladder. If all test expressions are false, codes inside the body of else is executed.

## ► Decision making and Branching

### ► else-if ladder Statement

The screenshot shows the Notepad++ interface with a file named "Ladder.java". The code implements an else-if ladder to determine the sign of a number:

```
1 class Ladder
2 {
3     public static void main(String[] args)
4     {
5         int number = 0;
6         if (number > 0)
7         {
8             System.out.println("Number is positive.");
9         }
10        else if (number < 0)
11        {
12            System.out.println("Number is negative.");
13        }
14        else
15        {
16            System.out.println("Number is 0.");
17        }
18    }
19 }
20 }
```

The screenshot shows the Windows Command Prompt window. It first runs the compilation command `javac Ladder.java` and then executes the program with `java Ladder`. The output shows that the number is 0.

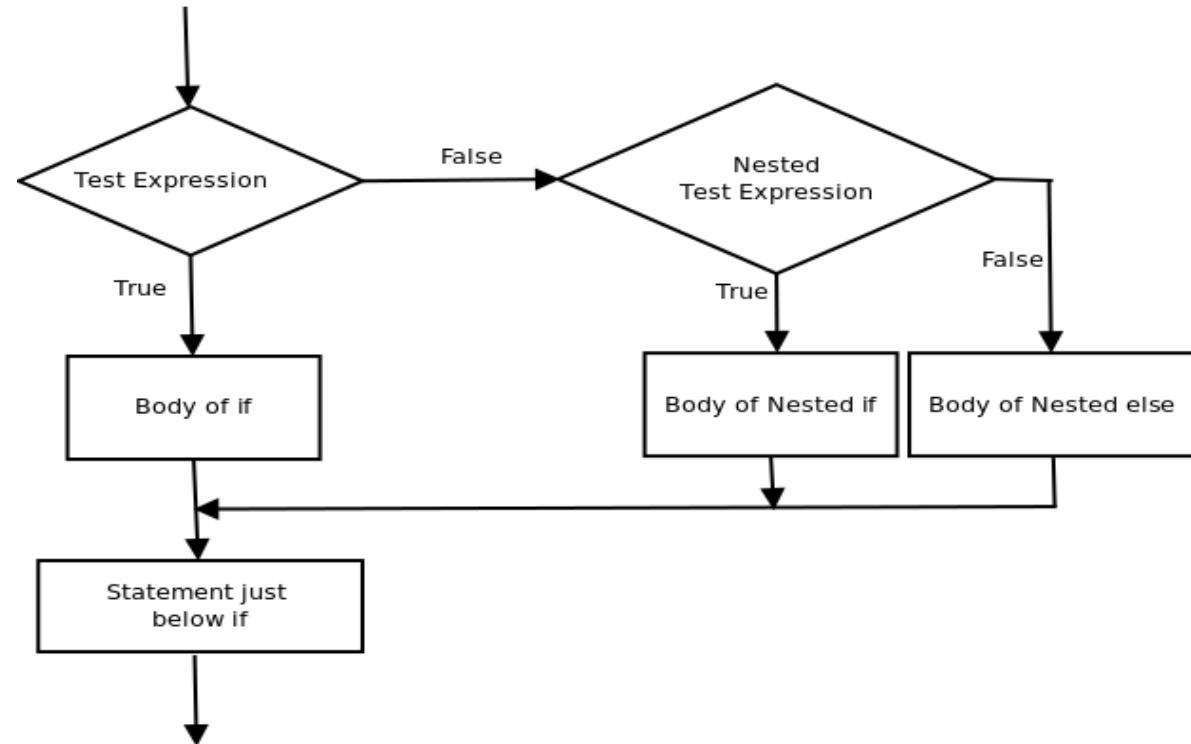
```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Ladder.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Ladder
Number is 0.
```

## ► Decision making and Branching

### ► Nested if...else:

► A nested if is an if statement that is the target of another if statement.  
Nested if statements means an if statement inside another if statement

Flow Chart:



## ► Decision making and Branching

```
1  class Number
2  {
3      public static void main(String[] args)
4      {
5          Double n1 = -1.0, n2 = 4.5, n3 = -5.3, largestNumber;
6          if (n1 >= n2)
7          {
8              if (n1 >= n3)
9              {
10                  largestNumber = n1;
11              }
12              else
13              {
14                  largestNumber = n3;
15              }
16          }
17          else
18          {
19              if (n2 >= n3)
20              {
21                  largestNumber = n2;
22              }
23              else
24              {
25                  largestNumber = n3;
26              }
27          }
28          System.out.println("Largest number is " + largestNumber);
29      }
30  }
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Number.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Number
Largest number is 4.5
```

► **Decision making and Branching**

► **Switch Statement:**

- The Java switch statement executes one statement from multiple conditions.  
It is like if-else-if ladder statement.
- The *switch statement* provides another way to decide which statement to execute next
- The **switch** statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement associated with the first case value that matches

## ► Decision making and Branching

### ► Switch Statement:

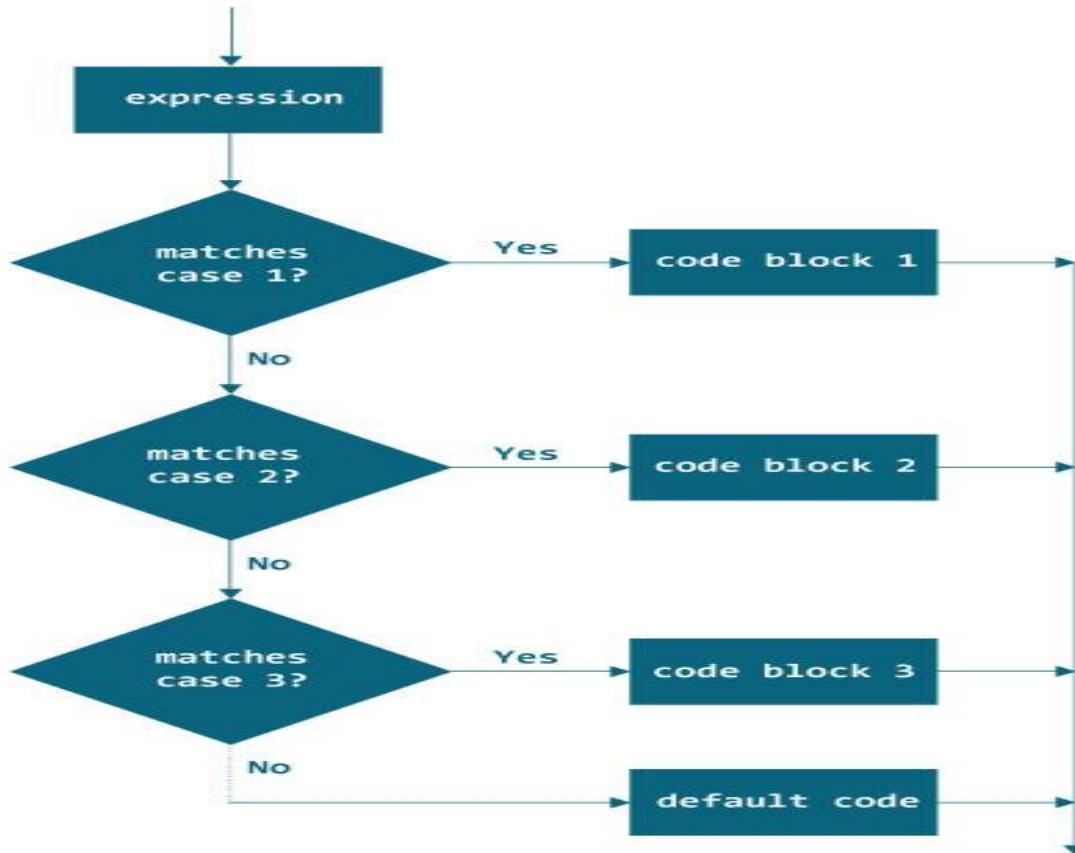
Syntax of a switch statement is:

```
switch      switch ( expression )
and          {
case         case value1 :
are          statement-list1
reserved    case value2 :
words       statement-list2
           case value3 :
           statement-list3
           case ...
}

```

If expression matches value2, control jumps to here

**Flowchart**



## ► Control Flow & Jump Statements

### ► switch statement

- A switch statement can have an optional default case
- The default case has no associated value and simply uses the reserved word default
- If the default case is present, control will transfer to it if no other case value matches
- If there is no default case, and no other value matches, control falls through to the statement after the switch
- The expression of a switch statement must result in an integral type, meaning an int or a char
- It cannot be a boolean value, a floating point value (float or double), or another integer type
- The implicit boolean condition in a switch statement is equality
- You cannot perform relational checks with a switch statement

C:\Users\Ravikanth\Desktop\OOPSJava\Day.java - Notepad++

File Edit Search View Encoding Language Settings Tools Ma

Day.java X

```
1 class Day
2 {
3     public static void main(String[] args)
4     {
5         int week = 7;
6         String day;
7         switch (week)
8         {
9             case 1:
10                day = "Sunday";
11                break;
12            case 2:
13                day = "Monday";
14                break;
15            case 3:
16                day = "Tuesday";
17                break;
18            case 4:
19                day = "Wednesday";
20                break;
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36 }
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Day.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Day
Saturday
C:\Users\Ravikanth\Desktop\OOPSJava>
```

```
case 5:
    day = "Thursday";
    break;
case 6:
    day = "Friday";
    break;
case 7:
    day = "Saturday";
    break;
default:
    day = "Invalid day";
    break;
}
System.out.println(day);
```

## Example :

The program below takes three inputs from the user: operator and 2 numbers. It performs calculation based on numbers and operator entered. Then the result is displayed on the screen. We have used Scanner object to take input from the user.

The screenshot shows the Java code for a calculator program in an IDE. The code uses a Scanner object to read input from the user. It prompts for an operator (+, -, \*, /) and two numbers. Based on the operator, it performs the corresponding arithmetic operation and prints the result. If an invalid operator is entered, it prints an error message. The code is annotated with line numbers 1 through 39. To the right, a Command Prompt window shows the execution of the program, including the compilation of the code and several runs of the application with different inputs and outputs.

```
Calculator.java
1 import java.util.Scanner;
2 class Calculator
3 {
4     public static void main(String[] args)
5     {
6         char operator;
7         int number1, number2;
8         int result=0;
9         float div;
10        Scanner scanner = new Scanner(System.in);
11        System.out.print("Enter operator (either +, -, * or /): ");
12        operator = scanner.next().charAt(0);
13        System.out.print("Enter number1 and number2 respectively: ");
14        number1 = scanner.nextInt();
15        number2 = scanner.nextInt();
16        switch (operator)
17        {
18            case '+':
19                result = number1 + number2;
20                System.out.print(number1 + "+" + number2 + " = " + result);
21                break;
22            case '-':
23                result = number1 - number2;
24                System.out.print(number1 + "-" + number2 + " = " + result);
25                break;
26            case '*':
27                result = number1 * number2;
28                System.out.print(number1 + "*" + number2 + " = " + result);
29                break;
30            case '/':
31                div =(float) number1 / number2;
32                System.out.print(number1 + "/" + number2 + " = " + result);
33                break;
34            default:
35                System.out.println("Invalid operator!");
36                break;
37        }
38    }
39}
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac Calculator.java
C:\Users\Ravikanth\Desktop\OOPSJava>java Calculator
Enter operator (either +, -, * or /):
+
Enter number1 and number2 respectively:
10
20
10+20 = 30
C:\Users\Ravikanth\Desktop\OOPSJava>java Calculator
Enter operator (either +, -, * or /):
*
Enter number1 and number2 respectively:
15
5
15*5 = 75
```

**► Decision making and Looping****► Loops in Java:**

► Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

They are:

**► while loop****► for loop****► do-while loop**

## ► Decision making and Looping

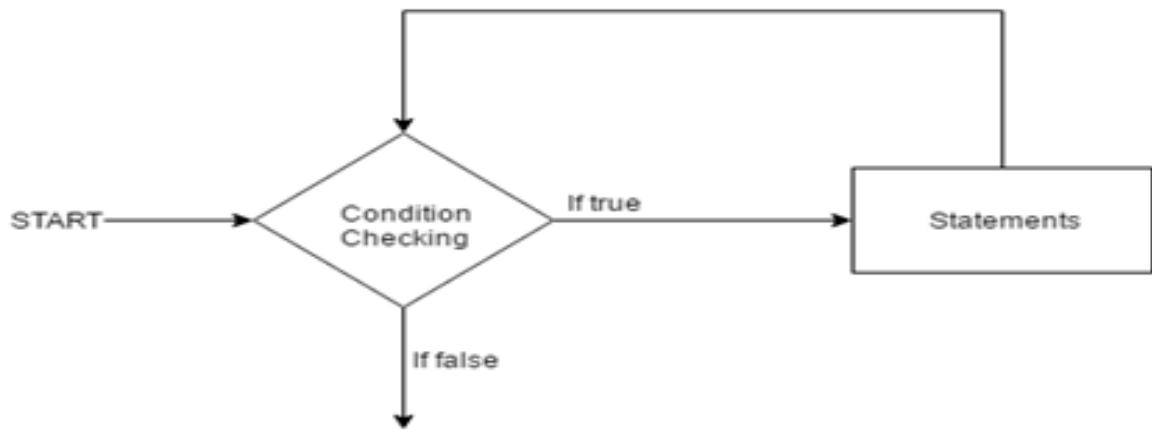
### ► while loop:

- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

#### Syntax:

```
while (boolean condition)
{
    loop statements...
}
```

#### Flowchart:



- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**.
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

## ► Decision making and Looping

### ► Example over While Loop

```
1 // Java program to illustrate while loop
2 class whileLoopDemo
3 {
4     public static void main(String args[])
5     {
6         int x = 1;
7         // Exit when x becomes greater than 4
8         while (x <= 4)
9         {
10            System.out.println("Value of x:" + x);
11
12            //increment the value of x for next iteration
13            x++;
14        }
15    }
16 }
17 }
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac whileLoopDemo.java
C:\Users\Ravikanth\Desktop\OOPSJava>java whileLoopDemo
Value of x:1
Value of x:2
Value of x:3
Value of x:4
```

## ► Decision making and Looping

### ► for Loop

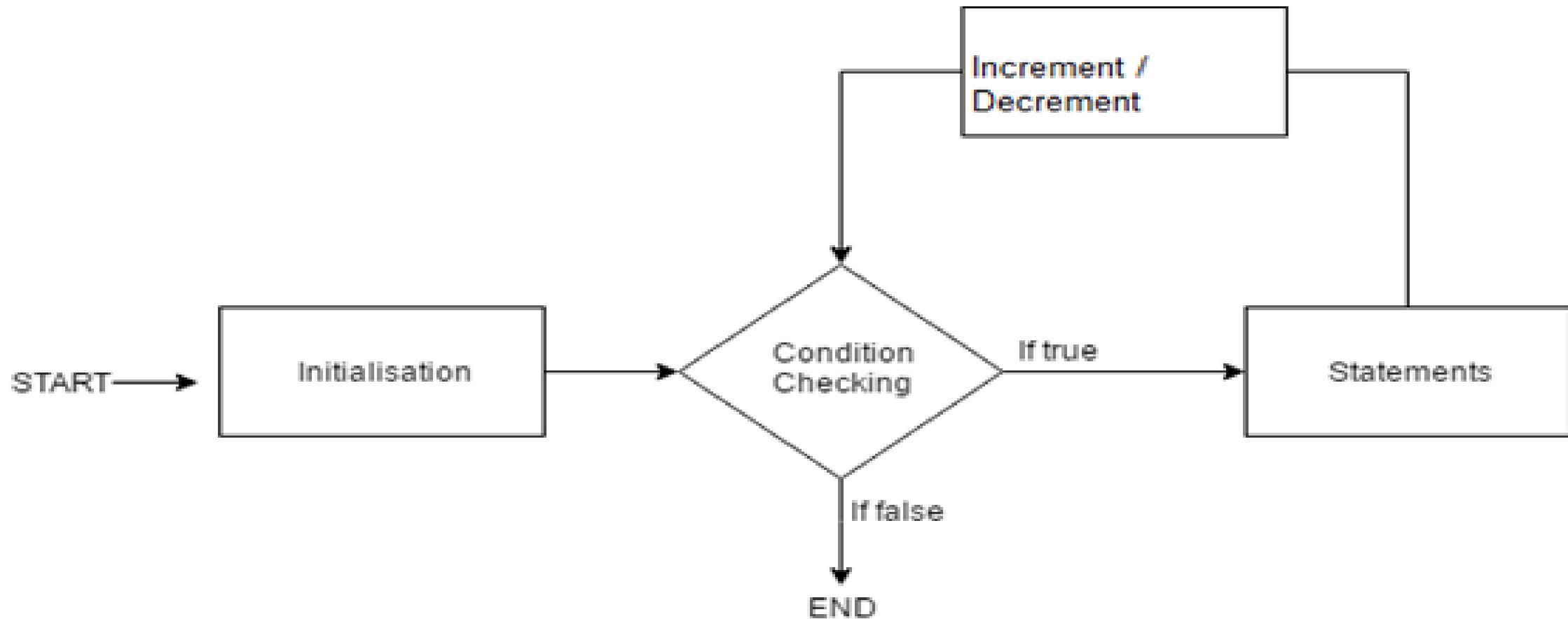
- for loop provides a concise way of writing the loop structure.
- Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

#### Syntax:

```
for (initialization condition; testing condition; increment/decrement)  
{  
    statement(s)  
}
```

**► Decision making and Looping****► for loop syntax explanation**

- **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Testing Condition:** It is used for testing the exit condition for a loop. It must return a Boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

**► Decision making and Looping****► for loop****Flowchart:**

## ► Decision making and Looping

### ► for loop

The screenshot shows a Java code editor with a file named `forLoopDemo.java`. The code demonstrates a for loop that prints the values of `x` from 2 to 4. To the right, a command prompt window shows the execution of the code, including the compilation step (`javac forLoopDemo.java`) and the execution step (`java forLoopDemo`), which outputs the values of `x`.

```
// Java program to illustrate for loop.  
class forLoopDemo  
{  
    public static void main(String args[])  
    {  
        // for loop begins when x=2  
        // and runs till x <=4  
        for (int x = 2; x <= 4; x++)  
            System.out.println("Value of x:" + x);  
    }  
}
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac forLoopDemo.java  
C:\Users\Ravikanth\Desktop\OOPSJava>java forLoopDemo  
Value of x:2  
Value of x:3  
Value of x:4
```

**► Decision making and Looping****► Enhanced for Loop**

- As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.
- Enhanced for loop provides a simpler way to iterate through the elements of a collection or array.
- It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.

- Decision making and Looping
- Enhanced for Loop Syntax:

### Syntax

Following is the syntax of enhanced for loop –

```
for(declaration : expression)
{
    // Statements
}
```



- **Declaration** – The newly declared block variable, is of a type compatible with the elements of the array you are accessing.
- The variable will be available within the for block and its value would be the same as the current array element.
- **Expression** – This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

## ► Enhanced for Loop Example:

- Suppose there is an array of names and we want to print all the names in that array.
- Let's see the difference with these two examples
- Enhanced for loop simplifies the work as follows-
- **Example:** Java program to illustrate enhanced for loop

```

1  public class enhancedforloop
2  {
3      public static void main(String args[])
4      {
5          String array[] = {"IIIT", "RGUKT", "Basar"};
6          //enhanced for loop
7          for (String x:array)
8          {
9              System.out.println(x);
10         }
11         /* for loop for same function
12         for (int i = 0; i < array.length; i++)
13         {
14             System.out.println(array[i]);
15         }
16     }
17 }
```

```

C:\Users\Ravikanth\Desktop\OOPSJava>javac enhancedforloop.java
C:\Users\Ravikanth\Desktop\OOPSJava>java enhancedforloop
IIIT
RGUKT
Basar
```

```

1  public class enhancedforloop
2  {
3      public static void main(String args[])
4      {
5          String array[] = {"IIIT", "RGUKT", "Basar"};
6          //enhanced for loop
7          /*for (String x:array)
8          {
9              System.out.println(x);
10         }*/
11         //for loop for same function
12         for (int i = 0; i < array.length; i++)
13         {
14             System.out.println(array[i]);
15         }
16     }
17 }
```

## ► Enhanced for Loop Example:

The screenshot shows the Notepad++ interface with the file `En_For_each.java` open. The code demonstrates the use of enhanced for loops to iterate over arrays of integers and strings.

```
1 public class En_For_each
2 {
3     public static void main(String args[])
4     {
5         int [] numbers = {10, 20, 30, 40, 50};
6         for(int x : numbers )
7         {
8             System.out.print( x );
9             System.out.print(",");
10        }
11        System.out.print("\n");
12        String [] names = {"Ravikanth", "Sreekanth", "Chandu", "Dheeraj"};
13        for( String name : names )
14        {
15            System.out.print( name );
16            System.out.print(",");
17        }
18    }
19 }
```

A command prompt window is shown running the command `javac En_For_each.java`, followed by the output of the program running `java En_For_each`.

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac En_For_each.java
C:\Users\Ravikanth\Desktop\OOPSJava>java En_For_each
10,20,30,40,50,
Ravikanth,Sreekanth,Chandu,Dheeraj,
```

## ► Decision making and Looping

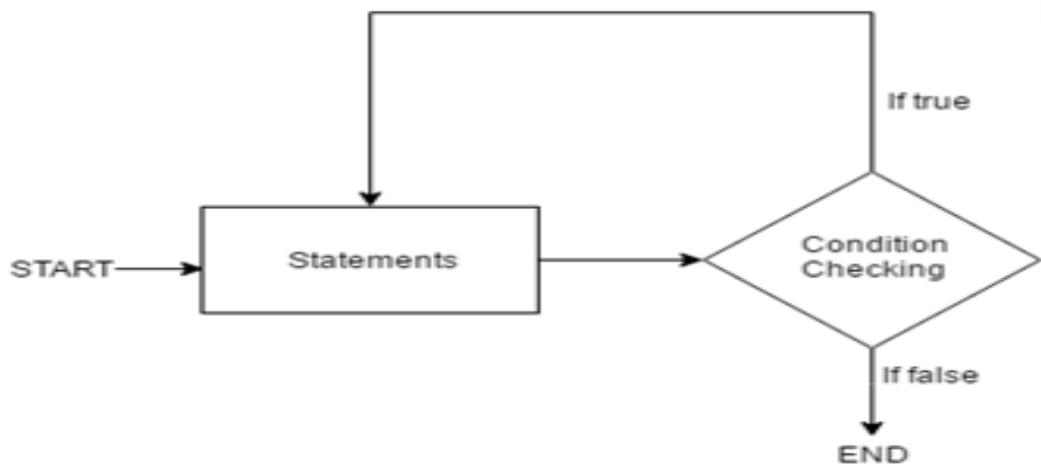
### ► do while:

► do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop**.

#### Syntax:

```
do
{
    statements...
}
while (condition);
```

#### Flowchart:



- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

## ► Decision making and Looping

### ► do-while loop Example

```
dowhileloopDemo.java X |  
1  class dowhileloopDemo  
2 {  
3     public static void main(String args[])  
4     {  
5         int x = 21;  
6         do  
7         {  
8             System.out.println("Value of x:" + x);  
9             x++;  
10        }  
11        while (x < 20);  
12    }  
13 }
```

```
Command Prompt  
C:\Users\Ravikanth\Desktop\OOPSJava>javac dowhileloopDemo.java  
C:\Users\Ravikanth\Desktop\OOPSJava>java dowhileloopDemo  
Value of x:21
```

## ► Java supports two Branching statements : break and continue

### ► break Statement:

- The break statement is used to terminate the execution of the nearest enclosing loop in which it appears.
- The break statement is widely used with for loop and while loop. When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the inner most loop.

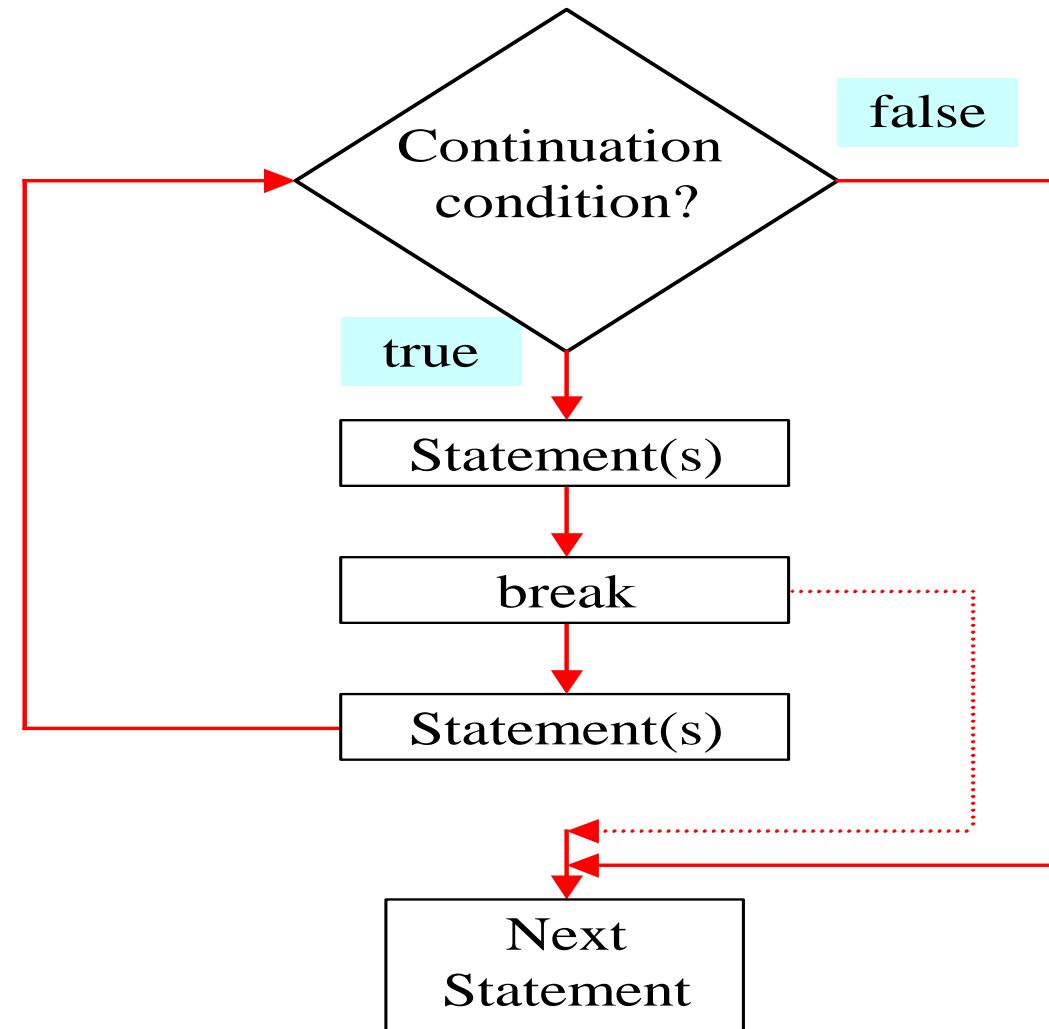
### Syntax:

```
break;
```

► Java supports two Branching statements : break and continue

► break Statement Syntax:

```
while ( i < 5 )  
{  
    //do something;  
    if (i <0)  
        break; //jumps out of the loop  
}
```



- Java supports two Branching statements : break and continue
- break Statement Example:

```
1 public class BreakDemo
2 {
3     public static void main(String[] args)
4     {
5         for (int i = 1; i <= 10; i++)
6         {
7             if (i == 5)
8             {
9                 break;      // terminate loop if i is 5
10            }
11            System.out.print(i + " ");
12        }
13        System.out.println("Loop is over.");
14    }
15 }
```

Command Prompt

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac BreakDemo.java
C:\Users\Ravikanth\Desktop\OOPSJava>java BreakDemo
1 2 3 4 Loop is over.
```

► Java supports two Branching statements : break and continue

► continue Statement:

► Like the break statement, the continue statement can only appear in the body of a loop.

► When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

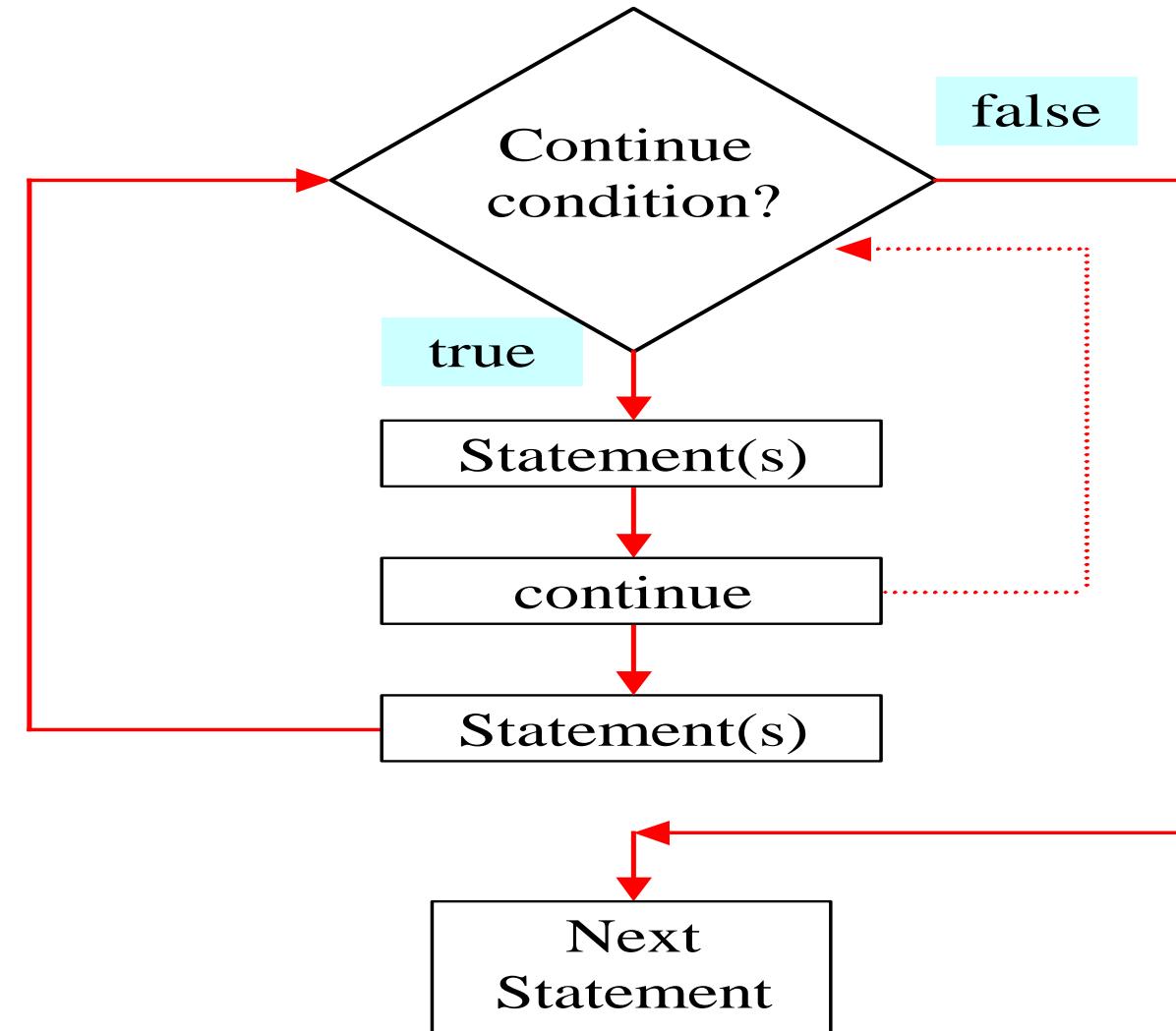
**Syntax:**

**continue;**

► Java supports two Branching statements : break and continue

► continue Statement Syntax:

```
while(i<5){  
    //do something1;  
    if (i<4)  
        continue;  
    //do something2;  
}
```



- Java supports two Branching statements : break and continue
- continue Statement Example:

The screenshot shows a Java code editor with the file `ContinueDemo.java`. The code contains a `for` loop that prints odd numbers from 1 to 9. It includes a `continue` statement to skip even numbers. Below the editor is a terminal window showing the command-line interface for compilation and execution.

```
1 public class ContinueDemo
2 {
3     public static void main(String[] args)
4     {
5         for (int i = 1; i <= 10; i++)
6         {
7             if (i % 2 == 0)
8             {
9                 continue; // skip next statement if i is even
10            }
11            System.out.println(i + " ");
12        }
13    }
14 }
```

```
C:\Users\Ravikanth\Desktop\OOPSJava>javac ContinueDemo.java
C:\Users\Ravikanth\Desktop\OOPSJava>java ContinueDemo
1
3
5
7
9
```

## ► Trace the Program

```
//Java Program to demonstate the use of if statement.  
public class IfExample {  
    public static void main(String[] args) {  
        //defining an 'age' variable  
        int age=20;  
        //checking the age  
        if(age>18){  
            System.out.print("Age is greater than 18");  
        }  
    }  
}
```

## ► Trace the Program

```
//A Java Program to demonstrate the use of if-else statement.  
//It is a program of odd and even number.  
public class IfElseExample {  
public static void main(String[] args) {  
    //defining a variable  
    int number=13;  
    //Check if the number is divisible by 2 or not  
    if(number%2==0){  
        System.out.println("even number");  
    }else{  
        System.out.println("odd number");  
    }  
}
```

## ► Trace the Program

### Leap Year Example:

A year is leap, if it is divisible by 4 and 400. But, not by 100.

```
public class LeapYearExample {  
    public static void main(String[] args) {  
        int year=2020;  
        if(((year % 4 ==0) && (year % 100 !=0)) || (year % 400==0)){  
            System.out.println("LEAP YEAR");  
        }  
        else{  
            System.out.println("COMMON YEAR");  
        }  
    }  
}
```

## ► Trace the Program

```
public class IfElseTernaryExample {  
    public static void main(String[] args) {  
        int number=13;  
        //Using ternary operator  
        String output=(number%2==0)?"even number":"odd number";  
        System.out.println(output);  
    }  
}
```

## ► Trace the Program

```
//Java Program to demonstrate the use of If else-if ladder.  
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.  
public class IfElseIfExample {  
    public static void main(String[] args) {  
        int marks=65;  
  
        if(marks<50){  
            System.out.println("fail");  
        }  
        else if(marks>=50 && marks<60){  
            System.out.println("D grade");  
        }  
        else if(marks>=60 && marks<70){  
            System.out.println("C grade");  
        }  
        else if(marks>=70 && marks<80){  
            System.out.println("B grade");  
        }  
        else if(marks>=80 && marks<90){  
            System.out.println("A grade");  
        }else if(marks>=90 && marks<100){  
            System.out.println("A+ grade");  
        }else{  
            System.out.println("Invalid!");  
        }  
    }  
}
```

## ► Trace the Program

```
public class PositiveNegativeExample {  
    public static void main(String[] args) {  
        int number=-13;  
        if(number>0){  
            System.out.println("POSITIVE");  
        }else if(number<0){  
            System.out.println("NEGATIVE");  
        }else{  
            System.out.println("ZERO");  
        }  
    }  
}
```

## ► Trace the Program

```
//Java Program to demonstrate the use of Nested If Statement.  
  
public class JavaNestedIfExample {  
    public static void main(String[] args) {  
        //Creating two variables for age and weight  
        int age=20;  
        int weight=80;  
        //applying condition on age and weight  
        if(age>=18){  
            if(weight>50){  
                System.out.println("You are eligible to donate blood");  
            }  
        }  
    }  
}
```

## ► Trace the Program

```
public class SwitchExample {  
    public static void main(String[] args) {  
        //Declaring a variable for switch expression  
        int number=20;  
        //Switch expression  
        switch(number){  
            //Case statements  
            case 10: System.out.println("10");  
            break;  
            case 20: System.out.println("20");  
            break;  
            case 30: System.out.println("30");  
            break;  
            //Default case statement  
            default: System.out.println("Not in 10, 20 or 30");  
        }  
    }  
}
```

## Trace the Program

```
//Java Program to demonstrate the example of Switch statement  
//where we are printing month name for the given number  
public class SwitchMonthExample {  
    public static void main(String[] args) {  
        //Specifying month number  
        int month=7;  
        String monthString="";  
        //Switch statement  
        switch(month){  
            //case statements within the switch block  
            case 1: monthString="1 - January";  
            break;  
            case 2: monthString="2 - February";  
            break;  
            case 3: monthString="3 - March";  
            break;  
            case 4: monthString="4 - April";  
            break;  
            case 5: monthString="5 - May";  
            break;  
            case 6: monthString="6 - June";  
            break;  
            case 7: monthString="7 - July";  
            break;  
            case 8: monthString="8 - August";  
            break;  
            case 9: monthString="9 - September";  
            break;  
            case 10: monthString="10 - October";  
            break;  
            case 11: monthString="11 - November";  
            break;  
            case 12: monthString="12 - December";  
            break;  
            default:System.out.println("Invalid Month!");  
        }  
        //Printing month of the given number  
        System.out.println(monthString);  
    }  
}
```

## ► Trace the Program

```
//Java Program to demonstrate the example of for loop  
//which prints table of 1  
public class ForExample {  
public static void main(String[] args) {  
    //Code of Java for loop  
    for(int i=1;i<=10;i++){  
        System.out.println(i);  
    }  
}  
}
```

## ► Trace the Program

```
public class NestedForExample {  
    public static void main(String[] args) {  
        //loop of i  
        for(int i=1;i<=3;i++){  
            //loop of j  
            for(int j=1;j<=3;j++){  
                System.out.println(i+" "+j);  
            } //end of i  
        } //end of j  
    }  
}
```

## ► Trace the Program

```
public class PyramidExample {  
    public static void main(String[] args) {  
        for(int i=1;i<=5;i++){  
            for(int j=1;j<=i;j++){  
                System.out.print("* ");  
            }  
            System.out.println();//new line  
        }  
    }  
}
```

## ► Trace the Program

```
public class WhileExample {  
    public static void main(String[] args) {  
        int i=1;  
        while(i<=10){  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

## ► Trace the Program

```
public class DoWhileExample {  
    public static void main(String[] args) {  
        int i=1;  
        do{  
            System.out.println(i);  
            i++;  
        }while(i<=10);  
    }  
}
```

## ► Trace the Program

```
//Java Program to demonstrate the use of break statement  
//inside the while loop.  
public class BreakWhileExample {  
    public static void main(String[] args) {  
        //while loop  
        int i=1;  
        while(i<=10){  
            if(i==5){  
                //using break statement  
                i++;  
                break;//it will break the loop  
            }  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

```
//Java Program to demonstrate the use of continue statement  
//inside the for loop.  
public class ContinueExample {  
    public static void main(String[] args) {  
        //for loop  
        for(int i=1;i<=10;i++){  
            if(i==5){  
                //using continue statement  
                continue;//it will skip the rest statement  
            }  
            System.out.println(i);  
        }  
    }  
}
```

**► Its Time to Code:****1. Program to check whether the given number is positive or negative**

► In this program we have specified the value of number during declaration and the program checks whether the specified number is positive or negative.

**2. Check whether the input number(entered by user) is positive or negative**

► Here we are using Scanner to read the number entered by user and then the program checks and displays the result.

**3. Sum of two numbers using Scanner**

► The scanner allows us to capture the user input so that we can get the values of both the numbers from user. The program then calculates the sum and displays it.

**4. calculate Student Grade and Category based on his/her performance**

► Use Scanner class and Switch cases to display the Grade



**For Details Contact Me @ :**  
**9247448766**  
**ravikanth.iiit@rgukt.ac.in**

## UNIT- II

### **Topics to be covered:**

#### **Arrays:**

One Dimensional Array, Creating an array, Array processing, Multidimensional Array,

#### Arrays:

- An Array is an Indexed Collection of fixed number of homogeneous data elements
- An *array* is a collection of variables of the same type, referred to by a common name. An *array* is a collection of variables of the same type, referred to by a common name
- The main advantage of array is we can represent multiple values under the same name. So, that readability of the code is improved
- But the main limitation of array is once we created an array there is no chance of increasing /decreasing size based on our requirement. Hence memory point of view arrays concept is not recommended
- We can resolve this problem using Collections

In Java, arrays can have one or more dimensions, although the one-dimensional array is the most common. Arrays are used for a variety of purposes because they offer a convenient means of grouping together related variables.

For example, you might use an array to hold a record of the daily high temperature for a month, a list of stock price averages, or a list of your collection of programming books.

The principal advantage of an array is that it organizes data in such a way that it can be easily manipulated.

For example, if you have an array containing the incomes for a selected group of households, it is easy to compute the average income by cycling through the array. In Java Arrays are implemented as objects.

#### **One-Dimensional Arrays**

A one-dimensional array is a list of related variables. For example, you might use a one-dimensional array to store the account numbers of the active users on a network. Another array might be used to store the current batting averages for a cricket team.

#### **Declaration of one Dimensional Arrays:**

*Syntax: type array-name[ ] = new type[size];*

1. Here, *type* declares the element type of the array.
2. The element type determines the data type of each element contained in the array.

3. The number of elements that the array will hold is determined by *size*.
4. Since arrays are implemented as objects, the creation of an array is a two-step process.  
First, you declare an array reference variable.  
Second, you allocate memory for the array, assigning a reference to that memory to the array variable.
5. Thus, arrays in Java are dynamically allocated using the **new** operator.

Here is an example.

```
int marks[ ] = new int[10];
```

The following creates an **int** array of 10 elements and links it to an array reference variable named **marks**:

This declaration works just like an object declaration. The **marks** variable holds a reference to the memory allocated by **new**. This memory is large enough to hold 10 elements of type **int**. As with objects, it is possible to break the preceding declaration in two.

For example:

```
int sample[ ];  
sample = new int[10];
```

In this case, when **sample** is first created, it refers to no physical object. It is only after the second statement executes that **sample** is linked with an array.

An individual element within an array is accessed by use of an index. An *index* describes the position of an element within an array. In Java, all arrays have zero as the index of their first element. Because **sample** has 10 elements, it has index values of 0 through 9. To index an array, specify the number of the element you want, surrounded by square brackets. Thus, the first element in **sample** is **sample[0]**, and the last element is **sample[9]**.

### Example:-

```
public class Arrays1
{
    public static void main(String args[])
    {
        int sample[ ] = new int[10];
        int i;
        for(i = 0; i < 10; i = i+1)
            sample[i] = i;
        for(i = 0; i < 10; i = i+1)
            System.out.println("This is sample[" + i + "]: " + sample[i]);
    }
}
```

### **Output:**

The output from the program is shown here:

```
This is sample[0]: 0
This is sample[1]: 1
This is sample[2]: 2
This is sample[3]: 3
This is sample[4]: 4
This is sample[5]: 5
This is sample[6]: 6
This is sample[7]: 7
This is sample[8]: 8
This is sample[9]: 9
```

Conceptually, the **sample** array looks like this:

|            |            |            |            |            |            |            |            |            |            |
|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| 0          | 1          | 2          | 3          | 4          | 5          | 6          | 7          | 8          | 9          |
| Sample [0] | Sample [1] | Sample [2] | Sample [3] | Sample [4] | Sample [5] | Sample [6] | Sample [7] | Sample [8] | Sample [9] |

### Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

```
// accessing the elements of the specified array
for (int i = 0; i < arr.length; i++)
    System.out.println("Element at index " + i + " : " + arr[i]);
```

### **Example: Java Program to find Min and Max element in user given array.**

```
import java.util.Scanner;
class MinMax
{
    public static void main(String args[ ])
    {
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter Size of the Array: ");
        int len=scan.nextInt();
        int nums[ ] = new int[len];
        int min,max;
        for(int i=0;i<len;i++)
```

```

{
    System.out.print("Enter "+i+"th element: ");
    nums[i]=scan.nextInt();
}
min=max=nums[0];
for(int i=0;i<len;i++)
{
    if(min>nums[i])
        min=nums[i];
    else
        max=nums[i];
}
System.out.println("Min element is: "+min+"\nMax element is:
"+max);
}
}

```

### Multidimensional arrays

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array. A multidimensional array is created by appending one set of square brackets ([ ]) per dimension.

#### Examples:

```

int[][] int_Array = new int[10][20]; //a 2D array or matrix
int[][][] int_Array = new int[10][20][10]; //a 3D array

```

#### **For example,**

```
int[ ][ ] a = new int[3][4];
```

Here, a is a two-dimensional (2d) array.

The array can hold maximum of 12 elements of type int.

|       | Column 1 | Column 2 | Column 3 | Column 4 |
|-------|----------|----------|----------|----------|
| Row 1 | a[0][0]  | a[0][1]  | a[0][2]  | a[0][3]  |
| Row 2 | a[1][0]  | a[1][1]  | a[1][2]  | a[1][3]  |
| Row 3 | a[2][0]  | a[2][1]  | a[2][2]  | a[2][3]  |

Similarly, you can declare a three-dimensional (3d) array. For example,

```
String[ ][ ][ ] personalInfo = new String[3][4][2];
```

Here, *personalInfo* is a 3d array that can hold maximum of 24 (3\*4\*2) elements of type String.

```
class MultidimensionalArray
{
    public static void main(String[] args)
    {
        int[][][] a = { {1, 2, 3}, {4, 5, 6, 9}, {7} };
        System.out.println("Length of row 1: " + a[0].length);
        System.out.println("Length of row 2: " + a[1].length);
        System.out.println("Length of row 3: " + a[2].length);
    }
}
```

When you run the program, the output will be:

Length of row 1: 3

Length of row 2: 4

Length of row 3: 1

Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays), you can use length attribute to find the length of each rows.

**Example: Print all elements of 2d array Using Loop**

```
class MultidimensionalArray
{
    public static void main(String[ ] args)
    {
        int[][][] a = {{ {1, -2, 3}, {-4, -5, 6, 9}, {7} }};
        for (int i = 0; i < a.length; ++i)
        {
            for(int j = 0; j < a[i].length; ++j)
            {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

**Example: Multiply two Matrices**

```
import java.util.Scanner;
public class JavaProgram
{
    public static void main(String args[])
    {
        int m, n, p, q, sum = 0, c, d, k;
        Scanner in = new Scanner(System.in);
        System.out.print("Enter Number of Rows and Columns of First Matrix : ");
        m = in.nextInt();
        n = in.nextInt();
        int first[][] = new int[m][n];
        System.out.print("Enter First Matrix Elements : ");
        for(c=0 ; c<m; c++)
        {
            for(d=0; d<n; d++)
            {
                first[c][d] = in.nextInt();
            }
        }
        System.out.print("Enter Number of Rows and Columns of Second Matrix : ");
        p = in.nextInt();
```

```

q = in.nextInt();
if ( n != p )
{
    System.out.print("Matrix of the entered order can't be
Multiplied..!!");
}
else
{
    int second[][] = new int[p][q];
    int multiply[][] = new int[m][q];
    System.out.print("Enter Second Matrix Elements :\n");
    for(c=0; c<p; c++)
    {
        for(d=0; d<q; d++)
        {
            second[c][d] = in.nextInt();
        }
    }
    System.out.print("Multiplying both Matrix...\n");
    for(c=0; c<m; c++)
    {
        for(d=0; d<q; d++)
        {
            for(k=0; k<p; k++)
            {
                sum = sum + first[c][k]*second[k][d];
            }
            multiply[c][d] = sum;
            sum = 0;
        }
    }
    System.out.print("Multiplication Successfully performed..!!\n");
    System.out.print("Now the Matrix Multiplication Result is :\n");
    for(c=0; c<m; c++)
    {
        for(d=0; d<q; d++)
        {
            System.out.print(multiply[c][d] + "\t");
        }
        System.out.print("\n");
    }
}

```

## UNIT- II

### **Topics to be covered:**

#### **Collection Framework: Vectors in Java**

#### **Need/ Purpose to go for Collections:**

- An array is an indexed collection of fixed number of homogeneous data elements.
- The main advantage of arrays is we can represent multiple values with a single variable
- So that reusability of the code will be improved

#### **Limitations of Object type Arrays:**

- Arrays are fixed in size i.e., once we create an array with some size there is no chance of increasing or decreasing it's size based on our requirement. Hence to use arrays we should know the size in advance
- Arrays concept is not implemented based on any data structure hence readymade method support is not available for every requirement. We have to write the code explicitly.

#### **Collections:**

- To overcome the above limitations of arrays we move on to collections
- Collections are growable in nature i.e., based on the requirement we can increase or decrease the size
- Collections can hold both homogeneous & heterogeneous elements
- Every collection class is implemented based on some standard data structure. Hence readymade support is available

## **Vectors**

1. Vector class is in **java.util** package of java.
2. Vector is dynamic array which can **resize/ grow** automatically according to the required need.
3. Vector **does not require any fix dimension** like String array and int array.
4. The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index.
5. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
6. Vectors (the **java.util.Vector** class) are commonly used instead of arrays, because they expand automatically when new data is added to them.
7. Vectors can hold only Objects and not primitive types (eg, int). If you want to put a primitive type in a Vector, put it inside an object (eg, to save an integer value use the Integer class or define your own class). If you use the Integer wrapper, you will not be able to change the integer value, so it is sometimes useful to define your own class.
8. Duplicate objects are allowed
9. Insertion order is preserved
10. Most of the methods present in vector are synchronized. Hence vector object is Thread-safe

### **To Create a Vector:**

Following are the list of constructors provided by the vector class.

#### **1. Vector v = new Vector();**

-Creates an empty vector object with default initial capacity 10, Once vector reaches it's max capacity a new vector Object will be created with  
new capacity=2\*current capacity

#### **2. Vector v = new Vector(int initialCapacity);**

-Create an empty vector object with specified initial capacity

**By which it Increases the performance**

**3. Vector v = new Vector(int initialCapacity, int incrementalCapacity);**

-Create a Vector with an initial size **with incremental value**

**4. Vector v=new Vector(Collection c);**

-Create an equivalent Vector object for the given collection

**Common Vector Methods**

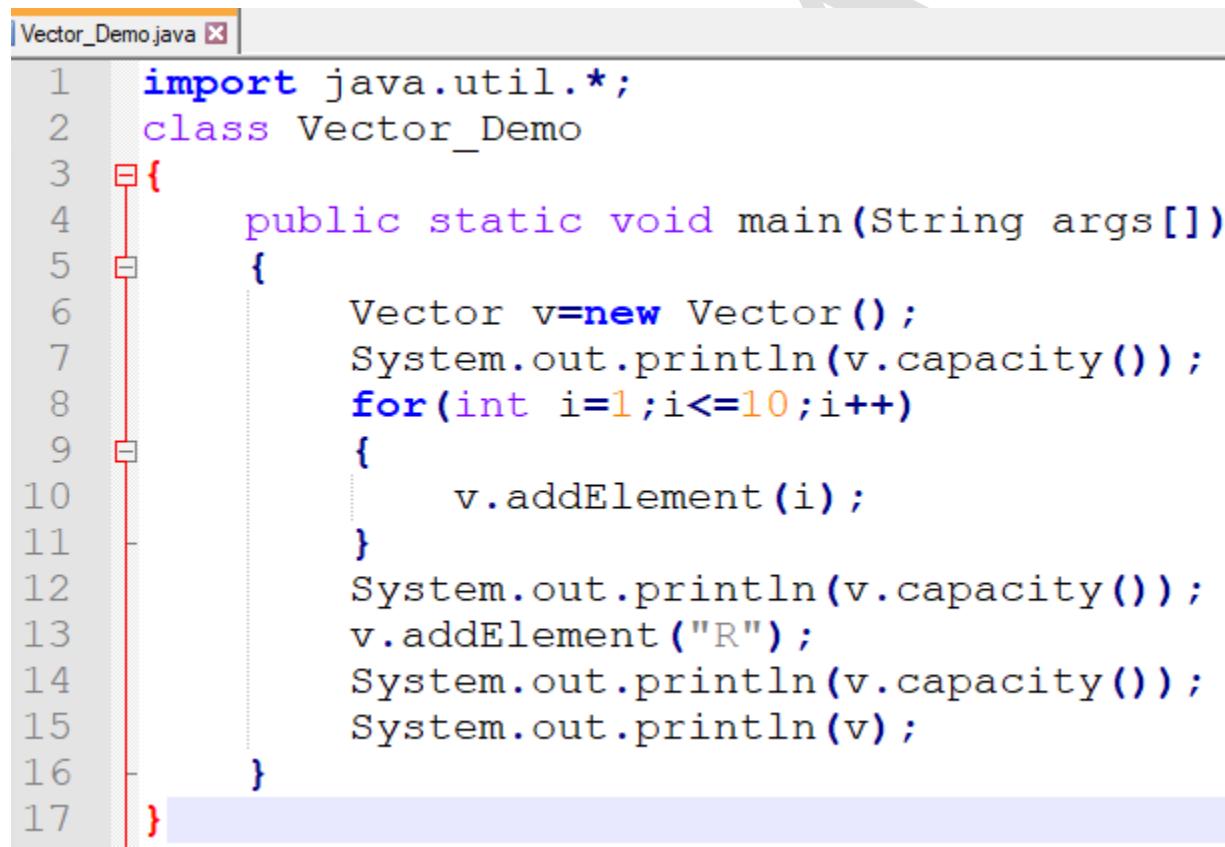
There are many methods in the Vector class and its parent classes.

| Method                 | Description  |
|------------------------|--|
| v.add(o)               | adds Object o to Vector v  |
| v.add(i, o)            | Inserts Object o at index i, shifting elements up as necessary.  |
| v.clear()              | removes all elements from Vector v   |
| v.contains(o)          | Returns true if Vector v contains Object o   |
| v.elementAt(int index) | To access the element present at particular index  |
| v.firstElement(i)      | Returns the first element.   |
| v.get(i)               | Returns the object at int index i.   |
| v.lastElement(i)       | Returns the last element.  |
| v.listIterator()       | Returns a ListIterator that can be used to go over the Vector. This is a useful alternative to the for loop. |
| v.remove(i)            | Removes the element at position i, and shifts all following elements down.                                   |
| v.removeAllElements()  | To remove all the elements present in an vector  |
| v.set(i,o)             | Sets the element at index i to o.  |
| v.capacity()           | Displays the Default capacity as 10  |
| v.size()               | Returns the number of elements in Vector v.  |

|                     |   |
|---------------------|---|
| v.toArray(Object[]) | The array parameter can be any Object subclass (eg, String). This returns the vector values in that array (or a larger array if necessary). This is useful when you need the generality of a Vector for input, but need the speed of arrays when processing the data. |
|---------------------|---|

## Demo Program over Vector Class Constructors:

### Example 1:



```

1 import java.util.*;
2 class Vector_Demo
3 {
4     public static void main(String args[])
5     {
6         Vector v=new Vector();
7         System.out.println(v.capacity());
8         for(int i=1;i<=10;i++)
9         {
10            v.addElement(i);
11        }
12        System.out.println(v.capacity());
13        v.addElement("R");
14        System.out.println(v.capacity());
15        System.out.println(v);
16    }
17 }
```

### Output:

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Vector_Demo
10
10
10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, R]
```

**Example 2:**

```
Vector_Demo.java |  
1 import java.util.*;  
2 class Vector_Demo  
3 {  
4     public static void main(String args[])  
5     {  
6         Vector v=new Vector(25);  
7         System.out.println(v.capacity());  
8         for(int i=1;i<=10;i++)  
9         {  
10             v.addElement(i);  
11         }  
12         System.out.println(v.capacity());  
13         v.addElement("R");  
14         System.out.println(v.capacity());  
15         System.out.println(v);  
16     }  
17 }  
18 ●
```

**Output:**

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Vector_Demo  
25  
25  
25  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, R]
```

**Example 3:**

```
1 import java.util.*;
2 class Vector_Demo
3 {
4     public static void main(String args[])
5     {
6         Vector v=new Vector(10,5);
7         System.out.println(v.capacity());
8         for(int i=1;i<=10;i++)
9         {
10            v.addElement(i);
11        }
12        System.out.println(v.capacity());
13        v.addElement("R");
14        System.out.println(v.capacity());
15        System.out.println(v);
16    }
17 }
```

**Output:**

```
C:\Users\Ravikanth\Desktop\OOPSJava>java Vector_Demo
10
10
15
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, R]
```

**Advantages of Vector:-**

- Size of the vector can be changed
- Multiple objects can be stored
- Elements can be deleted from a vector

**Disadvantages of Vector:**

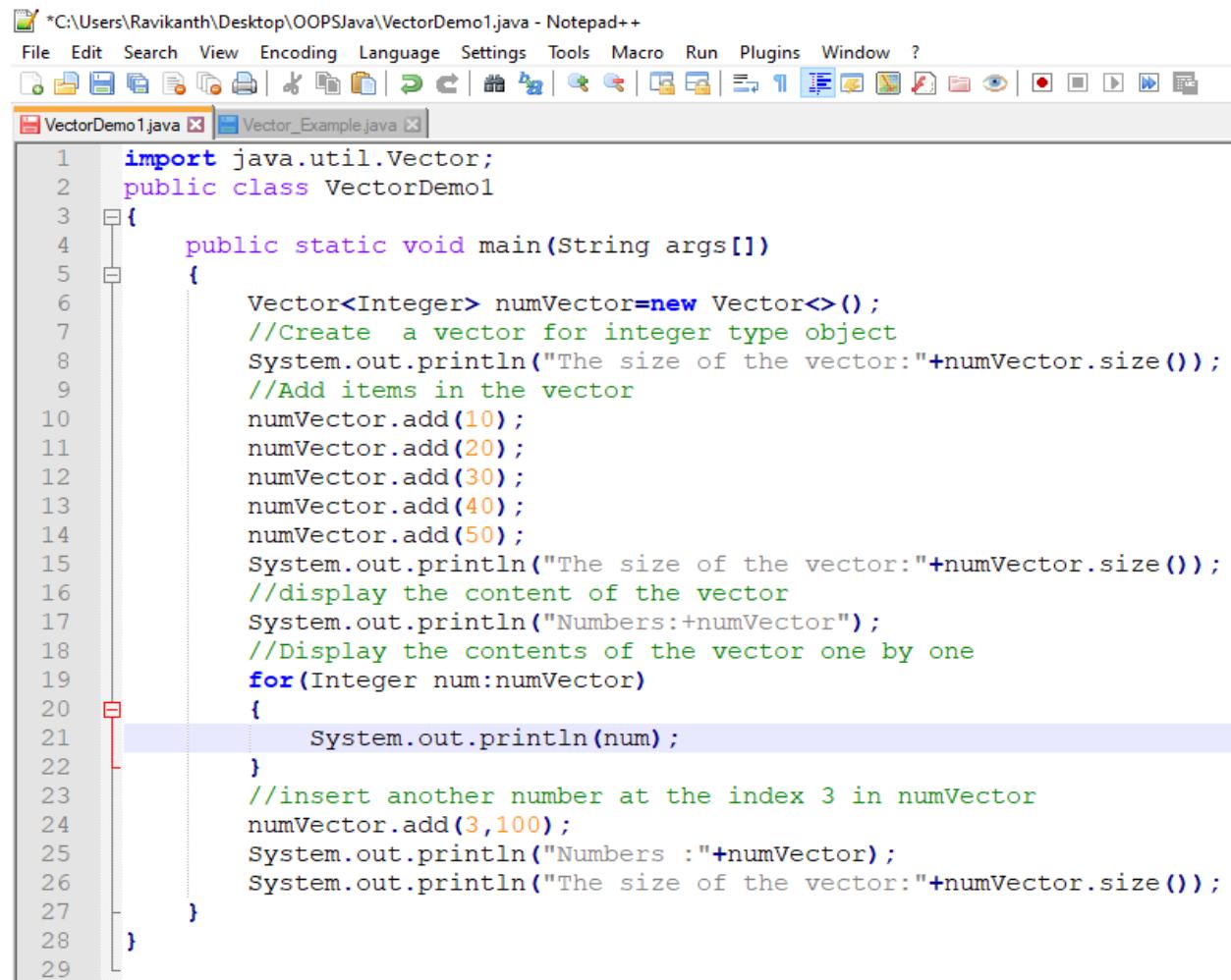
- A vector is an object, memory consumption is more.

**Differences between Arrays and Collections**

| <b>Arrays</b>                                      | <b>Collections</b>   |
|--|--|
| Arrays are fixed in size                           | Collections are growable in nature   |
| Wrt to memory arrays are not suitable              | Wrt to memory Collections are recommended to use                             |
| Wrt Performance Arrays are recommended to use      | Wrt Performance collections are not recommended to use                       |
| Array can hold only homogeneous data type elements | Collections can hold both homogeneous and heterogeneous elements             |
| Arrays can hold both primitives and object types   | Collections can hold only objects but not primitives                         |
| There is no underlying data structure for arrays   | Every collections class is implemented based on some standard data structure |

**Differences between a Vector and an Array**

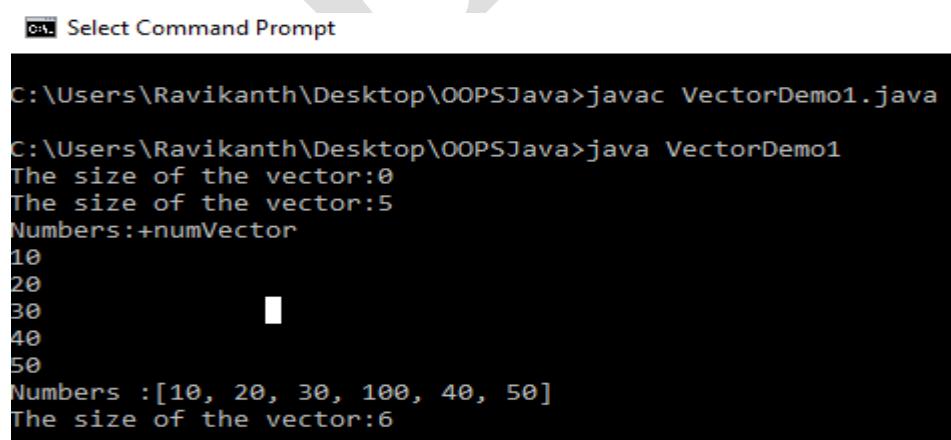
1. A vector is a dynamic array, whose size can be increased, whereas an array size can not be changed.
2. Reserve space can be given for vector, whereas for arrays can not.
3. A vector is a class whereas an array is not.
4. Vectors can store any type of objects, whereas an array can store only homogeneous values.

**Example 4:**


```

1 import java.util.Vector;
2 public class VectorDemol
3 {
4     public static void main(String args[])
5     {
6         Vector<Integer> numVector=new Vector<>();
7         //Create a vector for integer type object
8         System.out.println("The size of the vector:"+numVector.size());
9         //Add items in the vector
10        numVector.add(10);
11        numVector.add(20);
12        numVector.add(30);
13        numVector.add(40);
14        numVector.add(50);
15        System.out.println("The size of the vector:"+numVector.size());
16        //display the content of the vector
17        System.out.println("Numbers:+numVector");
18        //Display the contents of the vector one by one
19        for(Integer num:numVector)
20        {
21            System.out.println(num);
22        }
23        //insert another number at the index 3 in numVector
24        numVector.add(3,100);
25        System.out.println("Numbers :"+numVector);
26        System.out.println("The size of the vector:"+numVector.size());
27    }
28 }
29

```

**Output:**


```

Select Command Prompt

C:\Users\Ravikanth\Desktop\OOPSJava>javac VectorDemo1.java
C:\Users\Ravikanth\Desktop\OOPSJava>java VectorDemol
The size of the vector:0
The size of the vector:5
Numbers:+numVector
10
20
30
40
50
Numbers :[10, 20, 30, 100, 40, 50]
The size of the vector:6

```

**Example 5:**

```

2   public class Vector_Example1
3   {
4       public static void main(String args[])
5       {
6           Vector v=new Vector();
7           //Vector v1=new Vector();
8           //Vector<Integer> v=new Vector<Integer>(); for integer elements
9           //Vector<String> v=new Vector<String>(); for String elements
10          v.add(7);//adding element
11          v.add('a');
12          v.add("RGUKT");
13          v.add(3,4);//adding element at index(index,element)
14          //add at index method moves the already present at that index and added current data
15          v.set(3,"RGUKT1");//set the element at some index
16          // set method replaces the actual index information
17          //v=v1.clone();
18          boolean b=v.contains("RGUKT");//searching for element="RGUKT"
19          System.out.println(v.indexOf(7));//to know the index of element=7
20          System.out.println(v.get(2));//to get the element at index=2
21          System.out.println(v.elementAt(3));// same as get(3);
22          System.out.println(v.elementAt(2));// same as get(2);
23          System.out.println(v.elementAt(1));// same as get(1);
24          System.out.println(v.elementAt(0));// same as get(0);
25          System.out.println(b);// printing vector
26          System.out.println(v.firstElement());//print first element
27          System.out.println(v.lastElement()); // printing last element
28          System.out.println("size= "+v.size());// to get the size i.e element present in vector
29          System.out.println(v.capacity());//gives the capacity of vector
30          // printing vector Using for loop
31          for(int i=0;i<v.size();i++)
32          {
33              System.out.print(" "+v.get(i));
34          }
35      }

```

**Output:**

```

C:\Users\Ravikanth\Desktop\OOPSJava>java Vector_Example1
0
RGUKT
RGUKT1
RGUKT
a
7
true
7
RGUKT1
size= 4
10
 7 a RGUKT RGUKT1
C:\Users\Ravikanth\Desktop\OOPSJava>

```

UNIT-IITopic for the Day: Java Wrapper Classes

What is the need to work with Wrapper Classes

On Internet, we can only use objects not primitive datatypes (**byte, short, int, long, float, double, char , Boolean** ), we need to convert Primitive Datatypes into Object Type.

In new version because of AutoBoxing You will not get compile time error

```

WC_Ex1.java
1 import java.util.*;
2 class WC_Ex1
3 {
4     public static void main(String args[])
5     {
6         ArrayList l=new ArrayList();
7         l.add(10);
8         System.out.println(l);
9     }
10 }

Command Prompt

C:\Users\Ravikanth\Desktop\OOPSJava>javac WC_Ex1.java
Note: WC_Ex1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\Ravikanth\Desktop\OOPSJava>java WC_Ex1
[10]

```

```

import java.util.*;
class WC_Ex1
{
    public static void main(String args[])
    {
        ArrayList l=new ArrayList();
        l.add(10);
        System.out.println(l);
    }
}

```

Perfectly Valid

```

C:\Users\Ravikanth\Desktop\OOPSJava>java WC_Ex1
[10]

```

**Main Objectives of Wrapper Classes are:**

1. To wrap primitives into object form so that we can handle primitives also just like objects
2. To define several utility methods for primitives ( like `toString()`...)

For every primitive method wrapper class is going to create space by calling utility method

**Note:** For every primitive type corresponding Wrapper class is available

**Below table lists wrapper classes in Java API with constructor details.**

| Primitive            | Wrapper Class          | Constructor Argument                 |
|----------------------|------------------------|--------------------------------------|
| <code>boolean</code> | <code>Boolean</code>   | <code>boolean or String</code>       |
| <code>Byte</code>    | <code>Byte</code>      | <code>byte or String</code>          |
| <code>Char</code>    | <code>Character</code> | <code>Char</code>                    |
| <code>Int</code>     | <code>Integer</code>   | <code>int or String</code>           |
| <code>Float</code>   | <code>Float</code>     | <code>float, double or String</code> |
| <code>double</code>  | <code>Double</code>    | <code>double or String</code>        |
| <code>Long</code>    | <code>Long</code>      | <code>long or String</code>          |
| <code>short</code>   | <code>Short</code>     | <code>short or String</code>         |

For float we have 3 constructors :

```
Float f = new Float(float)
Float f = new Float(double)
Float f = new Float(String)
```

For char class only one constructor is available:

```

class Test
{
    public static void main(String[] args)
    {
        Character ch = new Character('a');
        System.out.println(ch);
    }
}

```

VALID:

```

class Test
{
    public static void main(String[] args)
    {
        Character ch = new Character("a");
        System.out.println(ch);
    }
}

```

Invalid: Error

Now lets check with Boolean:

If we are passing boolean primitive as argument the only allowed values are : true or false

If we are passing String argument then case is not important and content is also important.

If the content| is true(lower case or upper case) then it is treated as true otherwise it is treated as false.

```
class Test
{
    public static void main(String[] args)
    {
        Boolean X = new Boolean("yes");//false
        Boolean Y = new Boolean("no");//false
        System.out.println(X);//false
        System.out.println(Y);//false
        System.out.println(X.equals(Y));//true
    }
}
```

Note: Here in this case content is going to be checked and compared not the value

**Detailed Description**

Each of Java's eight primitive data types has a class dedicated to it. These are known as wrapper classes because they "wrap" the primitive data type into an object of that class. The wrapper classes are part of the `java.lang` package, which is imported by default into all Java programs.

The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

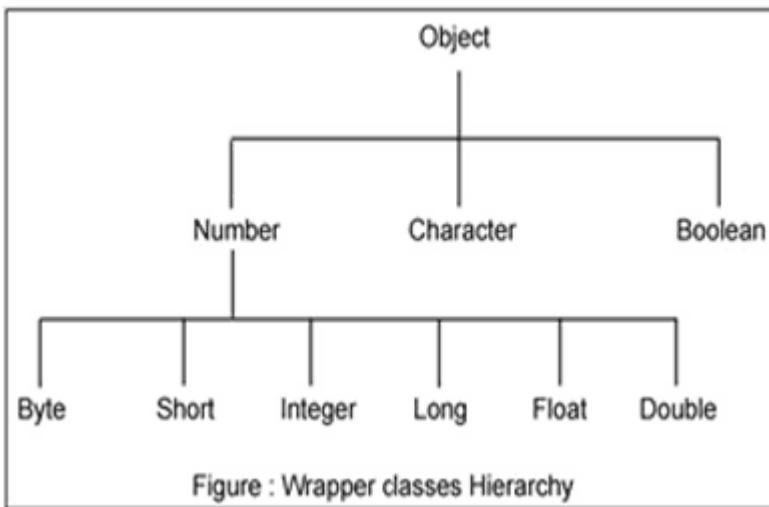
```
int x = 25;
Integer y = new Integer(33);
```

The first statement declares an `int` variable named `x` and initializes it with the value 25. The second statement instantiates an `Integer` object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable `y`.

Below table lists wrapper classes in Java API with constructor details.

| Primitive | Wrapper Class | Constructor Argument    |
|-----------|---------------|-------------------------|
| Boolean   | Boolean       | boolean or String       |
| Byte      | Byte          | byte or String          |
| Char      | Character     | char                    |
| Int       | Integer       | int or String           |
| Float     | Float         | float, double or String |
| Double    | Double        | double or String        |
| Long      | Long          | long or String          |
| Short     | Short         | short or String         |

Below is wrapper class hierarchy as per Java API



As explain in above table all wrapper classes (except Character) take String as argument constructor. Please note we might get NumberFormatException if we try to assign invalid argument in the constructor. For example to create Integer object we can have the following syntax.

```
Integer intObj = new Integer (25);
Integer intObj2 = new Integer ("25");
```

Here in we can provide any number as string argument but not the words etc. Below statement will throw run time exception (NumberFormatException)

**Example:**

```
/ Java program to demonstrate Wrapping and UnWrapping
// in Java Classes
class Wrapper_Class
{
    public static void main(String args[])
    {
        // byte data type
        byte a = 1;
        // wrapping around Byte object
        Byte byteobj = new Byte(a);

        // int data type
        int b = 10;
        //wrapping around Integer object
        Integer intobj = new Integer(b);

        // float data type
        float c = 18.6f;
        // wrapping around Float object
        Float floatobj = new Float(c);
```

```
// double data type
double d = 250.5;
// Wrapping around Double object
Double doubleobj = new Double(d);

// char data type
char e='a';
// wrapping around Character object
Character charobj=e;

// printing the values from objects
System.out.println("Values of Wrapper objects (printing as objects)");
System.out.println("Byte object byteobj: " + byteobj);
System.out.println("Integer object intobj: " + intobj);
System.out.println("Float object floatobj: " + floatobj);
System.out.println("Double object doubleobj: " + doubleobj);
System.out.println("Character object charobj: " + charobj);

// objects to data types (retrieving data types from objects)
// unwrapping objects to primitive data types
byte bv = byteobj;
int iv = intobj;
float fv = floatobj;
double dv = doubleobj;
char cv = charobj;

// printing the values from data types
System.out.println("Unwrapped values (printing as data types)");
System.out.println("byte value, bv: " + bv);
System.out.println("int value, iv: " + iv);
System.out.println("float value, fv: " + fv);
System.out.println("double value, dv: " + dv);
System.out.println("char value, cv: " + cv);
}
```

**Trace the Program Manually and Check the Results**

## **UNIT-IV**

### **Topics to be covered:**

**Exception Handling:** Limitations of Error handling, Advantages of Exception Handling, Types of Errors, Basics of Exception Handling, try blocks, throwing an exception, catching an exception, finally statement

#### **Exception Handling:-**

The process of converting system error messages into user friendly error message is known as Exception handling. This is one of the powerful features of Java to handle run time error and maintain normal flow of java application.

An exception is a problem that arises during the execution of a program

- User entered invalid data
- File Not Found

#### **Exception**

An Exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

#### **Usage of Exception Handling**

Handling the exception is nothing but converting system error generated message into user friendly error message. Whenever an exception occurs in the java application, JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user friendly error message. You can convert system error message into user friendly error message by using exception handling feature of java.

For Example: when you divide any number by zero then system generate / by zero so this is not understandable by user so you can convert this message into user friendly error message like Don't enter zero for denominator.

### **Hierarchy of Exception classes**

Throwable acts as root for java exception hierarchy

- Throwable class consists of two child classes: Exception & Error

#### **Exception:**

##### **1. Most of the cases Exceptions are caused by our programs and this are recoverable**

- **Ex:** For example, if our program requirement is to read data from a remote file location at runtime and if the file is not found then we will get **FileNotFoundException**
- If FileNotFoundException occurs then we can provide a local file and rest of the program will be continued normally

```
try
{
    // Read data from a remote file location
}
catch( FileNotFoundException e)
{
    // Use Local file & continue rest of the program normally
}
```

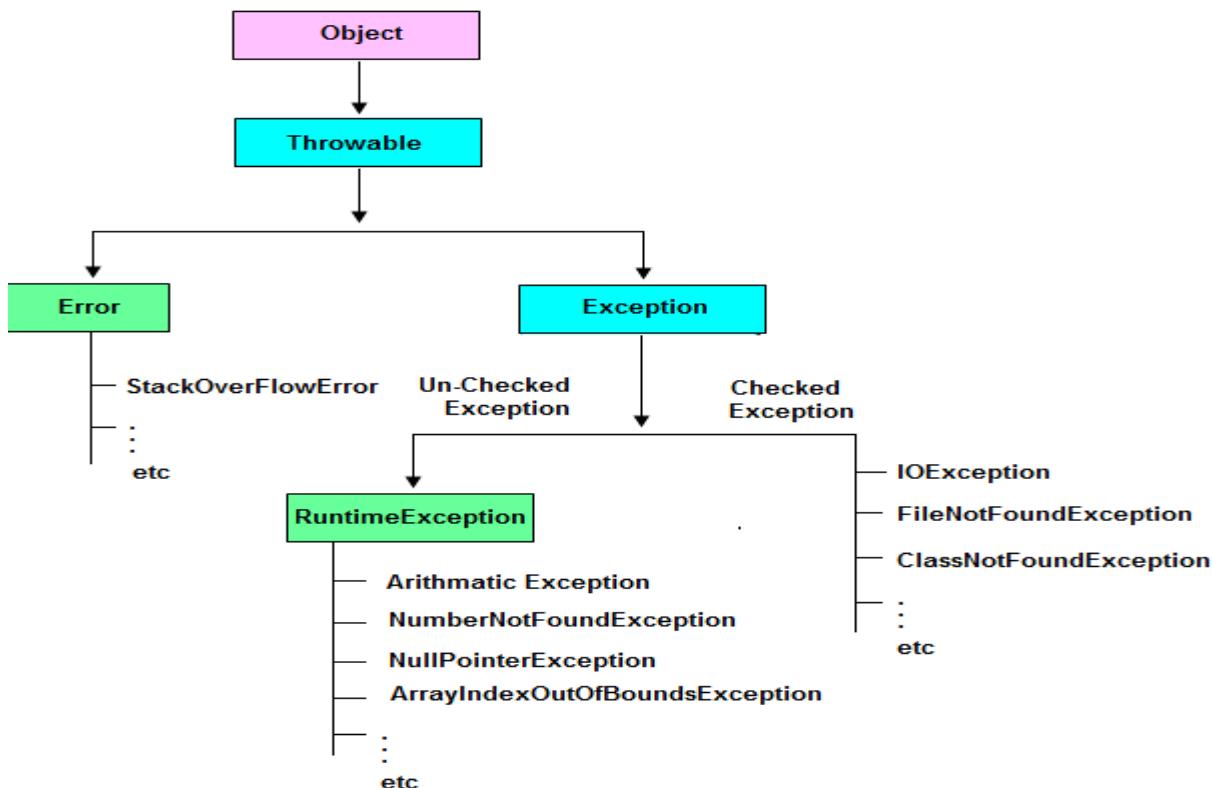
#### **Error:**

##### **1. Most of the times errors are not caused by the program these are due to lack of system resources**

**Errors are non recoverable**

- **Example:**
- For Example if OutOfMemory error occurs being a programmer we can't do anything and the program will be terminated abnormally

- System admin or server admin is responsible to increase heap memory



#### List of Java UnChecked Exceptions Defined in `java.lang`.

|   |   |
|---|---|
| <code>ArithmaticException</code>            | Arithmatic error, such as divide-by-zero.                         |
| <code>ArrayIndexOutOfBoundsException</code> | Array index is out-of-bounds.                                     |
| <code>ArrayStoreException</code>            | Assignment to an array element of an incompatible type.           |
| <code>ClassCastException</code>             | Invalid cast.   |
| <code>IllegalArgumentException</code>       | Illegal argument used to invoke a method.                         |
| <code>IllegalMonitorStateException</code>   | Illegal monitor operation, such as waiting on an unlocked thread. |
| <code>IllegalStateException</code>          | Environment or application is in incorrect state.                 |
| <code>IllegalThreadStateException</code>    | Requested operation not compatible with the current thread state. |
| <code>IndexOutOfBoundsException</code>      | Some type of index is out-of-bounds.                              |
| <code>NegativeArraySizeException</code>     | Array created with a negative size.                               |
| <code>NullPointerException</code>           | Invalid use of a null reference.                                  |
| <code>NumberFormatException</code>          | Invalid conversion of a string to a numeric format.               |

|                                 |  |
|---------------------------------|--|
| SecurityException               | Attempt to violate security.                     |
| StringIndexOutOfBoundsException | Attempt to index outside the bounds of a string. |
| UnsupportedOperationException   | An unsupported operation was encountered.        |

### List of Java Checked Exceptions Defined in java.lang.

|                            |   |
|----------------------------|---|
| ClassNotFoundException     | Class not found.  |
| CloneNotSupportedException | Attempt to clone an object that does not implement the Cloneable interface. |
| IllegalAccessException     | Access to a class is denied.  |
| InstantiationException     | Attempt to create an object of an abstract class or interface.              |
| InterruptedException       | One thread has been interrupted by another thread.                          |
| NoSuchFieldException       | A requested field does not exist.   |
| NoSuchMethodException      | A requested method does not exist.  |

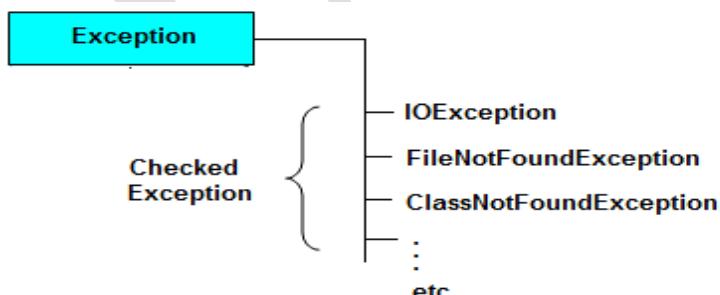
### Type of Exception

- Checked Exception
- Un-Checked Exception

### Checked Exception

Checked Exception are the exception which checked at compile-time. These exception are directly sub-class of java.lang.Exception class.

Checked means checked by compiler so checked exception are checked at compile-time.

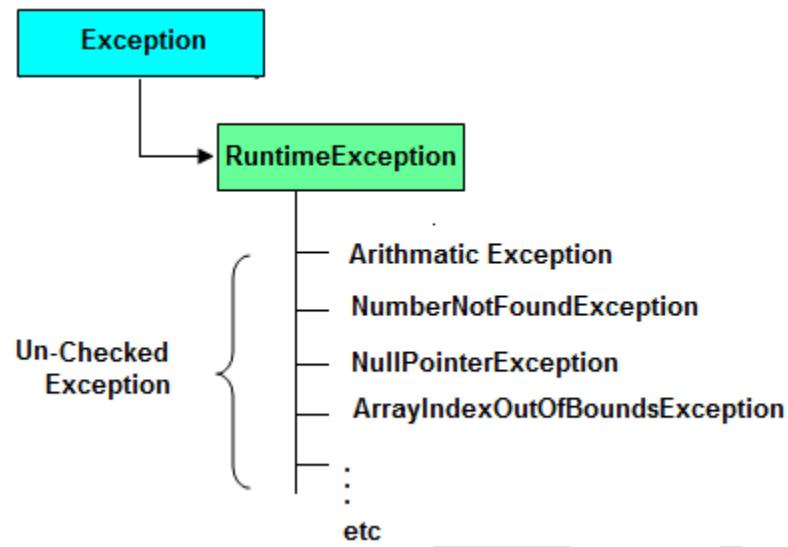


### Un-Checked Exception

Un-Checked Exception are the exception both identifies or raised at run time. These exceptions are directly sub-class of java.lang.RuntimeException class.

**Note:** In real time application mostly we can handle un-checked exception.

Only for remember: Un-checked means not checked by compiler so un-checked exception are checked at run-time not compile time.



### Handling the Exception

Handling the exception is nothing but converting system error generated message into user friendly error message in others word whenever an exception occurs in the java application, JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user-friendly error message. You can convert system error message into user-friendly error message by using exception handling feature of java.

We make use of Five keywords for Handling the Exception

- try
- catch
- finally
- throws
- throw

**Control Flow in try – catch Block**

```
try
{
    statement 1;
    statement 2;
    statement 3;
}
catch( X e)
{
    statement 4;
}
statement 5;
```

**Case1:**

If there is no exception

1,2,3,5 stmts will be executed and it is Normal Termination

**Case 2:**

If an exception raised at statement 2 and corresponding catch block is matched

1,4,5 stmts is executed and it is an normal termination

**Case 3:**

If an exception raised at statement 2 and corresponding catch block is not matched

1 stmst is executed and it is an abnormal termination

**Case 4:**

If an exception raised at statement 4 or statement 5

It is always an abnormal situation

**Conclusions Made:**

- With in the try block if anywhere exception raised then rest of try block won't be executed even though we handled that exception. Hence length of try block should be as less as possible. And we have to take only risky code within try block
- In addition to try block there may be a chance of raising an exception inside catch and finally blocks
- If any statements raises an exception which is not part of try block then it is always an Abnormal situation

**Control Flow in try-catch-finally:**

```
try
{
    statement 1;
    statement 2;
    statement 3;

}
catch( X e )
{
}
```

**statement 4;**

}

**finally**

{

**statement 5;**

}

**statement 6;**

#### **Case 1:**

If there is no exception

1,2,3,5,6 statements are executed and it is normal termination

#### **Case 2:**

If an exception raised at statement 2 and corresponding catch block matched

Statements 1,4,5,6 are executed and it is Normal Termination

#### **Case 3:**

If an exception is raised at statement 2 and the corresponding catch block is not matched

Statements 1,5 are executed and it is a Abnormal Termination

#### **Case 4:**

If an exception is raised at statement 4, then it is always an Abnormal termination, but before Abnormal termination finally block will be executed

### Case 5:

If an exception raised at statement 5 or at statement 6 then it is always Abnormal termination

#### Syntax for handling the exception

```
try
{
    //risky code
}

catch( X e)
{
    // Handling code
}

finally
{
    //Cleanup code
}
```

**Try with multiple catch Blocks**

```
try
{
    // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
    // statements provides user friendly error message
}
catch(type of exception-2 object-2)
{
    // statements provides user friendly error message
}
finally
{
    // statements which will execute compulsory
}
```

**Note:**

finally meant for cleanup activities related to try block, where as  
finalize( ) meant for clean-up activities related to an Object

## Various possible combinations of try-catch-finally

### Case 1:

```
try  
{  
}  
  
catch( X e)  
{  
}
```

Perfectly Valid

### Case 2:

```
try  
{  
}  
  
catch( X e)  
{  
}  
  
catch(X e)  
{
```

```
}
```

CompileTime Error: Exception X has already been caught

### Case 3:

```
try  
{  
}  
  
catch( X e)  
{  
}  
  
catch( Y e)  
{  
}
```

Perfectly valid as X and Y are different exceptions

### Case 4:

```
try  
{  
}  
  
catch( Exception e) // Parent
```

```
{  
}  
  
catch(ArithemeticException e) // Child  
  
{  
}  
}
```

CTE: Exception java.lang.AE has already been caught

#### Case 5:

```
try  
{  
}  
  
catch(ArithemeticException e) // Child for AE  
  
{  
}  
  
catch( Exception e) // Parent for Non- AE  
  
{  
}  
}
```

Perfectly Valid Syntax

**Case 6:**

```
try  
{  
}
```

CTE: try without catch or finally block

**Case 7:**

```
catch( X e)  
{  
}
```

CTE: Catch without try

**Case 8:**

```
finally  
{  
}
```

CTE: finally without try

**Case 9:**

```
try  
{
```

```
}
```

```
finally
```

```
{
```

```
}
```

Perfectly Valid Syntax

Explanation: thought it is abnormal condition still if I want to close my database connectivity or network connectivity we can go ahead

#### Case 10:

```
try
```

```
{
```

```
}
```

```
S.O.println("Hello");
```

```
catch(X e)
```

```
{
```

```
}
```

CTE1: try without catch or finally

CTE2: catch without try

**Case 11:**

```
try  
{  
}  
  
catch(X e)  
{  
}  
  
S.O.pln("Hello");  
  
catch(Y e)  
{  
}
```

CTE1: catch without try

**Case 12:**

```
try  
{  
}  
  
catch(X e)  
{
```

```
}
```

```
S.O.pln("Hello");
```

```
finally
```

```
{
```

```
}
```

```
CTE: finally without try
```

### Case 13:

```
try
```

```
{
```

```
}
```

```
catch(X e)
```

```
{
```

```
}
```

```
try
```

```
{
```

```
}
```

```
finally
```

```
{
```

}

Perfectly Valid syntax

### Case 14:

```
try  
{  
}  
  
finally  
{  
}  
  
catch(X e)  
{  
}
```

CTE: catch without try

### Case 15:

```
try  
{  
}  
  
catch(X e)
```

```
{  
}  
  
finally
```

```
{  
}  
  
finally  
{  
}
```

CTE: finally without try

#### Case 16:

```
try  
{  
    try  
    {  
        {  
    }  
  
    catch(X e)  
  
    {  
    }
```

```
finally  
{  
}  
}  
catch( X e)  
{  
}
```

Nested try catch perfectly valid syntax

### Case 17:

```
try  
{  
}  
catch(X e)  
{  
    try  
{  
}  
}  
finally
```

```
{  
}  
}
```

Perfectly Valid

**Case 18:**

```
try  
{  
}  
catch(X e)  
{  
}  
finally  
{  
    try  
{  
}  
}  
finally  
{  
}
```

}

}

Perfectly valid syntax

### Case 19:

try

{

try

{

}

}

catch( X e)

{

}

CTE: try without catch or finally

### Case 20:

try

S.O.Pln("Hello");

catch( X e)

{

}

CTE: Invalid as no curly braces for try

**Case 21:**

try

{

}

catch(X e)

S.O.P("HAI");

CTE: Invalid as no curly braces for catch

**Case 22:**

try

{

}

catch(X e)

{

}

finally

S.O.P("HAI");

CTE: Invalid as no curly braces for finally

### **Conclusions:**

- Whenever we are writing try block we should write catch or finally.  
That is “try” without catch or finally is invalid syntax
- Whenever we are writing catch block compulsory we should write try block that is catch without try is invalid
- Whenever we are writing finally block compulsory we should write try block. That is finally without try is invalid
- In try catch finally, order is important
- “try” with multiple catch blocks is valid but the order is important compulsory we should take from child to parent. By mistake if we are trying to take from parent to child then we will get compile time error
- If we are defining two catch blocks for the same exception we will get compile time error
- We can define try-catch-finally with in the try, with in the catch and within finally blocks. Hence nesting of try-catch-finally is valid
- For try-catch-finally curly braces are mandatory

## **try and catch block**

### **try block**

- Inside try block we write the block of statements which causes executions at run time in other words try block always contains problematic statements.
- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
- After executing appropriate catch block, even though we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

### **catch block**

- Inside catch block we write the block of statements which will generates user friendly error messages.
- Catch block will execute exception occurs in try block.
- You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- At a time only one catch block will execute out of multiple catch blocks.
- in catch block you declare an object of sub class and it will be internally referenced by JVM.

### **Syntax**

```
try
{
    ....
}
/* Here no other statements are allowed
between try and catch block */
catch()
{
    ....
}
```

- Each and every try block must contains at least one catch block
- One try block can contains another try block that is nested or inner try block can be possible.

Syntax

```
try
{
    .....
}
try
{
    .....
}
}
```

#### Example without Exception Handling

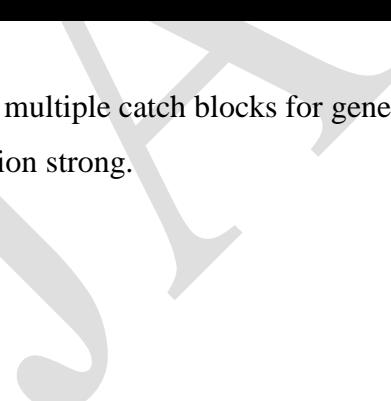
```
class ExceptionDemo
{
    public static void main(String[] args)
    {
        int a=10, ans=0;
        ans=a/0;
        System.out.println("Denominator can not be zero");
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac ExceptionDemo.java
C:\Users\IIIT\Desktop\RaviKanth>java ExceptionDemo
Exception in thread "main" java.lang.ArithmetricException: / by zero
at ExceptionDemo.main(ExceptionDemo.java:6)
```

**Example of Exception Handling**

```
class ExceptionDemo1
{
    public static void main(String[] args)
    {
        int a=10, ans=0;
        try
        {
            ans=a/0;
        }
        catch (Exception e)
        {
            System.out.println("Denominator cannot be zero");
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac ExceptionDemo1.java
C:\Users\IIIT\Desktop\RaviKanth>java ExceptionDemo1
Denominator cannot be zero
```

**Multiple catch block**

You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong.

**Example**

```
import java.util.*;
class ExceptionDemo2
{
    public static void main(String[] args)
    {
        int a, b, ans=0;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter any two numbers: ");
        try
        {
            a=s.nextInt();
            b=s.nextInt();
            ans=a/b;
            System.out.println("Result: "+ans);
        }
        catch(ArithmetcException ae)
        {
            System.out.println("Denominator can not be zero");
        }
        catch(Exception e)
        {
            System.out.println("Enter valid number");
        }
    }
}
```



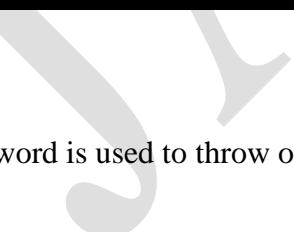
```
c:\ C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac ExceptionDemo2.java
C:\Users\IIIT\Desktop\RaviKanth>java ExceptionDemo2
Enter any two numbers:
10
0
Denominator can not be zero
C:\Users\IIIT\Desktop\RaviKanth>java ExceptionDemo2
Enter any two numbers:
2
s
Enter valid number
C:\Users\IIIT\Desktop\RaviKanth>java ExceptionDemo2
Enter any two numbers:
20
30
Result: 0
C:\Users\IIIT\Desktop\RaviKanth>
```

### **finally Block in Exception Handling**

- Inside finally block we write the block of statements which will released or close or terminate the resource (file or database) where data store permanently.
- Finally block will execute compulsory
- Writing finally block is optional.
- You can write finally block for the entire java program
- In some of the circumstances one can also write try and catch block in finally block.

**Example**

```
class ExceptionDemo3
{
    public static void main(String[] args)
    {
        int a=10, ans=0;
        try
        {
            ans=a/0;
        }
        catch (Exception e)
        {
            System.out.println("Denominator cannot be zero");
        }
        finally
        {
            System.out.println("I am from finally block");
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac ExceptionDemo3.java
C:\Users\IIIT\Desktop\RaviKanth>java ExceptionDemo3
Denominator cannot be zero
I am from finally block
```

**Throw**

throw keyword is used to throw our own exception in the program.

**Syntax :**

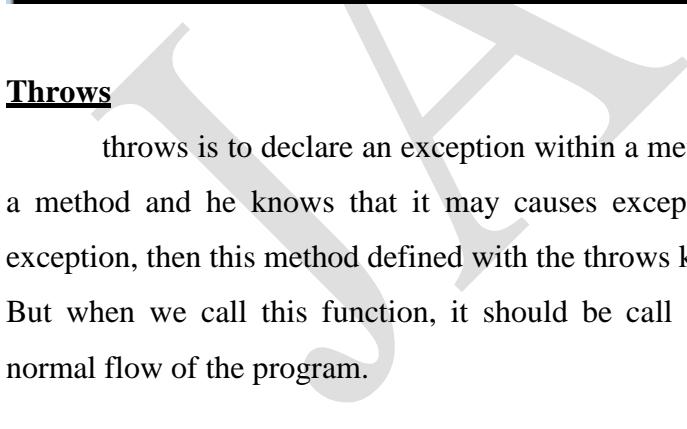
```
throw new Exception_class
```

In the below example, class checkDistance has one constructor to check whether entered distance valid or not. If the distance is less than zero than program throws a arithmetic exception with message 'Please enter valid distance...' .

```

class checkDistance
{
    int dist;
    checkDistance(int dist)
    {
        this.dist = dist;
        if(dist<0)
        {
            throw new ArithmeticException("Please enter valid distance...");
        }
        else
        {
            System.out.println("This is valid distance.");
        }
    }
    public static void main(String args[])
    {
        checkDistance v = new checkDistance(-10);
    }
}

```



```

C:\> C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac checkDistance.java
C:\Users\IIIT\Desktop\RaviKanth>java checkDistance
Exception in thread "main" java.lang.ArithmetricException: Please enter valid distance...
        at checkDistance.<init>(checkDistance.java:9)
        at checkDistance.main(checkDistance.java:18)

```

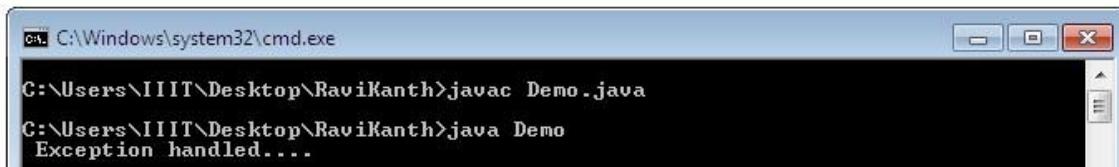
### Throws

throws is to declare an exception within a method. e.g suppose some programmer defines a method and he knows that it may causes exception, but he/she don't want to handle that exception, then this method defined with the throws keyword.

But when we call this function, it should be call with try and catch block to maintain the normal flow of the program.

In the below example, a method calc() may causes exception so we declare this method with throws keyword. But when we call this method, it should be call within try block to handle the exception.

```
class throwsDemo
{
    void calc() throws ArithmeticException
    {
        int c = 100/0;
    }
}
class Demo
{
    public static void main(String args[])
    {
        throwsDemo t = new throwsDemo();
        try
        {
            t.calc();
        }
        catch(Exception e)
        {
            System.out.println(" Exception handled.... ");
        }
    }
}
```



### User Defined Exception in Java

If any exception is design by the user known as user defined or Custom Exception.

Custom Exception is created by user.

Rules to design user defined Exception

1. Create a package with valid user defined name.
2. Create any user defined class.
3. Make that user defined class as derived class of Exception or RuntimeException class.
4. Declare parametrized constructor with string variable.
5. call super class constructor by passing string variable within the derived class constructor.
6. Save the program with public class name.java

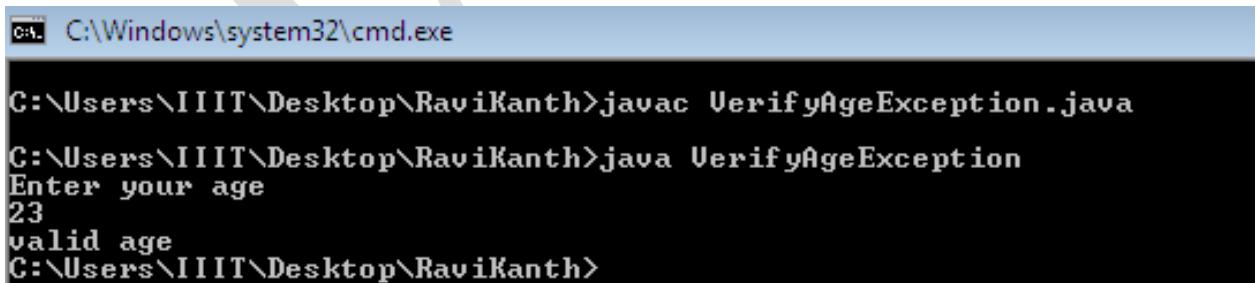
```
// AgeException.java → 6
package pack; → 1

public class AgeException extends Exception
{
    ↓   ↓
    2   3
    public AgeException(String s) → 4
    {
        super(s); → 5
    }
}
```

Example

```
import java.util.*;
class AgeException extends Exception
{
    public AgeException(String s)
    {
        super(s);
    }
}
class CheckAge
{
    public void verify(int age) throws AgeException
    {
        if (age>0)
        {
            System.out.print("valid age");
        }
        else
        {
            AgeException ae=new AgeException("Invalid age");
            throw(ae);
        }
    }
}
```

```
public class VerifyAgeException
{
    public static void main(String args[])
    {
        int a;
        System.out.println("Enter your age");
        Scanner s=new Scanner(System.in);
        a=s.nextInt();
        try
        {
            CheckAge ca=new CheckAge();
            ca.verify(a);
        }
        catch(AgeException ae)
        {
            System.err.println("Age should not be -ve");
        }
        catch(Exception e)
        {
            System.err.println(e);
        }
    }
}
```



The screenshot shows a Windows command prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The user has typed 'javac VerifyAgeException.java' and then 'java VerifyAgeException'. When prompted for an age, they enter '23', which is identified as a 'valid age'.

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac VerifyAgeException.java
C:\Users\IIIT\Desktop\RaviKanth>java VerifyAgeException
Enter your age
23
valid age
C:\Users\IIIT\Desktop\RaviKanth>
```

## **UNIT-IV**

### **Topics to be covered:**

**Packages:** Java API Packages, System Packages, Naming Conventions, Creating & Accessing a Package, Adding Class to a Package, Hiding Classes, Sample Programs

### **Package in Java**

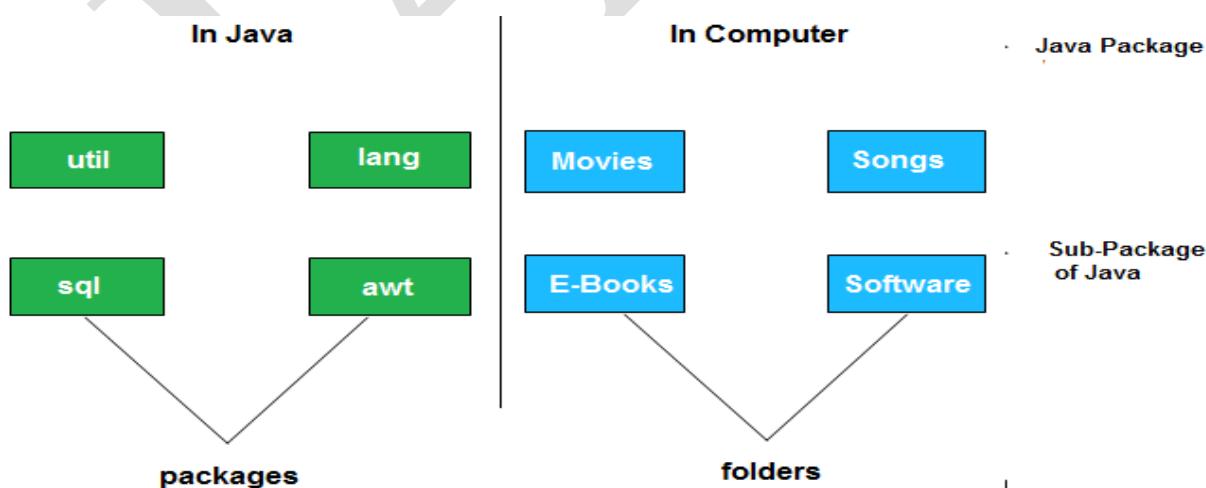
A package is a namespace that organizes a set of related classes and interfaces

or

A package is a collection of built-in classes, interfaces, abstract classes and sub packages based on their functionality

### **Purpose of package**

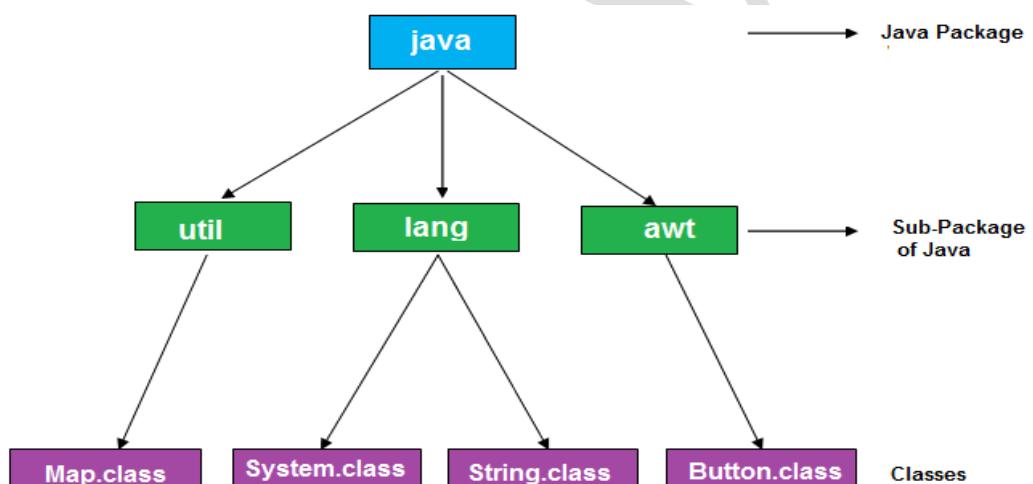
Conceptually packages are similar to different folders/ directories and sub-directories on your computers. The purpose of package concept is to provide common classes and interfaces for any program separately. In other words if we want to develop any class or interface which is common for most of the java programs than such common classes and interfaces must be place in a package.



The Java Platform provides an enormous class library ( a set of packages) suitable for developing various applications. This support in the library is known as “ Application Programming Interfaces” [ API].

Ex:

- A String object contains a state and behavior for character strings.
- A File object allows a programmer to easily create, delete, inspect , compare or modify a file over filesystem
- A Socket object allows for the creation and use of network sockets
- A GUI objects controls buttons and checkboxes and anything else related to Graphical User Interfaces



### A Package offers the following things:

- Namespace
- Organizational benefits of the related class files i.e., Classes , Interfaces, Enumerated Types
- Some level of security

## Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- Application development time is less, because reuse the code
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhance (improve)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

## Type of package

Packages are classified into two [ 02] types :

1. Built-in Packages or Pre-defined Packages or Standard Packages[ Java API]
2. User Defined Packages

## Built-in Package

- Package which are already designed by the Sun Microsystems and supply as a part of java software in the form of API support
- These Built-in packages are present in compressed files called JAR files( a JAR file or Java Archive is used for aggregating many files into one) “rt.jar” file present in JRE
- A package declaration resides at the top of a Java Source file. All source files to be placed in a package have a common package name
- They are inturn classified into 2 catagories : Core Packages and Extension Packages

## Some important Core packages are :

- java.lang [ Basic Language Functionalities]
- java.io [ File handling Operations]
- java.util [ Utility classes and collection data structure classes]
- java.math [Arbitrary precision arithmetic]
- java.awt [ Abstract window toolkit for native GUI components]

- java.awt.event [ AWT related event handler mechanisms]
  - java.applet [ To create an Applet and communicate with applet context]
  - java.net [ Network Programming ]
  - java.sql [ Java Database Connectivity ( JDBC) to access databases]
  - java.swing [ Lightweight programming for platform-independent rich GUI components]
- so on...

### **Some important extension packages are :**

- javax.swing
- javax.sql
- javax.servlet
- javax.servlet.http
- javax.servlet.jsp so on..

### **NOTE:**

**To include a library ( Built-in) package we have to use import keyword**

**Example: import java.awt.\*;**

### **Note:**

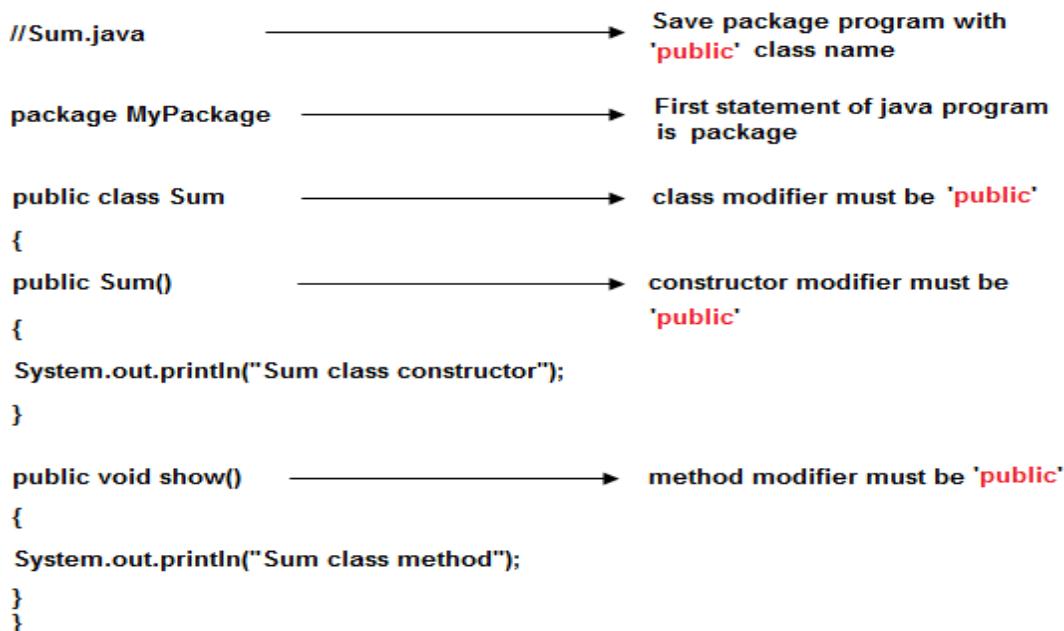
**java.lang is a special package, as it is imported by default in all the classes**

**For any kind of java application implicitly java.lang package is available. The major contribution of this package is it provides**

- Exceptional Handling classes
- Multi-threading Support
- String Handling
- Wrapper classes
- System Resources etc...

### Rules to create user defined package

- package statement should be the first statement of any package program.
- Choose an appropriate class name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main())
- modifier of constructor of the class which is present in the package must be public.  
(This is not applicable in case of interface because interface have no constructor.)
- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by default public)
- Every package program should be save either with public class name or public Interface name



### Compile package programs

- For compilation of package program first we save program with public className.java and it compile using below syntax:

**Syntax**

```
javac -d . className.java
```

**Syntax**

```
javac -d path className.java
```

**Explanations:** In above syntax "**-d**" is a specific tool which is tell to java compiler create a separate folder for the given package in given path. When we give specific path then it create a new folder at that location and when we use . (dot) then it crate a folder at current working directory.

**Note:** Any package program can be compile but can not be execute or run. These program can be executed through user defined program which are importing package program.

**User defined package**

- Our own created packages are known as User Defined Packages
- We can group different classes, interfaces, abstract classes etc in a separate directory and is known as a User-Defined Package
- Java compiler has got an option called “-d” [ directory] which specifies where to place the generated class files as a result of compilation
- “package” is the keyword used in java language to specify the package name.
- Package declaration should be the first statement of a java program

```
// Creation of User Defined Package
package OOPS;

// Creation of Class
public class A
{
    public void a(int i)
    {
        System.out.println("Hi"+i);
    }
}
```

**Now save the program as javac -d . A.java**

Java compiler will create A.class file and will be placed in OOPS folder

**Note :** If you want to save in current working directory itself then use[ . ] Dot is a system variable indicating present working directory

Now check whether the folder has got created or not

Now make use of “import” keyword to import the contents of a package in other java application

Save the file as **B.java**

```
// Accessing User Defined Package OOPS which is having class A
import OOPS.A;
class B
{
    public static void main(String aa[])
    {
        A a1=new A();
        a1.a(7);
    }
}
```

Now Compile and Run the A and B .java files

Run B.java file

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac -d . A.java
C:\Users\IIIT\Desktop\RaviKanth>javac B.java
C:\Users\IIIT\Desktop\RaviKanth>java B
Hai?
C:\Users\IIIT\Desktop\RaviKanth>
```

Note : If you want to create in any specific directory location then use path

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac -d C:/ A.java
C:\Users\IIIT\Desktop\RaviKanth>javac B.java
C:\Users\IIIT\Desktop\RaviKanth>java B
Hai?
```

**Ex2:**

**Including an Interface and a Class within the same package**

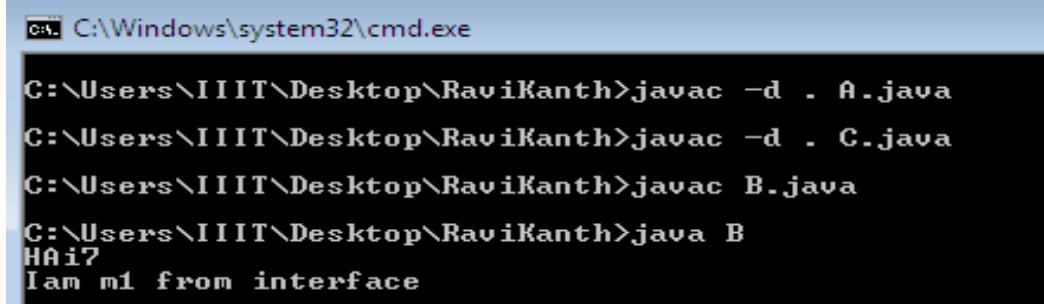
```
// Creation of User Defined Package
package OOPS;

// Creation of Class
public class A
{
    public void a(int i)
    {
        System.out.println("Hai"+i);
    }
}

package OOPS;
public interface C
{
    public void m1();
}
```

```
// Accessing User Defined Package OOPS which is
// having class A and interface C
import OOPS.A;
import OOPS.C;

class B implements C
{
    public void m1()
    {
        System.out.println("Iam m1 from interface");
    }
    public static void main(String aa[])
    {
        A a1=new A();
        a1.a(7);
        B b1=new B();
        b1.m1();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac -d . A.java
C:\Users\IIIT\Desktop\RaviKanth>javac -d . C.java
C:\Users\IIIT\Desktop\RaviKanth>javac B.java
C:\Users\IIIT\Desktop\RaviKanth>java B
Hai?
Iam m1 from interface
```

### Ex 3: Creating a sub-package JAVA in side a main package OOPS

```
// Creation of User Defined Package
package OOPS.JAVA;

// Creation of Class
public class A
{
    public void a(int i)
    {
        System.out.println("Hai"+i);
    }
}
```

```

package OOPS.JAVA;
public interface C
{
    public void m1();
}

// Accessing User Defined Package OOPS which is
// having class A and interface C
import OOPS.JAVA.A;
import OOPS.JAVA.C;

class B implements C
{
    public void m1()
    {
        System.out.println("Iam m1 from interface");
    }
    public static void main(String aa[])
    {
        A a1=new A();
        a1.a(7);
        B b1=new B();
        b1.m1();
    }
}

```



```

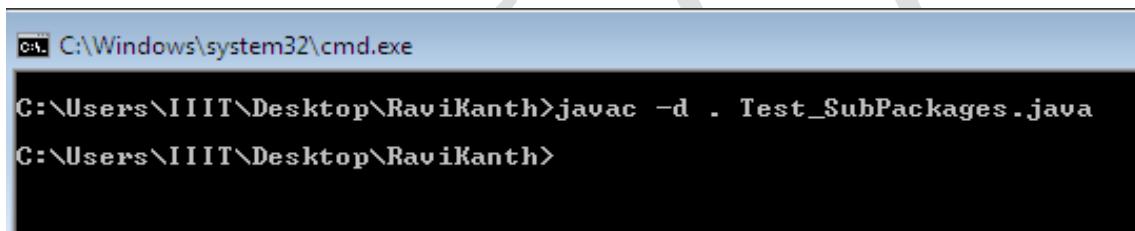
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac -d . A.java
C:\Users\IIIT\Desktop\RaviKanth>javac -d . C.java
C:\Users\IIIT\Desktop\RaviKanth>javac B.java
C:\Users\IIIT\Desktop\RaviKanth>java B
Hai?
Iam m1 from interface

```

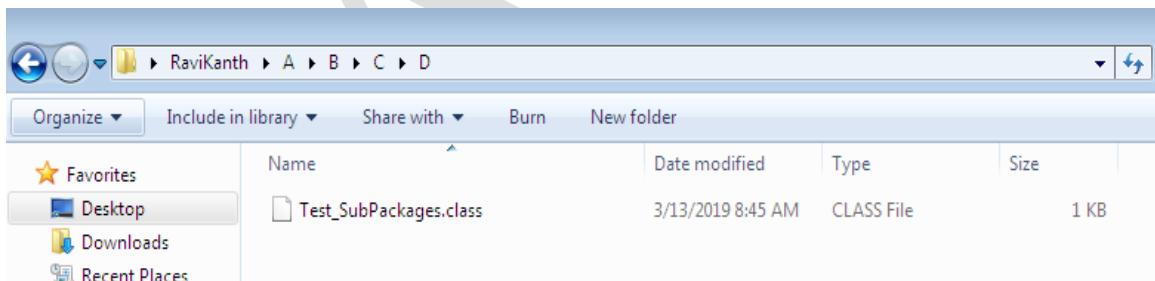
**Example over Sub Package**

```

1 // Example over sub packages in java
2 package A.B.C.D;
3 public class Test_SubPackages
4 {
5     public void dispaly()
6     {
7         System.out.println(" Lets create Sub Packages");
8     }
9 }
```

**Save the file as Test\_SubPackages.java****Compile the file as javac -d .Test\_SubPackages.java**


C:\Windows\system32\cmd.exe  
C:\Users\IIIT\Desktop\RaviKanth>javac -d . Test\_SubPackages.java  
C:\Users\IIIT\Desktop\RaviKanth>

**Now you can see the sub-folders created**

**Java compiler will place Test\_SubPackages.class file in A.B.C.D package and when you want to use this class in our application we should import it as follows:**

```
import A.B.C.D.Test_SubPackages;
```

or

```
import A.B.C.D.*;
```

**Difference between Inheritance and package**

- Inheritance concept always used to reuse the feature within the program between class to class, interface to interface and interface to class but not accessing the feature across the program.
- Package concept is to reuse the feature both within the program and across the programs between class to class, interface to interface and interface to class.

**Difference between package keyword and import keyword**

- “package” keyword is always used for creating the undefined package and placing common classes and interfaces.
- import is a keyword which is used for referring or using the classes and interfaces of a specific package.
- **Note:** Java compiler will not allow us to import same contents present in multiple packages. To overcome this we need to specify the package name and class name explicitly as follows:

**Packagename.classname**

**Ex: Pack1.Number or Pack2.Number**

**CASE STUDIES:**

1. Design a class with named as Number which contains 2 methods add\_Numbers and sub\_Numbers and place them in a package called “Calculator”. Now create another class called Calculations and import Calculator package and print the addition and subtraction of Numbers.
2. Design a package with name Pack1 which contains 2 classes 1 interface
3. Define the above interface methods in your class
4. Define one more package with the name of Pack2 and place 1 class and interface
5. Create a package content to understand the protected access modifier

**UNIT-IV****Topics to be covered:**

**Multi threading:** Creating Threads, Life of a Thread, Defining & Running Thread, Thread Methods, Thread Priority, Synchronization, Implementing runnable interface, Thread scheduling.

**Introduction to Multi-Tasking**

- A *process* is a program in execution. A process may be divided into a number of independent units known as *threads*.
- If two applications run on a computer (MS Word, MS Access), two processes are created.
- Multitasking of two or more processes is known as *process-based multitasking*.  
Multitasking of two or more threads is known as *thread-based multitasking*.
- The concept of multithreading in a programming language refers to thread-based multitasking.
- Process-based multitasking is totally controlled by the operating system.
- Thread-based multitasking can be controlled by the programmer to some extent in a program.

**1. Process-based multitasking**

**Ex:** chating onto facebook and listering to music , where two processes are running parallel

- Heavy weight , as it requires more resources
- More than one process are running in the memory address, the each process require separate memory space [ own memory address space]
- Inter process communication is expensive

**2. Thread-based multitasking**

**Ex:** text editor—doing both the tasks simultaneously like editing the test and printing the text

- Light weight, as it requires less resources
- Address space is shared
- Inter thread communication is cheap

### **Multithreading**

- Multi-threading is one of the important concept in java
- Java is a multi-threaded programming language which means we can develop multi-threaded program using java.
- A multi-threaded program consists of two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPU's
- It is a process or a program divided into two or more subprograms( process), which can be implemented at the same time in parallel
- Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program

### **Benefits of Multithreading**

1. It enables programmers to do multiple things at one time
2. Programmers can execute them in parallel which will eventually increases speed of the program execution'
3. Improved performance and concurrently
4. Simultaneous access to multiple applications'

## **Life-cycle of Thread**

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs and then dies. The following diagram shows the complete life cycle of a thread.
- The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
  - New Born State
  - Runnable state
  - Running state
  - Non-Runnable (Blocked) state
  - Terminated ( Dead State)

### **1) New Born State**

- The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

### **2) Runnable State**

- The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

### **3) Running State**

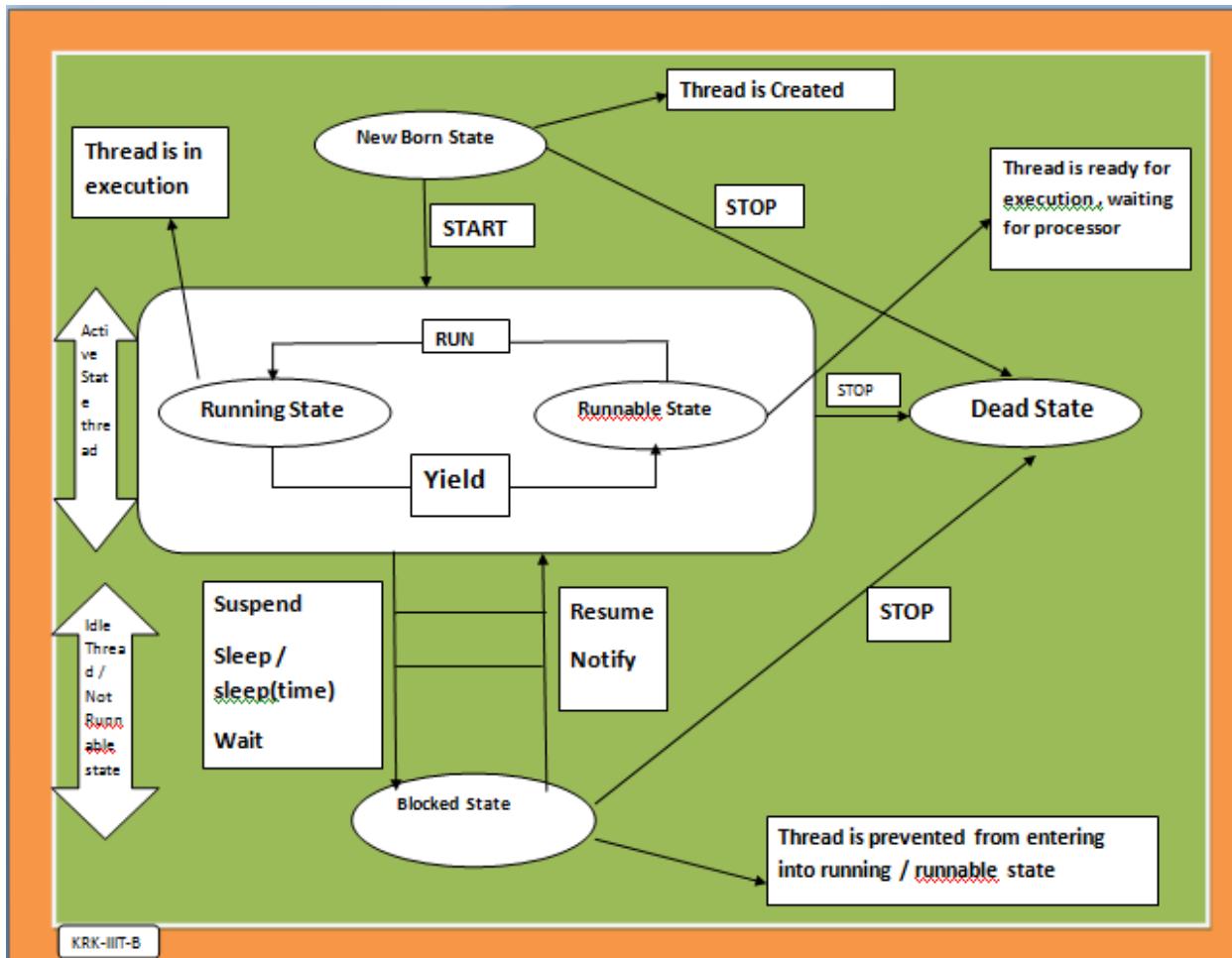
- The thread is in running state if the thread scheduler has selected it.

### **4) Non-Runnable (Blocked) State**

- This is the state when the thread is still alive, but is currently not eligible to run.

### **5) Terminated / Dead State**

- A thread is in terminated or dead state when its run() method exits.



## Creating Threads

- We can create a thread program either by :
  - By extending to Thread class
  - By implementing to Runnable interface

**Syntax:**

```
class ThreadExam extends Thread { }
```

```
class ThreadExam implements Runnable { }
```

```
// using Thread Methods
```

### Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

### Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

### Defining & Running Thread

**start()** method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

### Thread Methods

Following are the lsit of important methods in thread class:

| Method name  | Description   |
|--|---|
| <b>public void start()</b>                         | Starts the Thread in a separate path of execution, then invokes the run() method on this thread object                      |
| <b>public void run()</b>                           | If the thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object |
| <b>public final void setName(String name)</b>      | Changes the name of the Thread object. There is also a getName() method for retrieving the name                             |
| <b>public final void setPriority(int priority)</b> | Sets the priority of the Thread object. The possible values are between 1 to 10   |
| <b>public int getPriority()</b>                    | returns the priority of the thread  |
| <b>public final void</b>                           | A parameter of true denotes this Thread as a daemon   |

|  |  |
|--|--|
| <b>setDaemon(boolean on)</b>                   | Thread   |
| <b>public final void join(long millisec)</b>   | The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes |
| <b>public void interrupt()</b>                 | Interrupts this thread, causing it to continue execution if it was blocked for any reason  |
| <b>public final boolean isAlive()</b>          | Returns true if the thread is alive, which is any time after the thread had been started but before it runs to Completion  |
| <b>public static void yield()</b>              | Causes the currently running thread object to temporarily pause and allow other threads to execute   |
| <b>public static void sleep(long millisec)</b> | Causes the currently running thread to block for atleast the specified number of milliseconds  |
| <b>public static Thread currentThread()</b>    | Returns a reference to the currently running thread, which is the thread that invokes this method  |
| <b>public int getId()</b>                      | returns the id of the thread   |
| <b>public String getName()</b>                 | returns the name of the thread.  |
| <b>public void setName(String name)</b>        | changes the name of the thread   |
| <b>public boolean isDaemon()</b>               | tests if the thread is a daemon thread   |
| <b>public void suspend()</b>                   | is used to suspend the thread(deprecated)  |
| <b>public void resume()</b>                    | is used to resume the suspended thread(deprecated)   |
| <b>public void interrupt()</b>                 | interrupts the thread.   |
| <b>public void stop()</b>                      | is used to stop the thread(deprecated)   |

## Thread Priorities

- Every java thread has a priority that helps the operating system determine the order in which threads are scheduled
- Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling).
- Java thread priorities are in the range between MIN\_PRIORITY ( a constant of 1) and MAX\_PRIORITY( a constant of 10). By default, every thread is given priority NORM\_PRIORITY( a constant 5)
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads.
- However, thread priorities can not guarantee the order in which threads execute and are dependent on JVM specification that which scheduling algorithm it uses i.e., very much platform dependent
  1. public static int MIN\_PRIORITY
  2. public static int NORM\_PRIORITY
  3. public static int MAX\_PRIORITY

Default priority of a thread is 5 (NORM\_PRIORITY).

The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.

## Daemon Thread

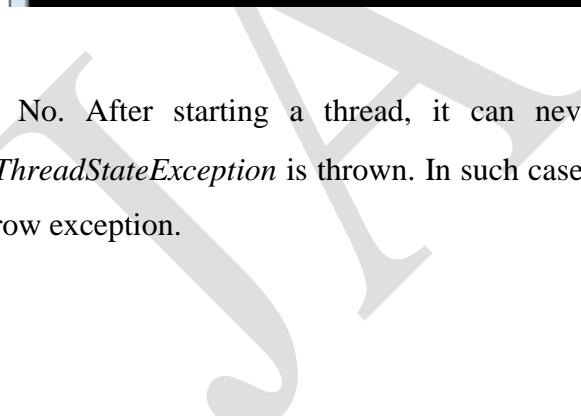
**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

It is a low priority thread.

**Example 1:**

```
// Creating a ThreadExample Program using Thread Class
public class ThreadExample extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        ThreadExample thread1=new ThreadExample();
        thread1.start();
    }
}
```

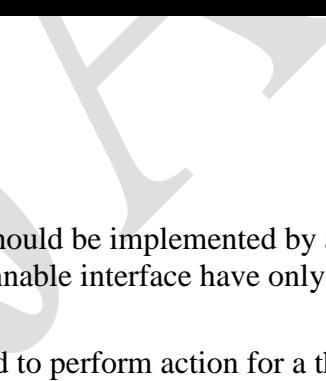


```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample
Thread is Running
```

**Note:** No. After starting a thread, it can never be started again. If you do so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

**Example2:**

```
// Creating two threads using Thread Class
public class ThreadExample2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        ThreadExample2 thread1=new ThreadExample2();
        ThreadExample2 thread2=new ThreadExample2();
        thread1.start();
        thread2.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample2.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample2
Thread is Running
Thread is Running
```

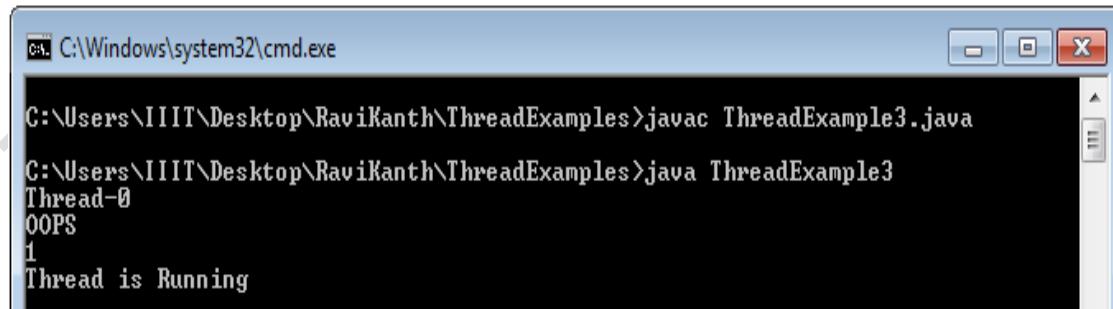
**Example 3:****Runnable interface:**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

**public void run():** is used to perform action for a thread.

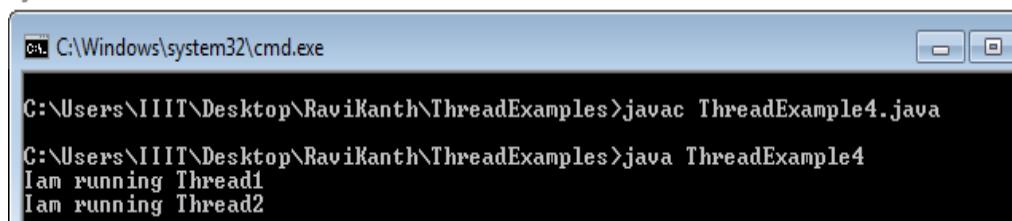
**Note:** If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitly create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

```
// Creating a thread using Runnable interface
public class ThreadExample3 implements Runnable
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        ThreadExample3 thread1=new ThreadExample3();
        Thread t1=new Thread(thread1);
        System.out.println(t1.getName());
        t1.setName("OOPS");
        System.out.println(t1.getName());
        t1.setPriority(Thread.MIN_PRIORITY); // Thread.MAX_PRIORITY
        System.out.println(t1.getPriority());
        t1.start();
    }
}
```



**Example 4:**

```
// Running multiple threads extending Thread class
class Thread1 extends Thread
{
    public void run()
    {
        System.out.println("Iam running Thread1");
    }
}
class Thread2 extends Thread
{
    public void run()
    {
        System.out.println("Iam running Thread2");
    }
}
public class ThreadExample4
{
    public static void main(String args[])
    {
        Thread1 thread1=new Thread1();
        Thread2 thread2=new Thread2();
        thread1.start();
        thread2.start();
    }
}
```

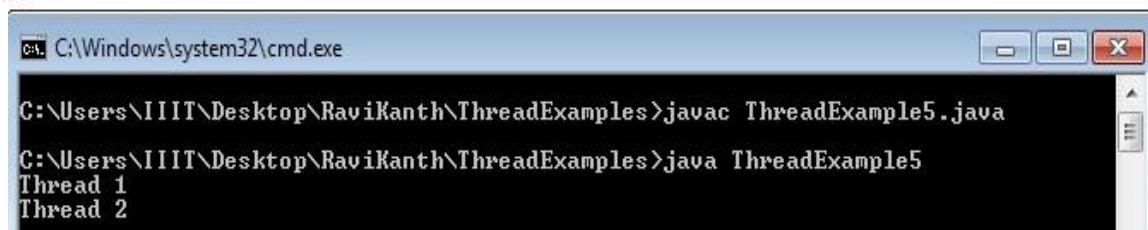


The screenshot shows a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The user has navigated to the directory 'C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples' and run the command 'javac ThreadExample4.java'. After compilation, they run the program with 'java ThreadExample4', which outputs the text 'Iam running Thread1' and 'Iam running Thread2' to the console.

```
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample4.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample4
Iam running Thread1
Iam running Thread2
```

**Example 5:**

```
// By creating instances of Thread class directly
public class ThreadExample5
{
    public static void main(String args[])
    {
        Thread t1=new Thread()
        {
            public void run()
            {
                System.out.println("Thread 1");
            }
        };
        Thread t2=new Thread()
        {
            public void run()
            {
                System.out.println("Thread 2");
            }
        };
        t1.start();
        t2.start();
    }
}
```



The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The command 'javac ThreadExample5.java' is entered and executed, followed by 'java ThreadExample5'. The output displays 'Thread 1' and 'Thread 2' on separate lines, indicating the parallel execution of the two threads.

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample5.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample5
Thread 1
Thread 2
```

**Example 6:**

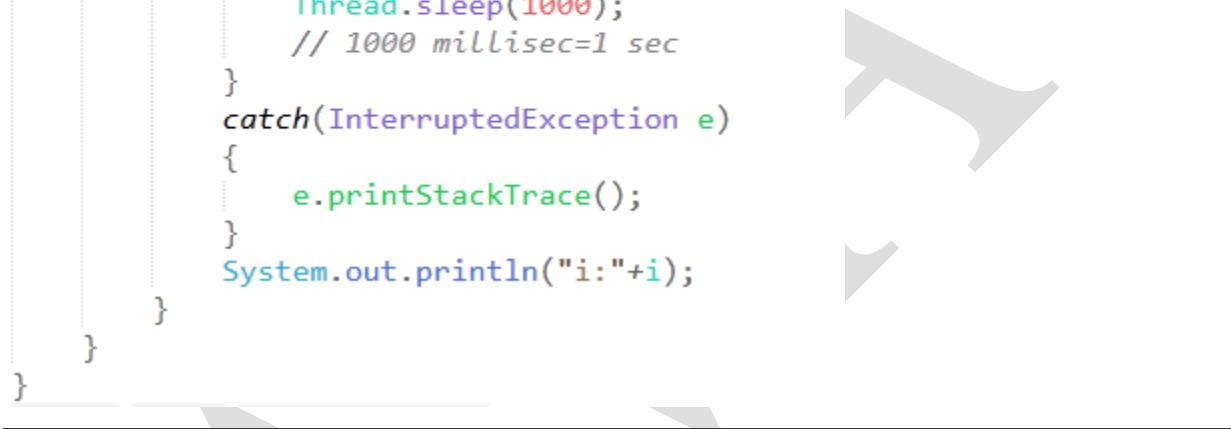
```
// Thread class sleep() method
// make a thread sleep for a period of time
class Thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            try
            {
                Thread.sleep(1000);
                // 1000 millisec=1 sec
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
            System.out.println("i:"+i);
        }
    }
}
public class ThreadExample6
{
    public static void main(String args[])
    {
        Thread1 thread1=new Thread1();
        Thread1 thread2=new Thread1();
        thread1.start();
        thread2.start();
        //Both the threads are having same priority
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample6.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample6
i:0
i:0
i:1
i:1
i:2
i:2
i:3
i:3
i:4
i:4
i:5
i:5
```

JA  
VA

**Example 7:**

```
// Thread class sleep() method
// make a thread sleep for a period of time
class Thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            try
            {
                Thread.sleep(1000);
                // 1000 millisec=1 sec
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
            System.out.println("i:"+i);
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample7.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample7
i:0
i:1
i:2
i:3
i:0
i:0
i:1
i:1
i:4
i:1
i:2
i:5
i:2
i:3
i:3
i:4
i:4
i:5
i:5
i:5
```

**Example 8:**

```
// Example over currentThread, getName, sleep methods
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("thread Name is:"+Thread.currentThread().getName()+"value:"+i);
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
public class ThreadExample8
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample8.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample8
thread Name is:Thread-0value:0
thread Name is:Thread-1value:0
thread Name is:Thread-1value:1
thread Name is:Thread-0value:1
thread Name is:Thread-0value:2
thread Name is:Thread-1value:2
thread Name is:Thread-0value:3
thread Name is:Thread-1value:3
thread Name is:Thread-0value:4
thread Name is:Thread-1value:4
thread Name is:Thread-0value:5
thread Name is:Thread-1value:5
thread Name is:Thread-0value:6
thread Name is:Thread-1value:6
thread Name is:Thread-1value:7
thread Name is:Thread-0value:7
thread Name is:Thread-1value:8
thread Name is:Thread-0value:8
thread Name is:Thread-0value:9
thread Name is:Thread-1value:9
```

**Explanation:**

Sleep method makes the current thread to go to sleep for the pre-defined time we are passing to the method those many milliseconds the thread is going to sleep. Once that time get elapsed the sleep thread will come to running state

**UNIT-IV****Topics to be covered:**

**Multi threading:** Creating Threads, Life of a Thread, Defining & Running Thread, Thread Methods, Thread Priority, Synchronization, Implementing runnable interface, Thread scheduling.

**Introduction to Multi-Tasking**

- A *process* is a program in execution. A process may be divided into a number of independent units known as *threads*.
- If two applications run on a computer (MS Word, MS Access), two processes are created.
- Multitasking of two or more processes is known as *process-based multitasking*.  
Multitasking of two or more threads is known as *thread-based multitasking*.
- The concept of multithreading in a programming language refers to thread-based multitasking.
- Process-based multitasking is totally controlled by the operating system.
- Thread-based multitasking can be controlled by the programmer to some extent in a program.

**1. Process-based multitasking**

**Ex:** chating onto facebook and listering to music , where two processes are running parallel

- Heavy weight , as it requires more resources
- More than one process are running in the memory address, the each process require separate memory space [ own memory address space]
- Inter process communication is expensive

**2. Thread-based multitasking**

**Ex:** text editor—doing both the tasks simultaneously like editing the test and printing the text

- Light weight, as it requires less resources
- Address space is shared
- Inter thread communication is cheap

### **Multithreading**

- Multi-threading is one of the important concept in java
- Java is a multi-threaded programming language which means we can develop multi-threaded program using java.
- A multi-threaded program consists of two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPU's
- It is a process or a program divided into two or more subprograms( process), which can be implemented at the same time in parallel
- Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program

### **Benefits of Multithreading**

1. It enables programmers to do multiple things at one time
2. Programmers can execute them in parallel which will eventually increases speed of the program execution'
3. Improved performance and concurrently
4. Simultaneous access to multiple applications'

## **Life-cycle of Thread**

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs and then dies. The following diagram shows the complete life cycle of a thread.
- The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
  - New Born State
  - Runnable state
  - Running state
  - Non-Runnable (Blocked) state
  - Terminated ( Dead State)

### **1) New Born State**

- The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

### **2) Runnable State**

- The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

### **3) Running State**

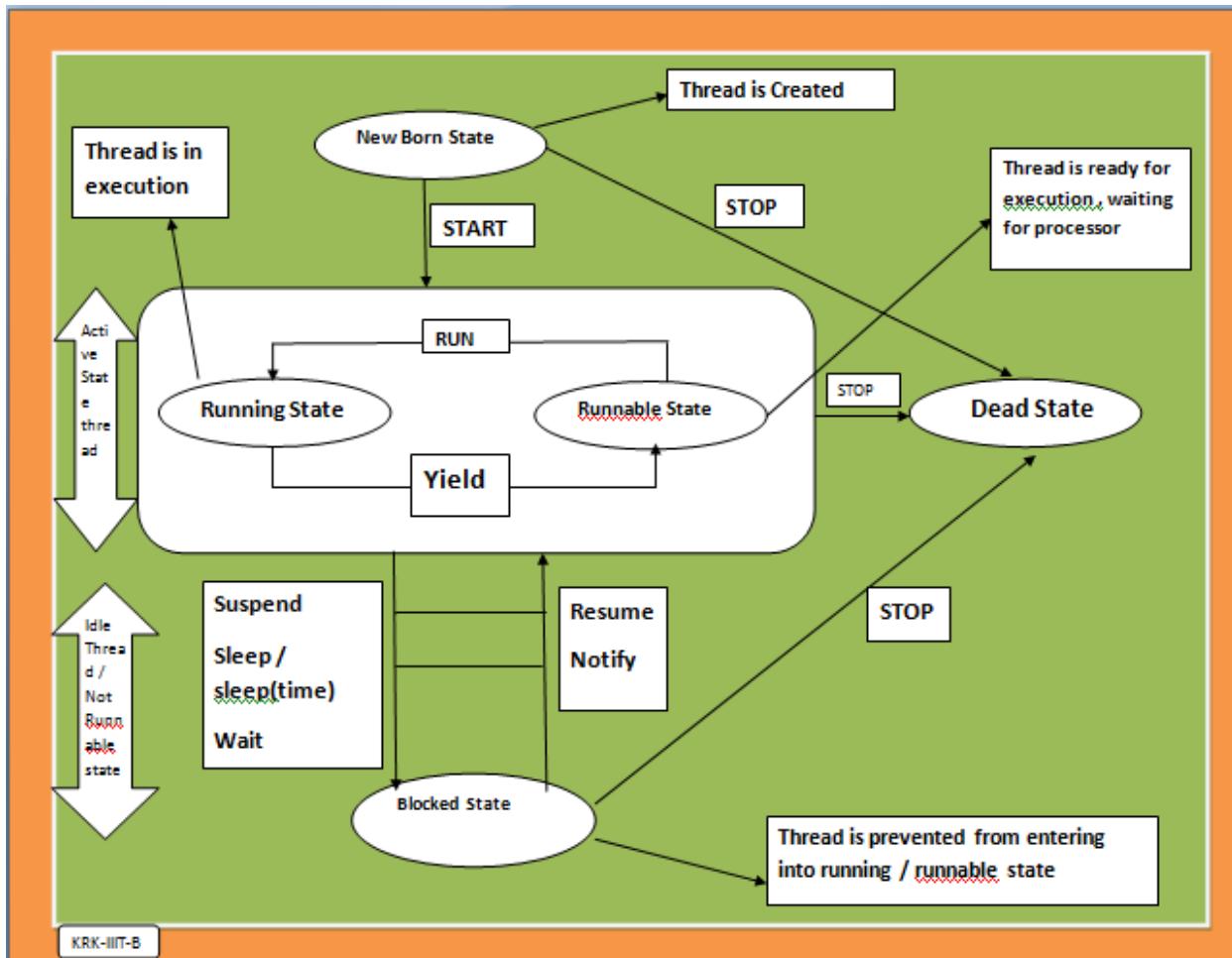
- The thread is in running state if the thread scheduler has selected it.

### **4) Non-Runnable (Blocked) State**

- This is the state when the thread is still alive, but is currently not eligible to run.

### **5) Terminated / Dead State**

- A thread is in terminated or dead state when its run() method exits.



## Creating Threads

- We can create a thread program either by :
  - By extending to Thread class
  - By implementing to Runnable interface

**Syntax:**

```
class ThreadExam extends Thread { }
```

```
class ThreadExam implements Runnable { }
```

```
// using Thread Methods
```

### Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

### Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

### Defining & Running Thread

**start()** method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

### Thread Methods

Following are the lsit of important methods in thread class:

| Method name  | Description   |
|--|---|
| <b>public void start()</b>                         | Starts the Thread in a separate path of execution, then invokes the run() method on this thread object                      |
| <b>public void run()</b>                           | If the thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object |
| <b>public final void setName(String name)</b>      | Changes the name of the Thread object. There is also a getName() method for retrieving the name                             |
| <b>public final void setPriority(int priority)</b> | Sets the priority of the Thread object. The possible values are between 1 to 10   |
| <b>public int getPriority()</b>                    | returns the priority of the thread  |
| <b>public final void</b>                           | A parameter of true denotes this Thread as a daemon   |

|  |  |
|--|--|
| <b>setDaemon(boolean on)</b>                   | Thread   |
| <b>public final void join(long millisec)</b>   | The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes |
| <b>public void interrupt()</b>                 | Interrupts this thread, causing it to continue execution if it was blocked for any reason  |
| <b>public final boolean isAlive()</b>          | Returns true if the thread is alive, which is any time after the thread had been started but before it runs to Completion  |
| <b>public static void yield()</b>              | Causes the currently running thread object to temporarily pause and allow other threads to execute   |
| <b>public static void sleep(long millisec)</b> | Causes the currently running thread to block for atleast the specified number of milliseconds  |
| <b>public static Thread currentThread()</b>    | Returns a reference to the currently running thread, which is the thread that invokes this method  |
| <b>public int getId()</b>                      | returns the id of the thread   |
| <b>public String getName()</b>                 | returns the name of the thread.  |
| <b>public void setName(String name)</b>        | changes the name of the thread   |
| <b>public boolean isDaemon()</b>               | tests if the thread is a daemon thread   |
| <b>public void suspend()</b>                   | is used to suspend the thread(deprecated)  |
| <b>public void resume()</b>                    | is used to resume the suspended thread(deprecated)   |
| <b>public void interrupt()</b>                 | interrupts the thread.   |
| <b>public void stop()</b>                      | is used to stop the thread(deprecated)   |

## Thread Priorities

- Every java thread has a priority that helps the operating system determine the order in which threads are scheduled
- Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling).
- Java thread priorities are in the range between MIN\_PRIORITY ( a constant of 1) and MAX\_PRIORITY( a constant of 10). By default, every thread is given priority NORM\_PRIORITY( a constant 5)
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads.
- However, thread priorities can not guarantee the order in which threads execute and are dependent on JVM specification that which scheduling algorithm it uses i.e., very much platform dependent
  1. public static int MIN\_PRIORITY
  2. public static int NORM\_PRIORITY
  3. public static int MAX\_PRIORITY

Default priority of a thread is 5 (NORM\_PRIORITY).

The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.

## Daemon Thread

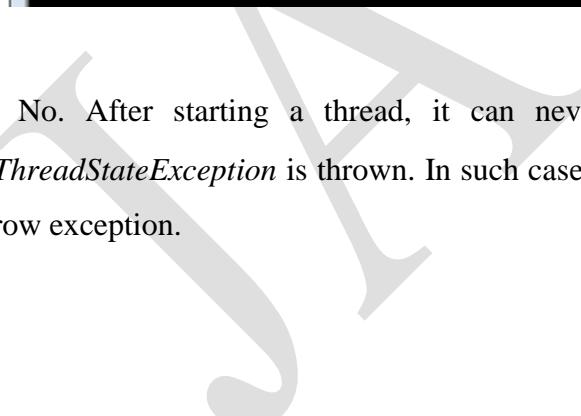
**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

It is a low priority thread.

**Example 1:**

```
// Creating a ThreadExample Program using Thread Class
public class ThreadExample extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        ThreadExample thread1=new ThreadExample();
        thread1.start();
    }
}
```

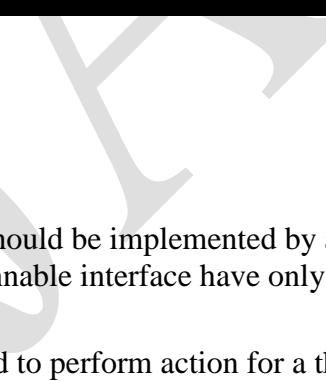


```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample
Thread is Running
```

**Note:** No. After starting a thread, it can never be started again. If you do so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

**Example2:**

```
// Creating two threads using Thread Class
public class ThreadExample2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        ThreadExample2 thread1=new ThreadExample2();
        ThreadExample2 thread2=new ThreadExample2();
        thread1.start();
        thread2.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample2.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample2
Thread is Running
Thread is Running
```

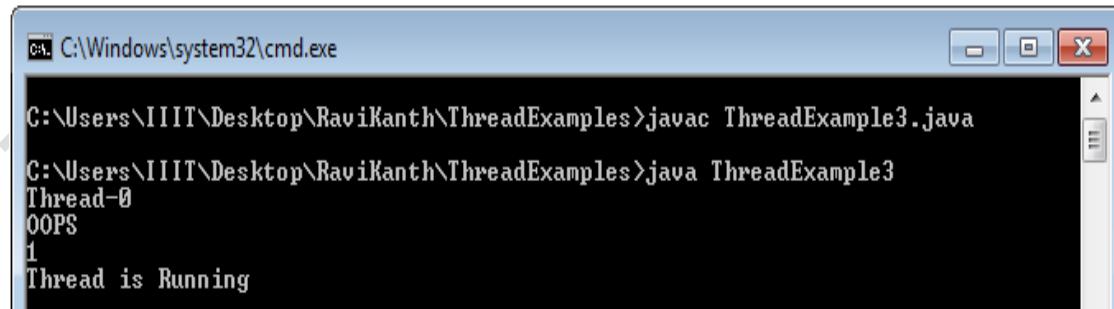
**Example 3:****Runnable interface:**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

**public void run():** is used to perform action for a thread.

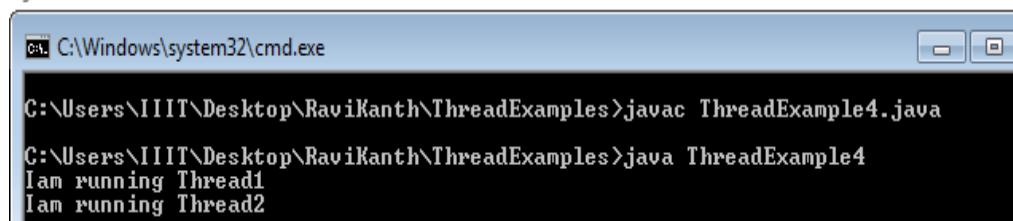
**Note:** If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitly create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

```
// Creating a thread using Runnable interface
public class ThreadExample3 implements Runnable
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        ThreadExample3 thread1=new ThreadExample3();
        Thread t1=new Thread(thread1);
        System.out.println(t1.getName());
        t1.setName("OOPS");
        System.out.println(t1.getName());
        t1.setPriority(Thread.MIN_PRIORITY); // Thread.MAX_PRIORITY
        System.out.println(t1.getPriority());
        t1.start();
    }
}
```



**Example 4:**

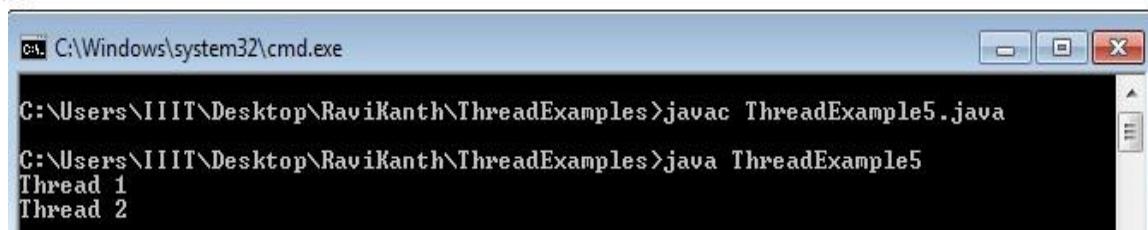
```
// Running multiple threads extending Thread class
class Thread1 extends Thread
{
    public void run()
    {
        System.out.println("Iam running Thread1");
    }
}
class Thread2 extends Thread
{
    public void run()
    {
        System.out.println("Iam running Thread2");
    }
}
public class ThreadExample4
{
    public static void main(String args[])
    {
        Thread1 thread1=new Thread1();
        Thread2 thread2=new Thread2();
        thread1.start();
        thread2.start();
    }
}
```



```
C:\ C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample4.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample4
Iam running Thread1
Iam running Thread2
```

**Example 5:**

```
// By creating instances of Thread class directly
public class ThreadExample5
{
    public static void main(String args[])
    {
        Thread t1=new Thread()
        {
            public void run()
            {
                System.out.println("Thread 1");
            }
        };
        Thread t2=new Thread()
        {
            public void run()
            {
                System.out.println("Thread 2");
            }
        };
        t1.start();
        t2.start();
    }
}
```



The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The command 'javac ThreadExample5.java' is entered and executed. The output shows two lines of text: 'Thread 1' and 'Thread 2', indicating that the two threads are running simultaneously.

```
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample5.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample5
Thread 1
Thread 2
```

**Example 6:**

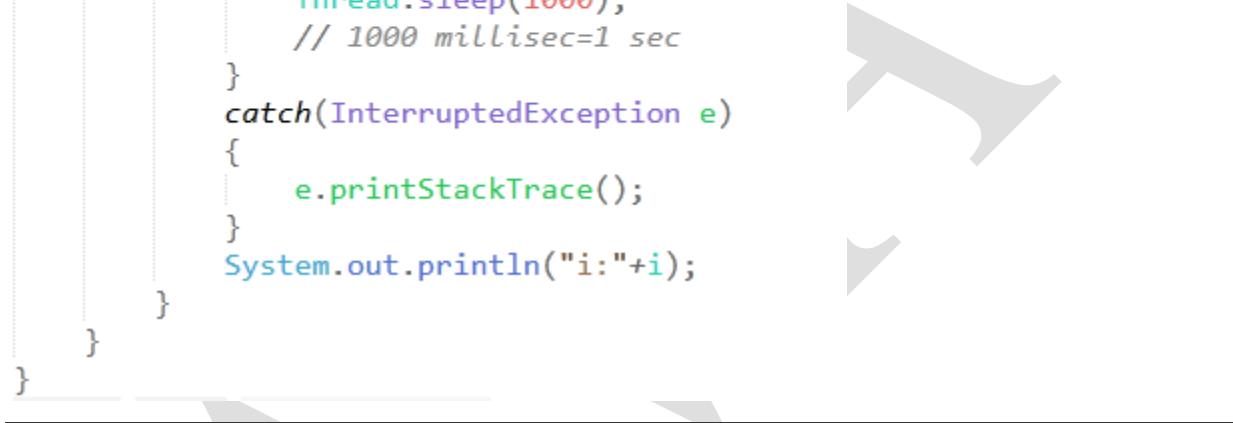
```
// Thread class sleep() method
// make a thread sleep for a period of time
class Thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            try
            {
                Thread.sleep(1000);
                // 1000 millisec=1 sec
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
            System.out.println("i:"+i);
        }
    }
}
public class ThreadExample6
{
    public static void main(String args[])
    {
        Thread1 thread1=new Thread1();
        Thread1 thread2=new Thread1();
        thread1.start();
        thread2.start();
        //Both the threads are having same priority
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample6.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample6
i:0
i:0
i:1
i:1
i:2
i:2
i:3
i:3
i:4
i:4
i:5
i:5
```

JA  
VA

**Example 7:**

```
// Thread class sleep() method
// make a thread sleep for a period of time
class Thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            try
            {
                Thread.sleep(1000);
                // 1000 millisec=1 sec
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
            System.out.println("i:"+i);
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample7.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample7
i:0
i:1
i:2
i:3
i:0
i:0
i:1
i:1
i:4
i:1
i:2
i:5
i:2
i:3
i:3
i:4
i:4
i:5
i:5
i:5
```

**Example 8:**

```
// Example over currentThread, getName, sleep methods
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("thread Name is:"+Thread.currentThread().getName()+"value:"+i);
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
public class ThreadExample8
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample8.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample8
thread Name is:Thread-0value:0
thread Name is:Thread-1value:0
thread Name is:Thread-1value:1
thread Name is:Thread-0value:1
thread Name is:Thread-0value:2
thread Name is:Thread-1value:2
thread Name is:Thread-0value:3
thread Name is:Thread-1value:3
thread Name is:Thread-0value:4
thread Name is:Thread-1value:4
thread Name is:Thread-0value:5
thread Name is:Thread-1value:5
thread Name is:Thread-0value:6
thread Name is:Thread-1value:6
thread Name is:Thread-1value:7
thread Name is:Thread-0value:7
thread Name is:Thread-1value:8
thread Name is:Thread-0value:8
thread Name is:Thread-0value:9
thread Name is:Thread-1value:9
```

**Explanation:**

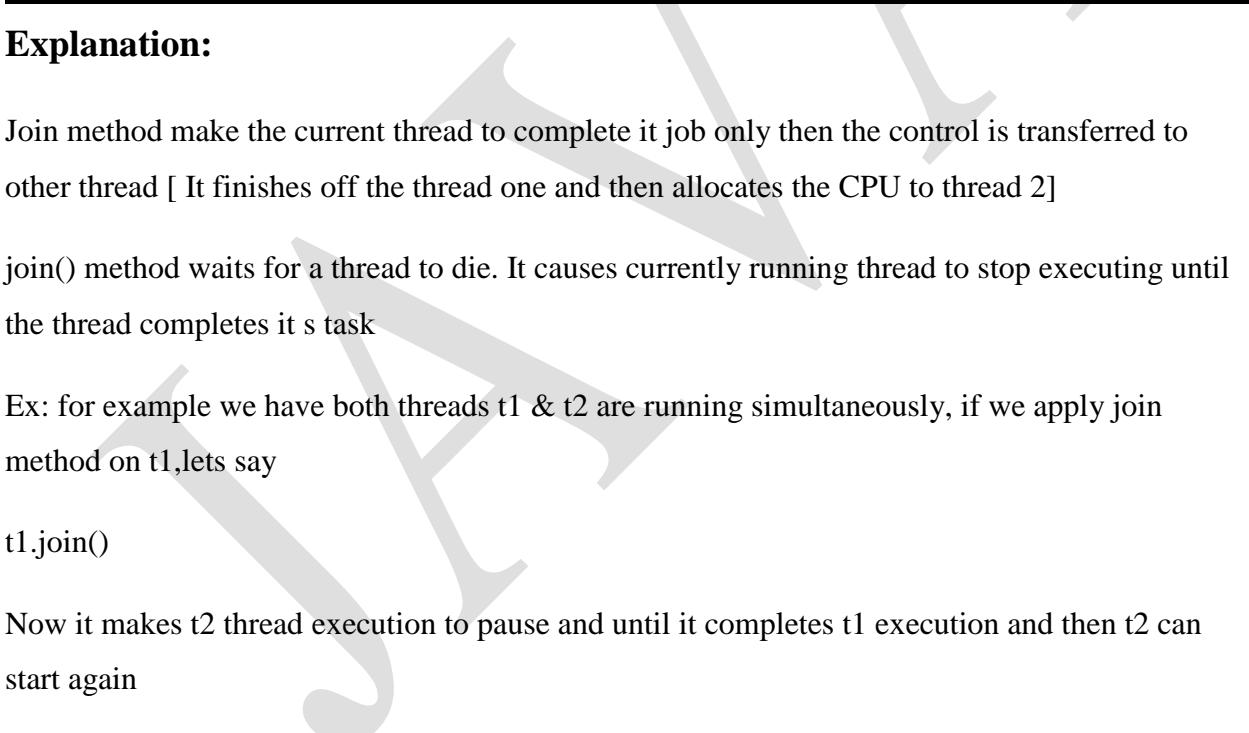
Sleep method makes the current thread to go to sleep for the pre-defined time we are passing to the method those many milliseconds the thread is going to sleep. Once that time get elapsed the sleep thread will come to running state

**Example 9:**

```
// Example over thread.join() methods
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("thread Name is:"+Thread.currentThread().getName()+"value:"+i);
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
public class ThreadExample9
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        try
        {
            t1.join();
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
        t2.start();
    }
}
```

---





```
C:\> C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample9.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample9
thread Name is:Thread-0value:0
thread Name is:Thread-0value:1
thread Name is:Thread-0value:2
thread Name is:Thread-0value:3
thread Name is:Thread-0value:4
thread Name is:Thread-0value:5
thread Name is:Thread-0value:6
thread Name is:Thread-0value:7
thread Name is:Thread-0value:8
thread Name is:Thread-0value:9
thread Name is:Thread-1value:0
thread Name is:Thread-1value:1
thread Name is:Thread-1value:2
thread Name is:Thread-1value:3
thread Name is:Thread-1value:4
thread Name is:Thread-1value:5
thread Name is:Thread-1value:6
thread Name is:Thread-1value:7
thread Name is:Thread-1value:8
thread Name is:Thread-1value:9
```

### Explanation:

Join method make the current thread to complete it job only then the control is transferred to other thread [ It finishes off the thread one and then allocates the CPU to thread 2]

join() method waits for a thread to die. It causes currently running thread to stop executing until the thread completes its task

Ex: for example we have both threads t1 & t2 are running simultaneously, if we apply join method on t1,lets say

```
t1.join()
```

Now it makes t2 thread execution to pause and until it completes t1 execution and then t2 can start again

**Example 10:**

Use of Yield() method

Yield() method causes to pause current executing thread for giving the chance to remaining waiting threads same priority. If there are no waiting threads, have the priority. We cannot predict the output, it depends on the processor

**Syntax:** public static void yield()

```
// Example over Thread class yield() methods
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("thread Name is:"+Thread.currentThread().getName()+"value:"+i);
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
public class ThreadExample10
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t1.yield();
        t2.start();
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample10.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample10
thread Name is:Thread-0value:0
thread Name is:Thread-1value:0
thread Name is:Thread-0value:1
thread Name is:Thread-1value:1
thread Name is:Thread-0value:2
thread Name is:Thread-1value:2
thread Name is:Thread-0value:3
thread Name is:Thread-1value:3
thread Name is:Thread-0value:3
thread Name is:Thread-1value:4
thread Name is:Thread-0value:4
thread Name is:Thread-0value:5
thread Name is:Thread-1value:5
thread Name is:Thread-1value:5
thread Name is:Thread-1value:6
thread Name is:Thread-0value:6
thread Name is:Thread-1value:7
thread Name is:Thread-0value:7
thread Name is:Thread-1value:8
thread Name is:Thread-0value:8
thread Name is:Thread-0value:9
thread Name is:Thread-1value:9
```

### Explanation:

yield method will pause the current thread and makes the other thread to run and if there is no other thread running then the current thread itself is executed

## Example 11

Try out with MAX\_PRIORITY, MIN\_PRIORITY, NORM\_PRIORITY

```
// Example over setPriority() method
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("thread Name is:"+
                Thread.currentThread().getName()+"value:"+i+
                "Priority"+Thread.currentThread().getPriority());
        }
    }
}
public class ThreadExample11
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>javac ThreadExample11.java
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample11
thread Name is:Thread-0value:0Priority5
thread Name is:Thread-1value:0Priority5
thread Name is:Thread-0value:1Priority5
thread Name is:Thread-1value:1Priority5
thread Name is:Thread-0value:2Priority5
thread Name is:Thread-1value:2Priority5
thread Name is:Thread-0value:3Priority5
thread Name is:Thread-1value:3Priority5
thread Name is:Thread-0value:4Priority5
thread Name is:Thread-1value:4Priority5
thread Name is:Thread-0value:5Priority5
thread Name is:Thread-1value:5Priority5
thread Name is:Thread-0value:6Priority5
thread Name is:Thread-1value:6Priority5
thread Name is:Thread-0value:7Priority5
thread Name is:Thread-0value:7Priority5
thread Name is:Thread-1value:8Priority5
thread Name is:Thread-0value:8Priority5
thread Name is:Thread-1value:9Priority5
thread Name is:Thread-0value:9Priority5
```

### Explanation:

Set priority method, if we have many threads running in the queue and if you want a specific thread to run , we can set the priority. Thread scheduler will pick the tread an execute them in priority

## Example 12

Now let us set the priority based on our requirement by setting maximum priority to current thread

```
// Example over setPriority() method
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("thread Name is:"+
                Thread.currentThread().getName()+"value:"+i+
                "Priority"+Thread.currentThread().getPriority());
        }
    }
}
public class ThreadExample12
{
    public static void main(String args[])
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.setPriority(Thread.MAX_PRIORITY);
        t1.start();
        t2.start();
    }
}
```

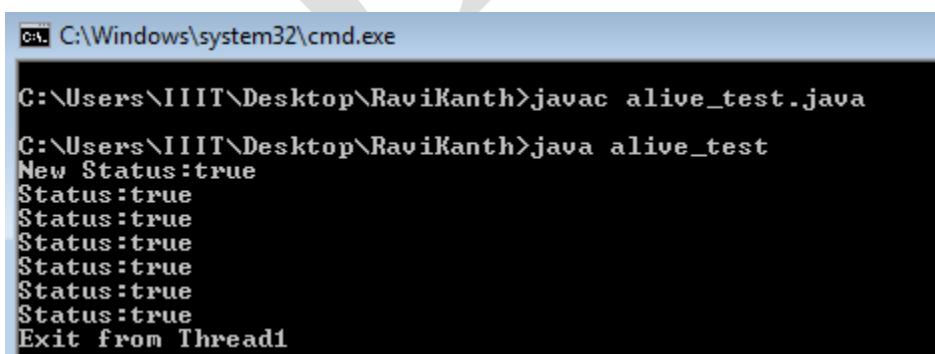
```
C:\Users\IIIT\Desktop\RaviKanth\ThreadExamples>java ThreadExample12
thread Name is:Thread-0value:0Priority10
thread Name is:Thread-0value:1Priority10
thread Name is:Thread-0value:2Priority10
thread Name is:Thread-0value:3Priority10
thread Name is:Thread-0value:4Priority10
thread Name is:Thread-0value:5Priority10
thread Name is:Thread-0value:6Priority10
thread Name is:Thread-0value:7Priority10
thread Name is:Thread-0value:8Priority10
thread Name is:Thread-0value:9Priority10
thread Name is:Thread-1value:0Priority5
thread Name is:Thread-1value:1Priority5
thread Name is:Thread-1value:2Priority5
thread Name is:Thread-1value:3Priority5
thread Name is:Thread-1value:4Priority5
thread Name is:Thread-1value:5Priority5
thread Name is:Thread-1value:6Priority5
thread Name is:Thread-1value:7Priority5
thread Name is:Thread-1value:8Priority5
thread Name is:Thread-1value:9Priority5
```

**Example 13****Use of isAlive( ) method**

`isAlive()` method tests the status of current thread whether it is alive or not. It returns True if it is alive else false

**syntax:** `public final boolean isAlive()`

```
class thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            System.out.println("Status:"+isAlive());
        }
        System.out.println("Exit from Thread1");
    }
}
class alive_test
{
    public static void main(String args[])
    {
        thread1 t1=new thread1();
        t1.start();
        System.out.println("New Status:"+t1.isAlive());
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac alive_test.java
C:\Users\IIIT\Desktop\RaviKanth>java alive_test
New Status:true
Status:true
Status:true
Status:true
Status:true
Status:true
Status:true
Exit from Thread1
```

**Example 14****Ex: Example on join method**

```

class thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            System.out.println("Status:"+isAlive());
        }
        System.out.println("Exit from Thread1");
    }
}
class alive_test1
{
    public static void main(String args[])
    {
        thread1 t1=new thread1();
        t1.start();
        try
        {
            t1.join();
        }
        catch(Exception e)
        {
        }
        System.out.println("New Status:"+t1.isAlive());
    }
}

```




```

C:\Windows\system32\cmd.exe

C:\Users\IIIT\Desktop\RaviKanth>javac alive_test1.java
C:\Users\IIIT\Desktop\RaviKanth>java alive_test1
Status:true
Status:true
Status:true
Status:true
Status:true
Status:true
Exit from Thread1
New Status:false

```

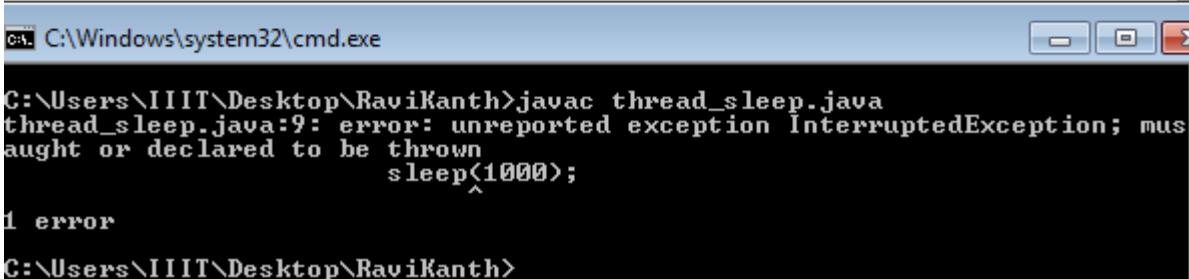
## Example 15

### Use of Sleep method:

It causes the currently running thread to block for atleast the specified number of milliseconds

We need to handle exception while using sleep() method

```
class thread_sleep extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            if(i==2)
                sleep(1000);
            System.out.println("A"+i);
        }
    }
    public static void main(String args[])
    {
        thread_sleep a=new thread_sleep();
        a.start();
    }
}
```



C:\Windows\system32\cmd.exe

C:\Users\IIIT\Desktop\RaviKanth>javac thread\_sleep.java  
thread\_sleep.java:9: error: unreported exception InterruptedException; must  
be caught or declared to be thrown  
 sleep^<1000>;  
1 error  
C:\Users\IIIT\Desktop\RaviKanth>

## Example 16

Make use of try catch blocks

```
class thread_sleep1 extends Thread
{
    int i;
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            try
            {
                if(i==2)
                    sleep(1000);
            }
            catch(Exception e)
            {
            }
            System.out.println("A"+i);
        }
    }
    public static void main(String args[])
    {
        thread_sleep1 a=new thread_sleep1();
        a.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac thread_sleep1.java
C:\Users\IIIT\Desktop\RaviKanth>java thread_sleep1
A1
A2
A3
A4
A5
C:\Users\IIIT\Desktop\RaviKanth>
```

**Example 17**

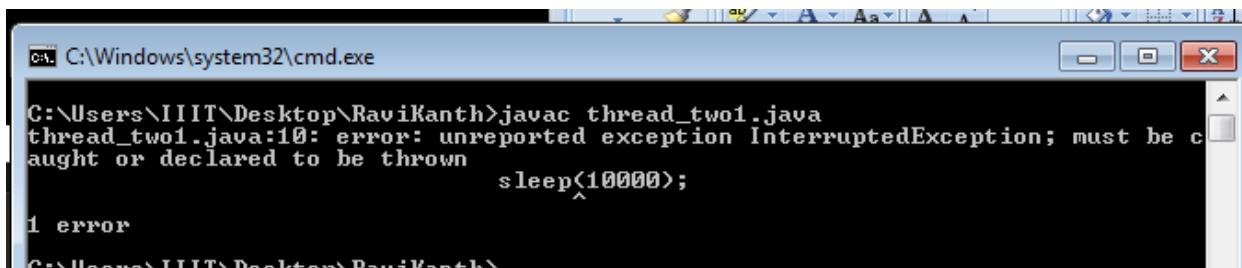
```
// Threading with out sleep method
class thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            System.out.println("Thread 1"+i);
        }
        System.out.println("Exit Thread1");
    }
}
class thread2 extends Thread
{
    public void run()
    {
        for(int j=0;j<=5;j++)
        {
            System.out.println("Thread 2"+j);
        }
        System.out.println("Exit Thread2");
    }
}
class thread_two
{
    public static void main(String args[])
    {
        thread1 t1=new thread1();
        thread2 t2=new thread2();
        t1.start();
        t2.start();
    }
}
```

```
C:\Users\IIIT\Desktop\RaviKanth>java thread_two
Thread 10
Thread 11
Thread 12
Thread 13
Thread 14
Thread 15
Exit Thread1
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Exit Thread2
```

**Example 18****// Using thread sleep method**

```
// with using sleep method
class thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            if(i==2)
                sleep(10000);
            System.out.println("Thread 1"+i);
        }
        System.out.println("Exit Thread1");
    }
}
class thread2 extends Thread
{
    public void run()
    {
        for(int j=0;j<=5;j++)
        {
            System.out.println("Thread 2"+j);
        }
        System.out.println("Exit Thread2");
    }
}
class thread_two1
{
    public static void main(String args[])
    {
        thread1 t1=new thread1();
        thread2 t2=new thread2();
        t1.start();
        t2.start();
    }
}
```

We get exception in output:



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac thread_two1.java
thread_two1.java:10: error: unreported exception InterruptedException; must be caught or declared to be thrown
        sleep(10000);
               ^
1 error
C:\Users\IIIT\Desktop\RaviKanth>
```

```
// Threading with yield method
class thread1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            if(i==2)
                yield();
            System.out.println("thread1"+i);
        }
        System.out.println("Exit Thread1");
    }
}
class thread2 extends Thread
{
    public void run()
    {
        for(int j=0;j<=5;j++)
        {
            System.out.println("Thread 2"+j);
        }
        System.out.println("Exit Thread2");
    }
}
class thread_yield
{
    public static void main(String args[])
    {
        thread1 t1=new thread1();
        thread2 t2=new thread2();
        t1.start();
        t2.start();
    }
}
```

```
C:\Users\IIIT\Desktop\RaviKanth>java thread_yield
thread10
thread11
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Exit Thread2
thread12
thread13
thread14
thread15
Exit Thread1
```

## Thread Scheduler

- **Thread scheduler** in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

**Note:** Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.

Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks.

The scheduler then determines which task should execute next, based on priority and other factors.

## Synchronization in Java

- Synchronization in java is the capability *to control the access of multiple threads to any shared resource.*
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

### Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

### Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
  1. Synchronized method.
  2. Synchronized block.
  3. static synchronization.
2. Cooperation (Inter-thread communication in java)

### Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

\*\*\*\*\*

### **Thread Priority:**

Every thread in Java has some priority. It may be default priority generated by JVM or customized priority provided by Programmer.

The valid range of Thread Priorities is 1 to 10

Where 1 is MIN\_Priority and 10 is MAX\_Priority

Thread class defines the following constants to represent some standard priorities

**Thread.MIN\_PRIORITY=1**

**Thread.NORM\_PRIORITY=5**

**Thread.MAX\_PRIORITY=10**

**Who is going to use thread priority and when do we use them ?**

**Explanation:**

**Thread Scheduler will use priorities while allocating processor**

**The thread which is having highest priority will get chance first**

- If two threads having same priority then we cannot expect exact execution order, it depends on Thread Scheduler [ Round Robin or First come first serve or Shortest Job First or so on]

Thread class defines the following methods to get and set the priority of a thread

Public final int getPriority()

Public final void setPriority(int p)

Allowed values range is 1 to 10

Otherwise Runtime Exception: illegal argument exception

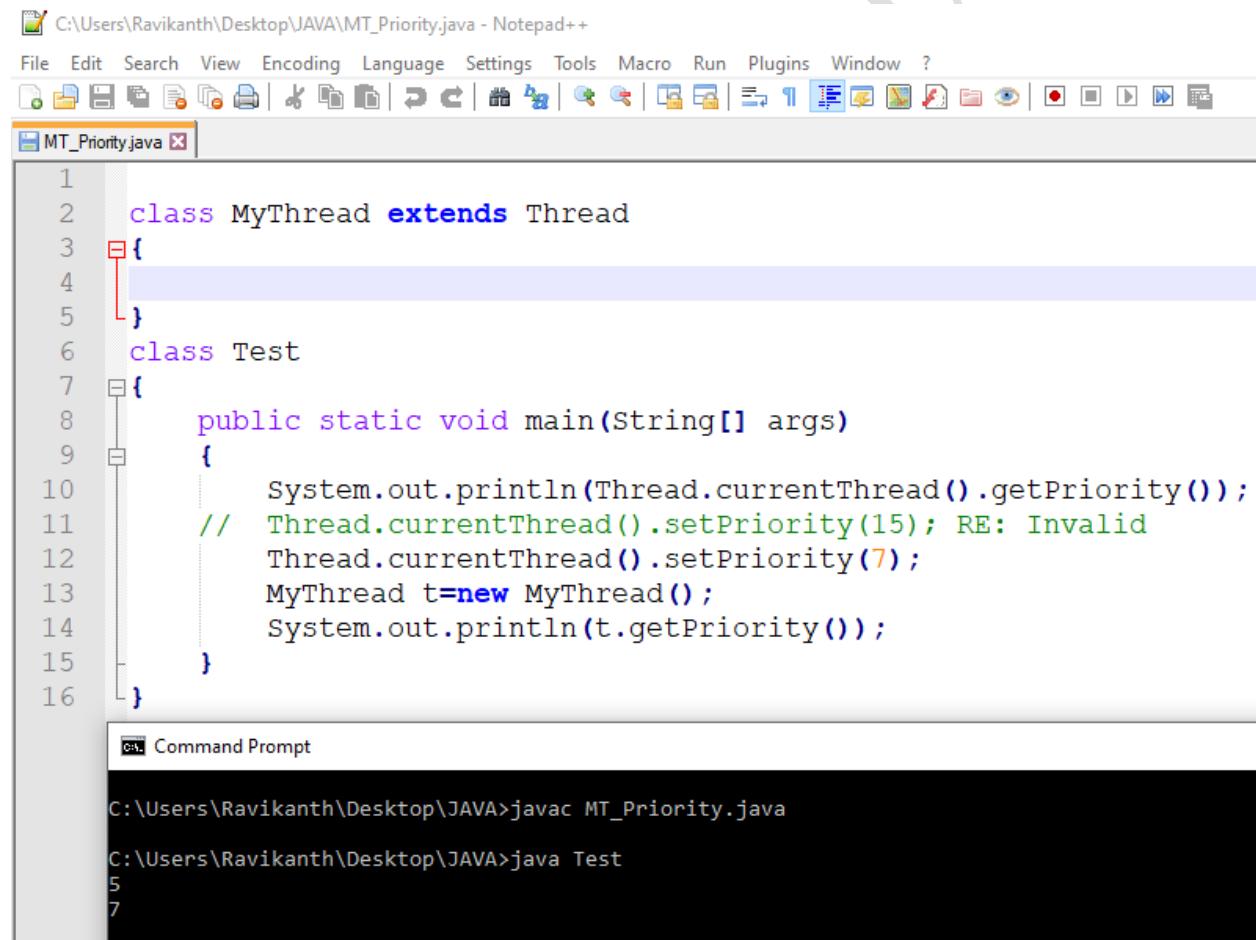
t.setPriority(7) –VALID

t.setPriority(17)- INVALID

### **Default Priority:**

The default priority only for the main thread is 5

But for all remaining threads Default Priority will be inherited from Parent to Child that is what ever priority parent thread has the same priority will be there for Child Thread



```

1  class MyThread extends Thread
2  {
3  }
4
5  class Test
6  {
7      public static void main(String[] args)
8      {
9          System.out.println(Thread.currentThread().getPriority());
10         // Thread.currentThread().setPriority(15); RE: Invalid
11         Thread.currentThread().setPriority(7);
12         MyThread t=new MyThread();
13         System.out.println(t.getPriority());
14     }
15 }
16

```

Command Prompt

```

C:\Users\Ravikanth\Desktop\JAVA>javac MT_Priority.java
C:\Users\Ravikanth\Desktop\JAVA>java Test
5
7

```

The screenshot shows a Java program named MT\_Priority.java being run in a Command Prompt window. The code defines a MyThread class that extends Thread, and a Test class with a main method. In the main method, the current thread's priority is set to 15, then to 7, and finally printed. However, an exception occurs during compilation or execution.

```
C:\Users\Ravikanth\Desktop\JAVA>javac MT_Priority.java
C:\Users\Ravikanth\Desktop\JAVA>java Test
5
Exception in thread "main" java.lang.IllegalArgumentException
    at java.base/java.lang.Thread.setPriority(Thread.java:1137)
    at Test.main(MT_Priority.java:11)
```

NOTE: Some Platforms won't provide proper support for thread priorities.

**UNIT-IV****Topics to be covered:****Multithreading : Synchronization****Synchronization in Java**

- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- Used to solve data inconsistency problem
- Synchronization is the modifier applicable only for methods & blocks
- Cannot be applied for classes and variables
- Synchronization keyword in java creates a block of code known as critical section
- To enter a critical section, a thread needs to obtain the corresponding objects lock

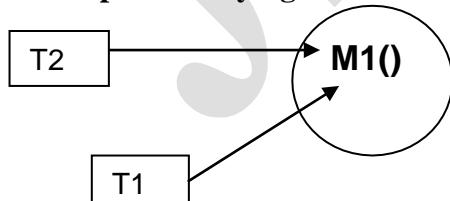
```
synchronized(object)
```

```
{
```

```
// Statements
```

```
}
```

**Ex: Two persons trying to access same method simultaneously**



Note: Now we make this m1() method as Synchronized method so that only one thread can access at a particular time

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

## Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

## Thread Synchronization

There are two types of thread synchronization **mutual exclusive** and **inter-thread communication**.

1. Mutual Exclusive
  1. Synchronized method.
  2. Synchronized block.
  3. static synchronization.
2. Cooperation (Inter-thread communication in java)

### Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

### Understanding the problem without Synchronization

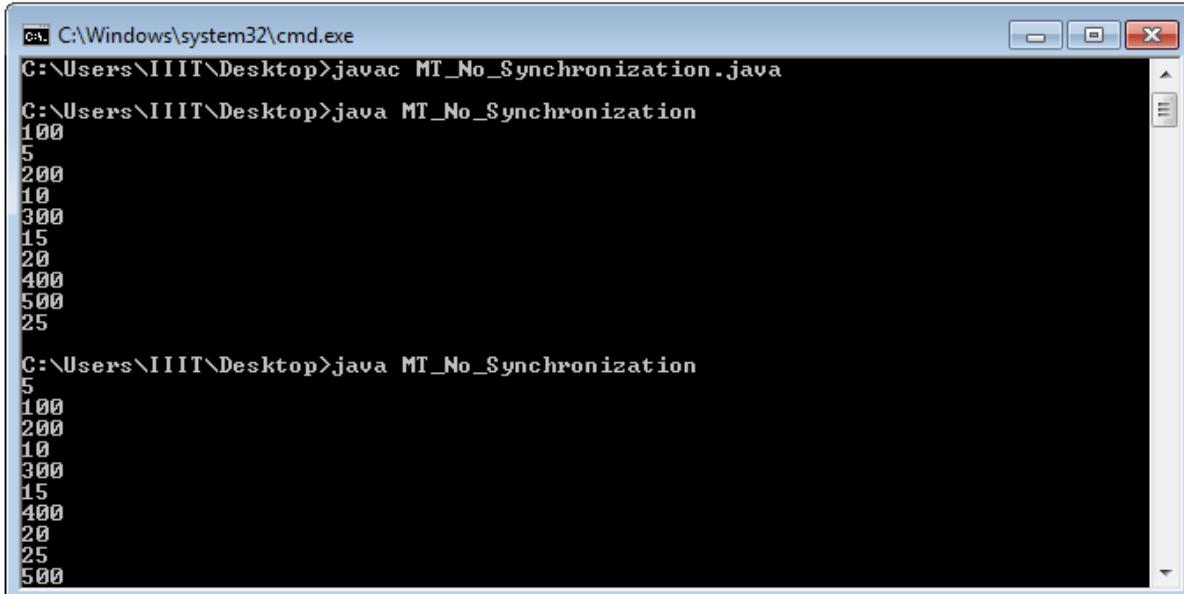
In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```
//In this example, there is no synchronization, so output is inconsistent.
class Table
{
    void printTable(int n)    //method not synchronized
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(400);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}

class MyThread1 extends Thread
{
    Table t;
    MyThread1(Table t)
    {
        this.t=t;
    }
    public void run()
    {
        t.printTable(5);
    }
}

class MyThread2 extends Thread
{
    Table t;
    MyThread2(Table t)
    {
        this.t=t;
    }
    public void run()
    {
        t.printTable(100);
    }
}
```

```
class MT_No_Synchronization
{
    public static void main(String args[])
    {
        Table obj = new Table(); //only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop>javac MT_No_Synchronization.java
C:\Users\IIIT\Desktop>java MT_No_Synchronization
100
5
200
10
300
15
20
400
500
25

C:\Users\IIIT\Desktop>java MT_No_Synchronization
5
100
200
10
300
15
400
20
25
500
```

### Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//In this example, If you declare any method as synchronized
class Table
{
    synchronized void printTable(int n) //synchronized method
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(400);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }

    class MyThread1 extends Thread
    {
        Table t;
        MyThread1(Table t)
        {
            this.t=t;
        }
        public void run()
        {
            t.printTable(5);
        }
    }

    class MyThread2 extends Thread
    {
        Table t;
        MyThread2(Table t)
        {
            this.t=t;
        }
        public void run()
        {
            t.printTable(100);
        }
    }
}
```

```

class MT_Synchronization
{
    public static void main(String args[])
    {
        Table obj = new Table(); //only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}

```

```

C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac MT_Synchronization.java
C:\Users\IIIT\Desktop\RaviKanth>java MT_Synchronization
5
10
15
20
25
100
200
300
400
500
C:\Users\IIIT\Desktop\RaviKanth>

```

### Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method. Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

#### Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

#### Syntax to use synchronized block

1. synchronized (object reference expression)
2. {
3. //code block
4. }

### Example of synchronized block

Let's see the simple example of synchronized block.

```
//Program of synchronized block
class Table
{
    void printTable(int n)
    {
        synchronized(this)
        {//synchronized block
            for(int i=1;i<=5;i++)
            {
                System.out.println(n*i);
                try
                {
                    Thread.sleep(400);
                }
                catch(Exception e)
                {
                    System.out.println(e);
                }
            }
        } // end of Syn_Block
    }//end of the method
}
class MyThread1 extends Thread
{
    Table t;
    MyThread1(Table t)
    {
        this.t=t;
    }
    public void run()
    {
        t.printTable(5);
    }
}
class MyThread2 extends Thread
{
    Table t;
    MyThread2(Table t)
    {
        this.t=t;
    }
    public void run()
    {
        t.printTable(100);
    }
}
```

```
public class Synchronized_Block
{
    public static void main(String args[])
    {
        Table obj = new Table(); //only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac Synchronized_Block.java
C:\Users\IIIT\Desktop\RaviKanth>java Synchronized_Block
5
10
15
20
25
100
200
300
400
500
C:\Users\IIIT\Desktop\RaviKanth>
```

### Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.

```
//Example of static synchronization
//In this example we are applying synchronized keyword
//on the static method to perform static synchronization.
class Table
{
    synchronized static void printTable(int n)
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(400);
            }
            catch(Exception e){}
        }
    }
}
class MyThread1 extends Thread
{
    public void run()
    {
        Table.printTable(1);
    }
}
class MyThread2 extends Thread
{
    public void run()
    {
        Table.printTable(10);
    }
}
```

```
class MyThread3 extends Thread
{
    public void run()
    {
        Table.printTable(100);
    }
}
class MyThread4 extends Thread
{
    public void run()
    {
        Table.printTable(1000);
    }
}
public class static_synchronization
{
    public static void main(String t[])
    {
        MyThread1 t1=new MyThread1();
        MyThread2 t2=new MyThread2();
        MyThread3 t3=new MyThread3();
        MyThread4 t4=new MyThread4();
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac static_synchronization.java
C:\Users\IIIT\Desktop\RaviKanth>java static_synchronization
1
2
3
4
5
6
7
8
9
10
10
20
30
40
50
60
70
80
90
100
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
100
200
300
400
500
600
700
800
900
1000
C:\Users\IIIT\Desktop\RaviKanth>
```

## Inter-thread communication in Java

**Inter-thread communication or Co-operation** is all about allowing synchronized threads to communicate with each other. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

### 1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

### 2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax: public final void notify()

### 3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax: public final void notifyAll()

## Understanding the process of inter-thread communication

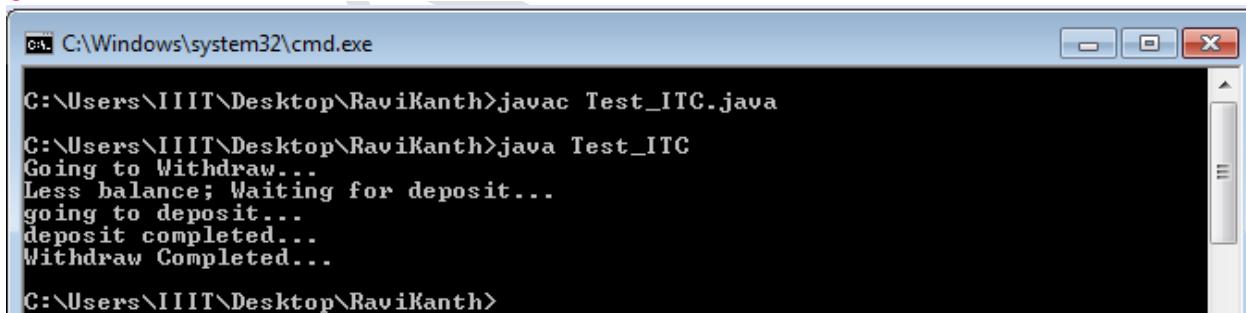
The point to point explanation is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by one thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

**Example of inter thread communication in java**

```
//Example over inter thread communication.
class Customer
{
    int amount=10000;
    synchronized void withdraw(int amount)
    {
        System.out.println("Going to Withdraw...");
        if(this.amount<amount)
        {
            System.out.println("Less balance; Waiting for deposit...");
            try
            {
                wait();
            }
            catch(Exception e){}
        }
        this.amount-=amount;
        System.out.println("Withdraw Completed...");
    }
    synchronized void deposit(int amount)
    {
        System.out.println("going to deposit...");
        this.amount+=amount;
        System.out.println("deposit completed... ");
        notify();
    }
}
```

```
class Test_ITC
{
    public static void main(String args[])
    {
        final Customer c=new Customer();
        new Thread()
        {
            public void run()
            {
                c.withdraw(15000);
            }
        }.start();
        new Thread()
        {
            public void run()
            {
                c.deposit(10000);
            }
        }.start();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac Test_ITC.java
C:\Users\IIIT\Desktop\RaviKanth>java Test_ITC
Going to Withdraw...
Less balance; Waiting for deposit...
going to deposit...
deposit completed...
Withdraw Completed...

C:\Users\IIIT\Desktop\RaviKanth>
```

\*\*\*\*\*

### **Thread Synchronization**

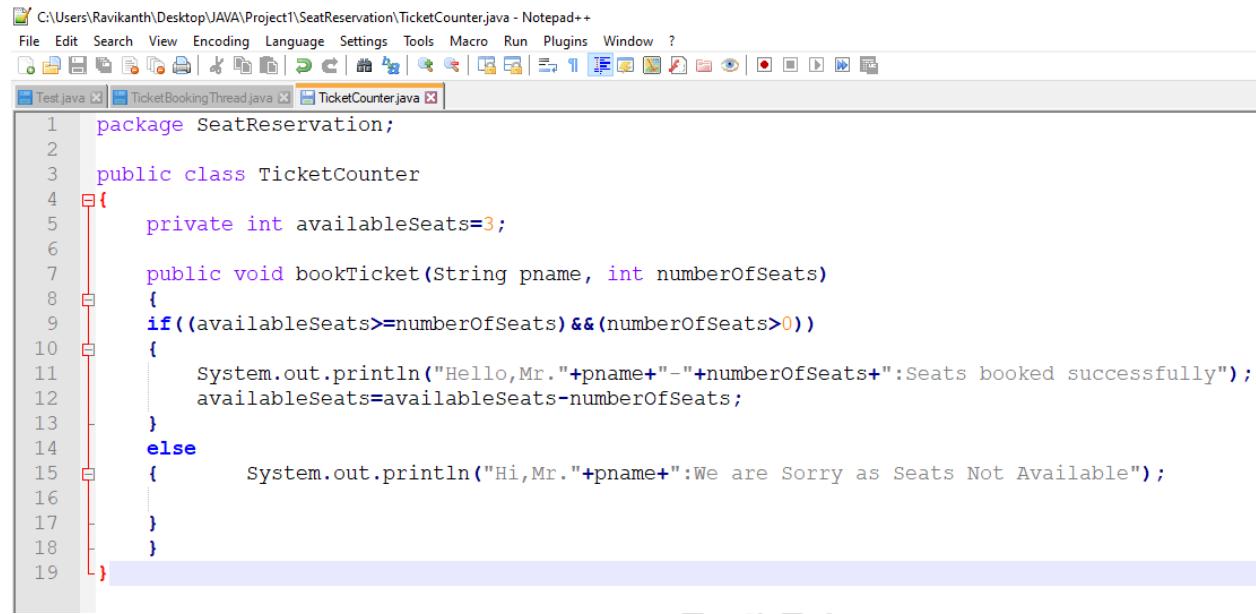
- Synchronized is the modifier applicable only for methods and blocks but not for classes and variables
- If multiple threads are trying to operate simultaneously on the same java object then there may be chance of data inconsistency problem
- To overcome this problem we should go for Synchronized keyword
- If a method or a Block declared as synchronized then at a time only one thread is allowed to execute that method or Block on the given Object, so that Data inconsistency problem will be resolved

#### **Note:**

- The main advantage of synchronized keyword is we can resolve Data Inconsistency problems
- But the main dis-advantage of synchronized keyword is it increases WAITING TIME of threads and creates Performance Problems
- Hence if there is no specific requirement then it is not recommended to use synchronized keyword
- Internally synchronization is implemented by using Lock every object in Java has a unique Lock
- Whenever we are using Synchronized keyword then only Lock concept will come into picture
- If a thread wants to execute synchronized method on the given object first it has to get lock of that object
- Once thread got the Lock then it is allowed to execute any synchronized method on that object
- Once method execution completes automatically thread releases the Lock
- Acquiring and Releasing Lock is internally takes care by JVM and Programmer is not responsible for this activity

## Example Program:

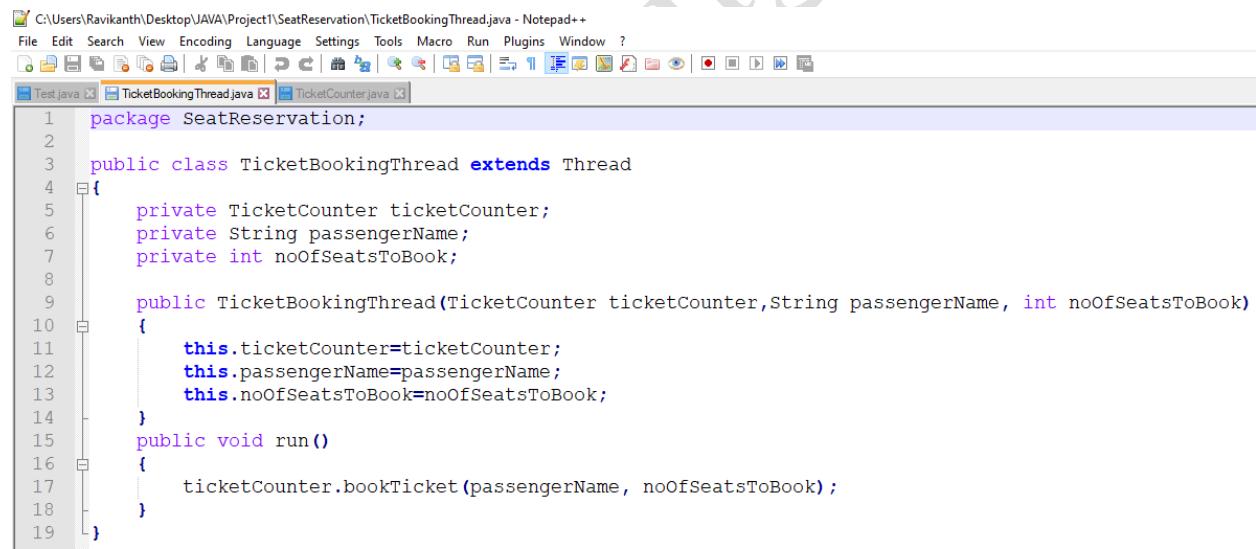
### Project1: Seat Reservation System



```

1 package SeatReservation;
2
3 public class TicketCounter
4 {
5     private int availableSeats=3;
6
7     public void bookTicket(String pname, int numberOfSeats)
8     {
9         if((availableSeats>=numberOfSeats) &&(numberOfSeats>0))
10        {
11            System.out.println("Hello,Mr." +pname+"-"+numberOfSeats+":Seats booked successfully");
12            availableSeats=availableSeats-numberOfSeats;
13        }
14        else
15        {
16            System.out.println("Hi,Mr." +pname+":We are Sorry as Seats Not Available");
17        }
18    }
19 }

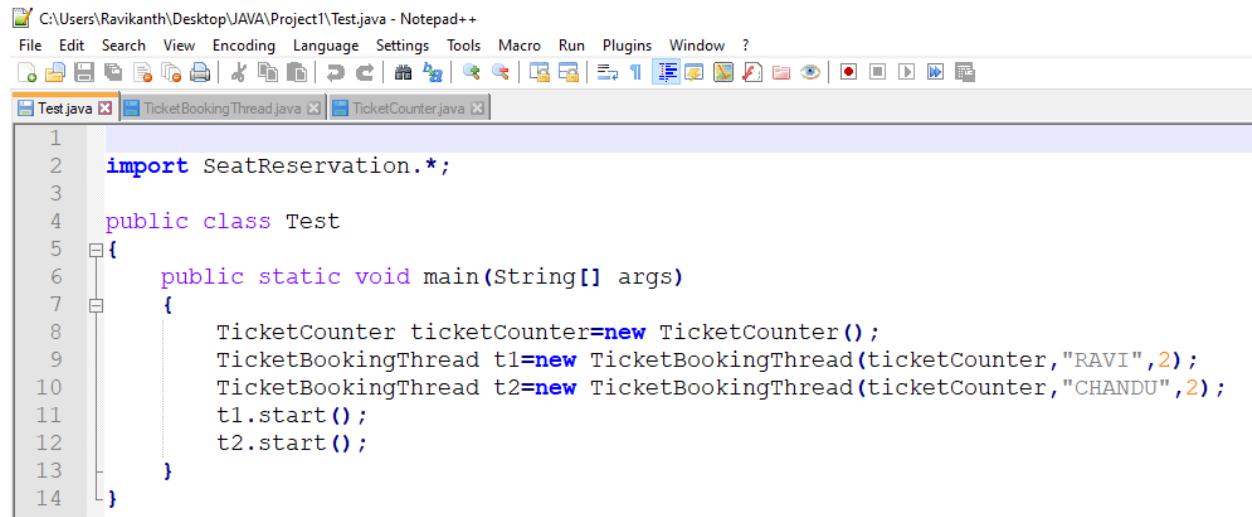
```



```

1 package SeatReservation;
2
3 public class TicketBookingThread extends Thread
4 {
5     private TicketCounter ticketCounter;
6     private String passengerName;
7     private int noOfSeatsToBook;
8
9     public TicketBookingThread(TicketCounter ticketCounter, String passengerName, int noOfSeatsToBook)
10    {
11        this.ticketCounter=ticketCounter;
12        this.passengerName=passengerName;
13        this.noOfSeatsToBook=noOfSeatsToBook;
14    }
15    public void run()
16    {
17        ticketCounter.bookTicket(passengerName, noOfSeatsToBook);
18    }
19 }

```



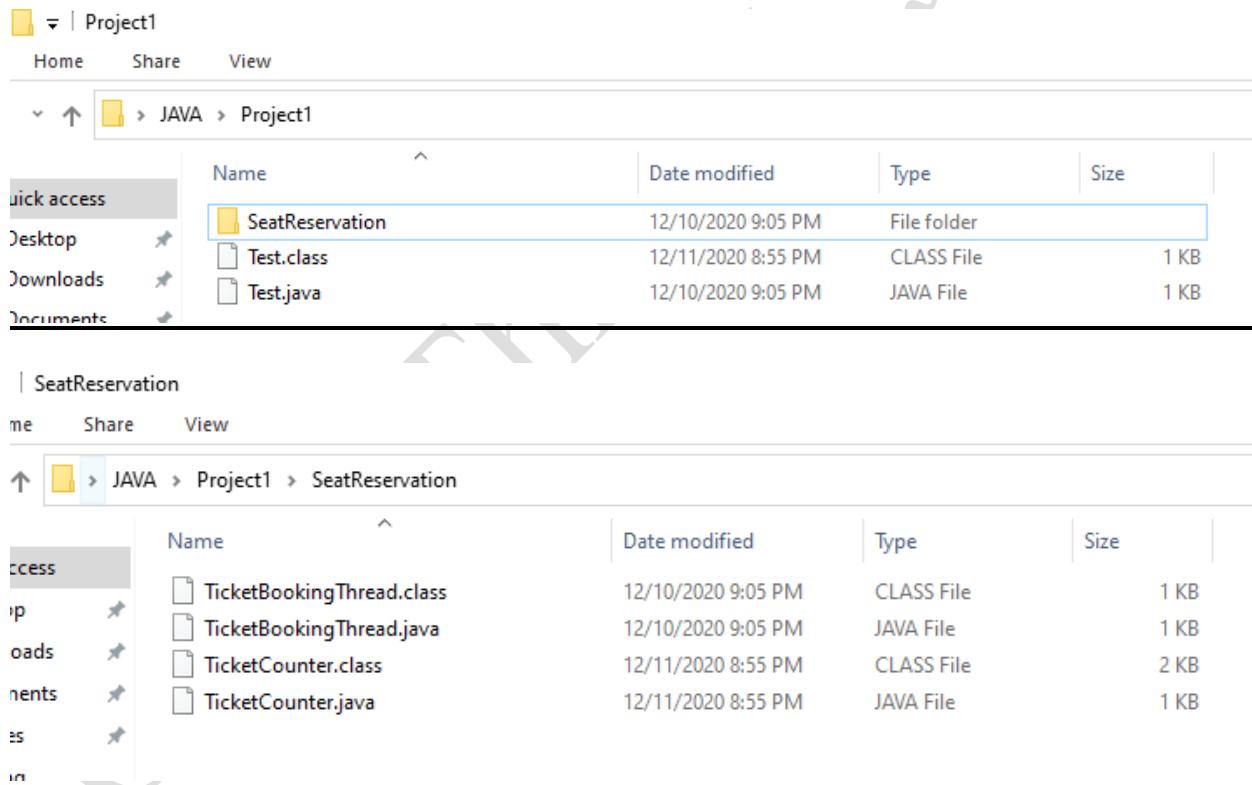
```

C:\Users\Ravikanth\Desktop\JAVA\Project1\Test.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.java TicketBookingThread.java TicketCounter.java

1 import SeatReservation.*;
2
3
4 public class Test
5 {
6     public static void main(String[] args)
7     {
8         TicketCounter ticketCounter=new TicketCounter();
9         TicketBookingThread t1=new TicketBookingThread(ticketCounter, "RAVI", 2);
10        TicketBookingThread t2=new TicketBookingThread(ticketCounter, "CHANDU", 2);
11        t1.start();
12        t2.start();
13    }
14 }

```

### Compile the Project1:



**Execute and Run Project1:**

Command Prompt

```
C:\Users\Ravikanth\Desktop\JAVA\Project1>javac Test.java
C:\Users\Ravikanth\Desktop\JAVA\Project1>java Test
Hello,Mr.CHANDU-2:Seats booked successfully
Hello,Mr.RAVI-2:Seats booked successfully
C:\Users\Ravikanth\Desktop\JAVA\Project1>
```

**NOW LET US USE SYNCHRONIZATION KEYWORD**

```
C:\Users\Ravikanth\Desktop\JAVA\Project1>javac Test.java
C:\Users\Ravikanth\Desktop\JAVA\Project1>java Test
Hello,Mr.CHANDU-2:Seats booked successfully
Hello,Mr.RAVI-2:Seats booked successfully
C:\Users\Ravikanth\Desktop\JAVA\Project1>javac Test.java
C:\Users\Ravikanth\Desktop\JAVA\Project1>java Test
Hello,Mr.RAVI-2:Seats booked successfully
Hi,Mr.CHANDU:We are Sorry as Seats Not Available
C:\Users\Ravikanth\Desktop\JAVA\Project1>
```

**Do observe the Changes in the Result before using Synchronization and After using Synchronization concept**

```
Command Prompt  
C:\Users\Ravikanth\Desktop\JAVA\Project1>javac Test.java  
C:\Users\Ravikanth\Desktop\JAVA\Project1>java Test  
Hello,Mr.CHANDU-2:Seats booked successfully  
Hello,Mr.RAVI-2:Seats booked successfully  
C:\Users\Ravikanth\Desktop\JAVA\Project1>javac Test.java  
C:\Users\Ravikanth\Desktop\JAVA\Project1>java Test  
Hello,Mr.RAVI-2:Seats booked successfully  
Hi,Mr.CHANDU:We are Sorry as Seats Not Available  
C:\Users\Ravikanth\Desktop\JAVA\Project1>
```

Project2: Customers performing Transactions on Account

C:\Users\Ravikanth\Desktop\JAVA\Project2\Banking\Account.java - Notepad++

```

1 package Banking;
2
3 public class Account
4 {
5     private int balance=6000;
6
7     public int getBalance()
8     {
9         return balance;
10    }
11    public void withdraw(int amount)
12    {
13        balance=balance-amount;
14    }
15 }

```

C:\Users\Ravikanth\Desktop\JAVA\Project2\Banking\AccountHolder.java - Notepad++

```

1 package Banking;
2
3 public class AccountHolder implements Runnable
4 {
5     private Account account;
6
7     public AccountHolder(Account account)
8     {
9         this.account=account;
10    }
11    public void run()
12    {
13        for(int i=1;i<=4;i++)
14        {
15            makeWithdrawal(2000);
16            if(account.getBalance()<0)
17            {
18                System.out.println("Dear Customer Account is OverDrawn..."); 
19            }
20        }
21    }

```

```

22     private void makeWithdrawal(int withdrawAmount)
23     {
24         if(account.getBalance()>=withdrawAmount)
25         {
26             System.out.println(Thread.currentThread().getName()+" is going to withdraw Amount of Rs: "+withdrawAmount);
27             try
28             {
29                 Thread.sleep(3000);
30             }
31             catch(InterruptedException ex)
32             {
33             }
34         }
35         account.withdraw(withdrawAmount);
36         System.out.println(Thread.currentThread().getName()+" Completes the Withdrawal of Rs: "+withdrawAmount+" Successfully");
37     }
38     else
39     {
40         System.out.println("Not enough in account for"+Thread.currentThread().getName()+" to withdraw Rs: "+account.getBalance());
41     }
42 }
43

```

Activate Windows

```

C:\Users\Ravikanth\Desktop\JAVA\Project2\Customer.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Customer.java Account.java AccountHolder.java
1 import Banking.*;
2
3 public class Customer
4 {
5     public static void main(String [] args)
6     {
7         Account account=new Account();
8         AccountHolder accountHolder=new AccountHolder(account);
9         Thread t1=new Thread(accountHolder);
10        Thread t2=new Thread(accountHolder);
11        t1.setName(" Mr.RAVI");
12        t2.setName(" Mr.CHANDU ");
13        t1.start();
14        t2.start();
15    }
16 }

```

Execute and Run the Code:

Project2

| Name           | Date modified      | Type        | Size |
|----------------|--------------------|-------------|------|
| Banking        | 12/10/2020 9:30 PM | File folder |      |
| Customer.class | 12/11/2020 9:04 PM | CLASS File  | 1 KB |
| Customer.java  | 12/10/2020 9:42 PM | JAVA File   | 1 KB |

Banking

| Name                | Date modified      | Type       | Size |
|---------------------|--------------------|------------|------|
| Account.class       | 12/10/2020 9:30 PM | CLASS File | 1 KB |
| Account.java        | 12/10/2020 9:11 PM | JAVA File  | 1 KB |
| AccountHolder.class | 12/11/2020 9:04 PM | CLASS File | 2 KB |
| AccountHolder.java  | 12/11/2020 9:04 PM | JAVA File  | 1 KB |

Command Prompt

```
C:\Users\Ravikanth\Desktop\JAVA\Project2>javac Customer.java
C:\Users\Ravikanth\Desktop\JAVA\Project2>java Customer
Mr.CHANDU is going to withdraw Amount of Rs: 2000
Mr.RAVI is going to withdraw Amount of Rs: 2000
Mr.CHANDU Completes the Withdrawl of Rs: 2000 Successfully
Mr.RAVI Completes the Withdrawl of Rs: 2000 Successfully
Mr.CHANDU is going to withdraw Amount of Rs: 2000
Mr.RAVI is going to withdraw Amount of Rs: 2000
Mr.RAVI Completes the Withdrawl of Rs: 2000 Successfully
Dear Customer Account is OverDrawn...
Mr.CHANDU Completes the Withdrawl of Rs: 2000 Successfully
Dear Customer Account is OverDrawn...
Not enough in account for Mr.RAVI to withdraw Rs: -2000
Dear Customer Account is OverDrawn...
Not enough in account for Mr.RAVI to withdraw Rs: -2000
Dear Customer Account is OverDrawn...
Not enough in account for Mr.CHANDU to withdraw Rs: -2000
Dear Customer Account is OverDrawn...
Not enough in account for Mr.CHANDU to withdraw Rs: -2000
Dear Customer Account is OverDrawn...
C:\Users\Ravikanth\Desktop\JAVA\Project2>
```

**Now Let us use Synchronization Concept**

```
Command Prompt  
C:\Users\Ravikanth\Desktop\JAVA\Project2>javac Customer.java  
C:\Users\Ravikanth\Desktop\JAVA\Project2>java Customer  
Mr.RAVI is going to withdraw Amount of Rs: 2000  
Mr.RAVI Completes the Withdrawl of Rs: 2000 Successfully  
Mr.RAVI is going to withdraw Amount of Rs: 2000  
Mr.RAVI Completes the Withdrawl of Rs: 2000 Successfully  
Mr.RAVI is going to withdraw Amount of Rs: 2000  
Mr.RAVI Completes the Withdrawl of Rs: 2000 Successfully  
Not enough in account for Mr.RAVI to withdraw Rs: 0  
Not enough in account for Mr.CHANDU to withdraw Rs: 0  
Not enough in account for Mr.CHANDU to withdraw Rs: 0  
Not enough in account for Mr.CHANDU to withdraw Rs: 0  
Not enough in account for Mr.CHANDU to withdraw Rs: 0  
C:\Users\Ravikanth\Desktop\JAVA\Project2>
```

**Do execute and try the Code for both the Projects.**

\*\*\*\*\*

UNIT-III**Topics to be covered:**

**Class and objects:** Defining a class, Methods, Creating objects, Accessing class members, Constructors, Static members, Nesting of Methods, this keyword, Command line input.

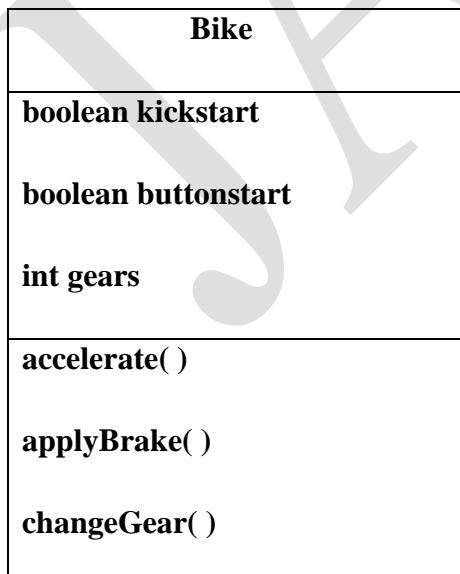
**Class:**

A Class is a blueprint or a prototype that defines the variables and methods common to all objects of a certain kind.

In other words, a class can be thought of as a User-Defined datatype and an object as a variable of that data type that can contain data and methods,i.e, functions working on that data.

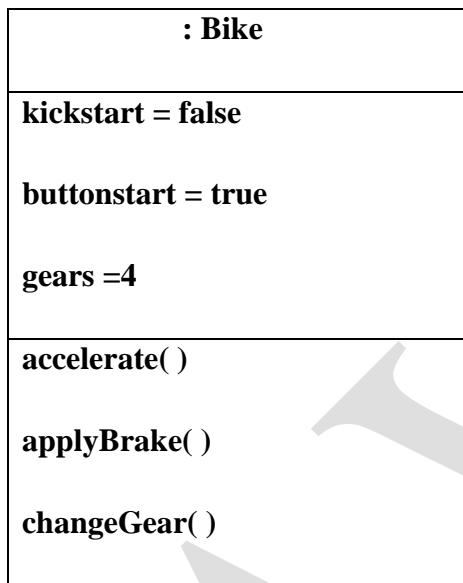
**Differences b/w class and Object:**

- A class is a model for creating objects and doesnot exist physically.
- A object is anything that exists physically
- Both the class and object contains Variables and methods.

**Ex: Bike Class**

**Object:**

An object is a software bundle that encapsulates variables and methods operating on those variables.

**Ex: Bike Object****Defining a Class**

A class is created by using the keyword ‘class’. Class Body consists of class members which includes Fields/ Variables & Methods/ Operations

[ Access\_Modifier] class ClassName // Access Modifier is Optional

{

// Members of Class

**Fields Declaration;**

**Methods Declaration;**

}

**Ex:** class Ece

```
class EceStudent
```

**Field Declaration:**

[Access Modifier] type variable\_name = [ initial value];

**Ex:** int rollNumber;

```
float percentage;
```

**Method Declaration:**

[ Access\_Modifier] returntype methodName ( parameter\_list) // Formal Parameters

```
{
```

// Method Body;

```
}
```

**Ex:** int addMarks( int x, int y)

```
{
```

// Body

```
}
```

**Example covering all together:**

```
class Person
{
    // Fields
    String name;
    int age;
    //Methods
    void talk()
    {
        //code
    }
}
```

```

        }
        void sleep()
        {
            //code
        }
    }
}

```

class Person has two variables and two methods.

This class method is stored in JVM's method area.

When we want to use this class, we should create an object to the class as:

```

class PersonExample
{
    public static void main(String args[])
    {
        //create Person class object
        Person P1 =new Person();
        //call the methods
        SOP(P1.name); //output is : null ... as they occupied by default
        values
        SOP(P1.age); // output is: 0
    }
}

```

### Creating a Object:

The class code along with method code is stored in 'Method area' of the JVM When an object is created, the memory is allocated on heap, after creation of the object. JVM produced a unique reference number for the object from the memory address of the object. This reference number is called hash code num.

To know the Object reference number, we can use hashCode() method of Object class.

Ex: Person x = new Person(); // x ref. to Person obj.

SOP(x.hashCode()); //displays the hash code ref. num stored in x.

**Default values of instance variables by java compiler:**

| Data Type              | Default Value (for fields) |
|------------------------|----------------------------|
| byte                   | 0                          |
| short                  | 0                          |
| int                    | 0                          |
| long                   | 0L                         |
| float                  | 0.0f                       |
| double                 | 0.0d                       |
| char                   | '\u0000'                   |
| String (or any object) | null                       |
| boolean                | false                      |

We can create an object using new keyword

### Declaring an Object

Student obj;

### Instantiate the Object

Obj= new Student( );

### Both together

Student obj = new Student ( );

### Accessing Members of a class using Dot. Operator

Object\_Name.Variable\_Name = Value;

Object\_Name.Method\_Name(Parameters\_list);

**Ex:** obj.roll= 1230;

obj.add(92,78);

**We can create any number of objects and we can create any number of instance variables.**

Student obj1= new Student( );

Student obj2= new Student( );

**Single Object can have more than one references.**

Student obj1= new Student( );

Student obj2= obj1;

**Example1:**

```
class Person
{
    String name;
    int age;
    void talk()
    {
        System.out.println("My name is :" +name);
        System.out.println ("My age is: "+age);
    }
}
Class Demo
{
    public static void main(String args[])
    {
        Person x = new Person();
        x.talk();
    }
}
```

**O/P:** My name is null

My age is 0

**Example2:**

```

class Employee
{
    String empid; // data member (or instance variable)
    String empname; // data member (or instance variable)
    empid=RCTS160326;
    empname="RaviKanth";
    public static void main(String args[])
    {
        Employee e1=new Employee(); // Creating an object of class
        System.out.println("Employee ID: "+e1.empid);
        System.out.println("Name: "+e1.empname);
    }
}

```

**Output:** Employee ID: RCTS160326

Name: RaviKanth

**Example3:****Over Multiple classes:**

```

class Test
{
    int a,b,c;

    void getData(int x, int y)
    {
        a=x;
        b=y;
    }

    void add()
}

```

```
{  
    c=a+b;  
  
    System.out.println("Addition of two numbers is:" +c);  
}  
  
}  
  
class Main  
{  
  
    public static void main(String a[ ])  
    {  
        Test obj=new Test();  
  
        obj.getData( 4,6);  
  
        obj.add( );  
    }  
}
```

**Output: 10**

#### Initializing the Objects in Java can be done in 3 Possible ways:

1. By reference variable
2. By method
3. By constructor

#### **1) Object and Class Example: Initialization through reference variable**

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

**Example: 4**

```

class Student
{
    int id;
    String name;
}

class TestStudent2
{
    public static void main(String args[])
    {
        Student s1=new Student();
        s1.id=1230;
        s1.name="RGUKT";
        System.out.println(s1.id+" "+s1.name);
        //printing members with a white space
    }
}

```

**Output:** 1230 RGUKT

**2) Object and Class Example: Initialization through method**

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation( ) method.

**Example 5:**

```

class Student
{
    int rollno;
    String name;
    void insertRecord(int r, String n)
    {
        rollno=r;
        name=n;
    }
    void displayInformation()
    {
        System.out.println(rollno+" "+name);
    }
}

```

```

    }
class TestStudent4
{
    public static void main(String args[])
    {
        Student s1=new Student();
        Student s2=new Student();
        s1.insertRecord(1230,"Rajesh");
        s2.insertRecord(222,"Mahesh");
        s1.displayInformation();
        s2.displayInformation();
    }
}

```

**Output:** 111 Rajesh  
222 Mahesh

### 3) Initializing through constructor:

#### Example: 6

```

class Person
{
    //instance variables
    String name;
    int age;
    Person()
    {
        //default constructor
        name="RGUKT";
        age=10;
    }
    void talk()
    {
        SOP("My name is: "+name);
        SOP("My age is: "+age);
    }
}
class Demo
{
    public static void main(String args[])
    {
        Person x = new Person();
    }
}

```

```

        x.talk();
    }
}

```

**O/P:** My name is RGUKT  
My age is 10

### Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

```
int a=10, b=20;
```

Initialization of reference variables:

```
Rectangle r1=new Rectangle(), r2=new Rectangle(); //creating two objects
```

### Example 7:

```

class Rectangle
{
    int length;
    int width;
    void insert(int l,int w)
    {
        length=l;
        width=w;
    }
    void calculateArea()
    {
        System.out.println(length*width);
    }
}
class TestRectangle2
{
    public static void main(String args[])
    {
        Rectangle r1=new Rectangle(),r2=new Rectangle();
        //creating two objects
        r1.insert(11,5);
        r2.insert(3,15);
        r1.calculateArea();
        r2.calculateArea();
    }
}

```

```
}
```

```
}
```

**Output:**

55

45



**Methods in Java:-**

A method represents a group of statements that performs a task. Here the task represents a calculation or processing of data or generating a report etc..

ex: sqrt() method

It calculates square root value and returns that value.

Method has two parts:

1. method header or method prototype
2. method body

**1. Method header or method prototype:**

It contains method name, method parameters and method return type.

Syntax:

```
return datatype methodname(parameter1, parameter2....)
```

**2. Method body:**

Method body consists a group of statements which contains logic to perform the task.

Syntax:

```
{
    stmts;
}
```

**Method types:**

1. Method with No parameters and no return type
2. Method with parameters and with return type
3. Method without parameters and with return type
4. Method with parameters and without return type

**Static Keyword:**

- A static method is a method that doesn't act upon instance variables of a class.
- A static method is declared by using the keyword static.
- A static methods are called using **classname.methodname()**
- The reason why static methods can't act on instance variables in that the JVM first executes the static methods and they only it creates the objects. Since the objects are not available at the time of calling the static methods, the instance variables are also not available.

The static members can be:

1. static variables
2. static methods
3. static block

**1. static variables:-**

- static variables in java is available which belongs to the class and initialized only once at the start of the execution.
- It is a variable which belongs to the class and not to the object (instance).
- static variables initialized first, before the initialization of any instance variables.
- a single copy to be shared by all instance of a class.
- A static variable can be accessed directly by the class name and doesn't need any object.

**Syntax:**

classname.varablename;

**Example:**

```
class Person
{
    int rollno;
    String name;
    static String branch="ECE";
    Person(int n,String s)
    {
        rollno=n;
        name=s;
    }
    void display()
    {
```

```

        SOP(rollno+" "+name+" "+branch);
    }
}
class Person_Demo
{
    public static void main(String args[])
    {
        Person p=new Person(1001,"RGUKT");
        p.display();
    }
}

```

O/P: 1001 RGUKT ECE

## **2. static methods:**

- static method in java is a method which belongs to the class and not the object. A static method can access only static data.
- It can't access non static data.
- A static method can be accessed directly by the class name and doesn't need any object.
- A static method can't refer 'this' and 'super' keywords.
- Syntax:

**classname.methodname();**

### **Example:**

```

class Test
{
    static int count=10;
    static void access()
    {
        System.out.println(count);
    }
    public static void main(String args[])
    {
        Test.access()
    }
}

```

O/P: 10

**3. static block:-**

A static block is block of statements declared as static.

Syntax:

```
static
{
    //code
}
```

- JVM executes a static block on a highest priority basis. This means JVM first goes to static block even before it looks for main() method in a program.

**Example:**

```
class Test
{
    Static
    {
        SOP("static block");
    }
    public static void main(String args[])
    {
        SOP("main block");
    }
}
O/P: static block
      main block
```

**Difference between instance variables and class variables:-**

- An instance variable is a variable whose separate copy is available to each objects
- A class variable is a variable whose single copy in memory is shared by all objects
- Instance variables are created in the objects on heap memory.
- Class variables (static variables) are stored on method area.

**this keyword:**

this is a reference variable that refers to the current object.

It is a keyword in java language represents current class object

**Usage of this keyword**

- It can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- It can be used to invoke current class method (implicitly)
- It can be passed as an argument in the method call.
- It can be passed as argument in the constructor call.
- It can also be used to return the current class instance.

**Use of this keyword in java:**

The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then JVM get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

"**this**" keyword can be use in two ways.

- this . (this dot)
- this() (this off)

**this . - - (this dot)**

which can be used to differentiate variable of class and formal parameters of method or constructor.

**Syntax**

this.data member of current class.

**Example without using this keyword**

```
class Employee
{
    int id;
    String name;
    Employee(int id,String name)
    {
        id = id;
```

```

        name = name;
    }
    void show()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"rgukt");
        Employee e2 = new Employee(112,"iiit");
        e1.show();
        e2.show();
    }
}

```

**Output**

0 null  
0 null

In the above example, parameter (formal arguments) and instance variables are same that is why we are using "**this**" keyword to distinguish between local variable and instance variable.

**Example of this keyword in java**

```

class Employee
{
    int id;
    String name;
    Employee(int id,String name)
    {
        this.id = id;
        this.name = name;
    }
    void show()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"rgukt");
        Employee e2 = new Employee(112,"iiit");
        e1.show();
        e2.show();
    }
}

```

```

        }
    }
}
```

**Output**

111 rgukt  
112 iiit

**Difference between this and super keyword**

**Super keyword** is always pointing to base class (scope outside the class) features and "**this**" keyword is always pointing to current class (scope is within the class) features.

**Example when no need of this keyword**

```

class Employee
{
    int id;
    String name;
    Employee(int i,String n)
    {
        id = i;
        name = n;
    }
    void show()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"rgukt");
        Employee e2 = new Employee(112,"iiit");
        e1.show();
        e2.show();
    }
}
```

**Output**

111 rgukt  
112 iiit

In the above example, no need of use this keyword because parameter (formal arguments) and instance variables are different. This keyword is only use when parameter (formal arguments) and instance variables are same.

### **Constructors in Java**

Constructor is a special type of method with the same name as the class name

Constructor will not have return type even void type

It initializes the Object, which would assign the initial value

Constructor is called the movement we create an object

**Def:** A constructor is similar to a method that is used to initialize the instance variables. The sole purpose of a constructor is to initialize the instance variables.

#### **Characteristics of constructors:-**

- The constructor's name and class name should be same. The constructor's name should end with a pair of braces.

**Ex:**

```
student()
{}
```

- A constructor may have or may not have parameters. Parameters are variables to receive data from outside into the constructor. If a constructor does not have any parameters, it is called as 'default constructor'. If a constructor has 1 or more parameters, it is called parameterized constructor.

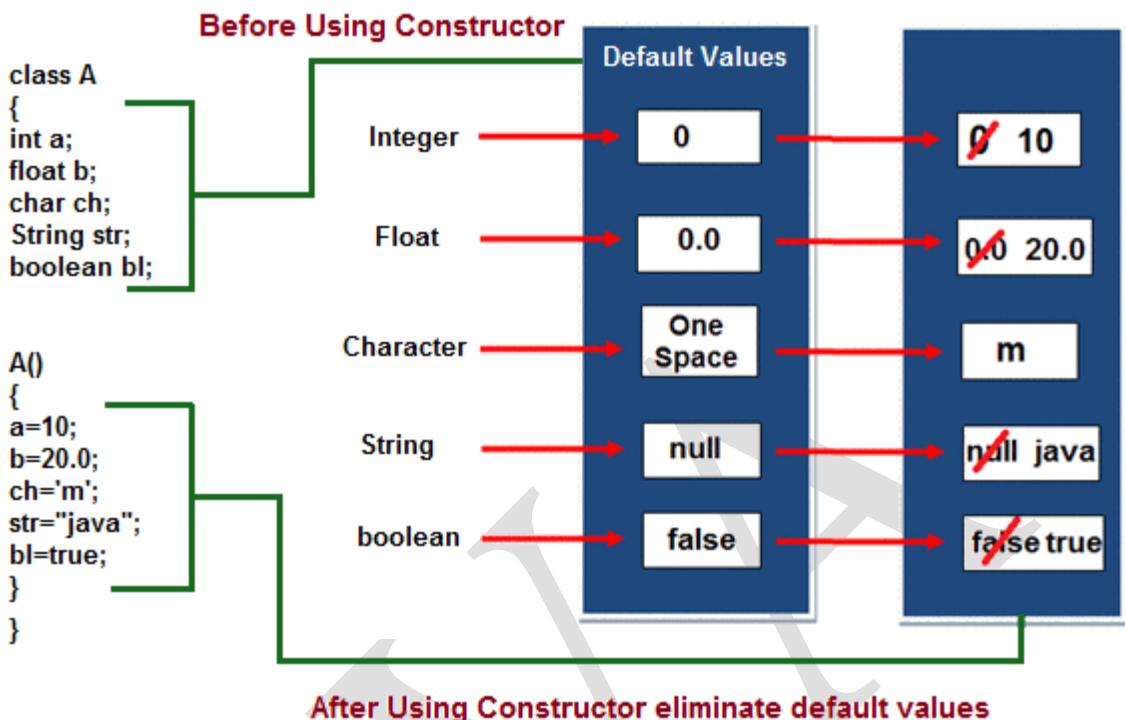
**Ex:**

parameterized constructor:

```
student(int i, int s)
{
```

```
}
```

- A constructor does not return any value, not even void type.
- A constructor is automatically called and executed at the time of creating an object. While creating an object, if nothing is passed to the object the default constructor is called and executed.
- If some values are passed to the object, then the parameterized constructor is called.
- A constructor is called and executed only once per object.



Constructors are of two types:

**Default Constructor:**

If we are not creating any constructor then it is automatically created by Compiler

```

class Student
{
    Student()
    {
    }
}

```

Student obj=new Student( );

**Ex:**

```
class Test
{
    int a,b; //Instance Variable
    Test( )
    {
        System.out.println("Default Constructor");
        a=10;
        b=20;
    }
    void disp( )
    {
        System.out.println(" Value of A"+ a);
        System.out.println("Value of B"+b);
    }
}
class Main
{
    Public static void main(String a[])
    {
        Test obj=new Test();
        obj.disp();
    }
}
```

**Parameterized Constructor :** We are initializing the values using parameters

```
class Student  
{  
    Student( int x, int y)  
    {  
        a=x;  
        b=y;  
    }  
}  
  
Student obj=new Student(10,20 );
```

**Ex:**

```
class Student  
{  
    Student( int x, int y)  
    {  
        a=x;  
        b=y;  
    }  
}  
  
Student obj=new Student(20,30);
```

**Ex:**

Class Rectangle

```
{  
    int length,width,result;  
  
    Rectangle(int x,int y)  
    {  
        length=x;  
        width=y;  
    }  
  
    void rectArea( )  
    {  
        result=length*width;  
        System.out.println("Area of Rectangle"+res);  
    }  
}  
  
class Test  
{  
    Public static void main(String a[])  
    {  
        Rectangle obj=new Rectangle(20,30);  
        obj.recArea( );  
    }  
}
```

**Example:** - Example over default constructor

```
class Sum
{
    int a,b;
    Sum() // Default Constructor
    {
        a=10;
        b=20;
    }
    public static void main(String s[])
    {
        Sum s=new Sum();
        c=a+b;
        System.out.println("Sum: "+c);
    }
}
```

**Example:** Example over parameterized constructor

```
class Test
{
    int a, b;
    Test(int n1, int n2)
    {
        System.out.println("I am from Parameterized Constructor...");
        a=n1;
        b=n2;
        System.out.println("Value of a = "+a);
        System.out.println("Value of b = "+b);
    }
}
class TestDemo1
{
    public static void main(String k [])
    {
        Test t1=new Test(10, 20);
    }
}
```

**Command Line Arguments:**

- The java command line argument is an argument i.e., passed at the time of running the program.
- the arguments passed from the console can be received in java and it can be used as input.
- You can pass N number of arguments from a command prompt.

Example:

```
class CMD
{
    public static void main(String args[])
    {
        SOP("Your first argument: "+args[0]);
    }
}
```

Input: RGUKT

Output: Your first argument: RGUKT

## UNIT-III

### **Topics to be covered:**

**Polymorphism** – Static Polymorphism, Dynamic Polymorphism, Method overloading, Polymorphism with Static Methods, Private Methods and Final Methods.

### **Java Polymorphism**

A Polymorphism can be defined as an object showing different behaviors at different stages of its life cycle. Or performing single action in different ways.

### **Two types of Polymorphism**

1. Compile time Polymorphism / Static Binding/ Early Binding
  - We achieve CTP through Method Overloading or Constructor Overloading
2. Run-time Polymorphism/ Dynamic Binding/ Late Binding
  - We achieve RTP through Method Overriding

### **Overloading is of three types**

1. Method Overloading
2. Constructor Overloading
3. Operator Overloading

### **Method Overloading in Java**

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as method overloading.

A method can be used to perform different tasks at different stages

## Syntax

```
class class_Name  
{  
    Returntype method()  
    {.....}  
    Returntype method(datatype1 variable1)  
    {.....}  
    Returntype method(datatype1 variable1, datatype2 variable2)  
    {.....}  
    Returntype method(datatype2 variable2)  
    {.....}  
    Returntype method(datatype2 variable2, datatype1 variable1)  
    {.....}  
}
```

## Different ways to overload the method

1. Number of Arguments
2. Type of Arguments [ Data type]
3. Sequence of Arguments

## Method Overloading

If the class contains more than one method with the same name but different number of arguments then they are called as method overloading

```
void m1(int a)
```

```
void m1(int a, int b)
```

If the class contains more than one method with the same name and same number of arguments but different type of arguments then they are called as method overloading

void m2(int a)

void m2(char ch)

If the class contains more than one method with the same name and same type but they differ in sequence or order of passing the arguments are called as method overloading

void m3(int a, float b)

void m3(float b, int a)

**Ex:**

**Class Test**

{

**Public void m1(int i) // method 1**

{  
}

**Public void m1(long l) // method 2**

{  
}

}

**Example1:**

```
// Demo on Method OverLoading [ Number of Arguments]
class Student
{
    int a,b,c,w;
    void add(int x,int y)
    {
        a=x; b=y; c=a+b;
        System.out.println("c value is:"+c);
    }
    void add(int x,int y,int z)
    {
        a=x; b=y; c=z; w=a+b+c;
        System.out.println("W value is:"+w);
    }
}
class Demo_CTP
{
    public static void main(String arg[])
    {
        Student s1=new Student();
        s1.add(10,20);
        s1.add(10,20,30);
    }
}
```

C:\Users\RAVI KANTH\Desktop>javac Demo\_CTP.java

C:\Users\RAVI KANTH\Desktop>java Demo\_CTP

c value is:30

w value is:60

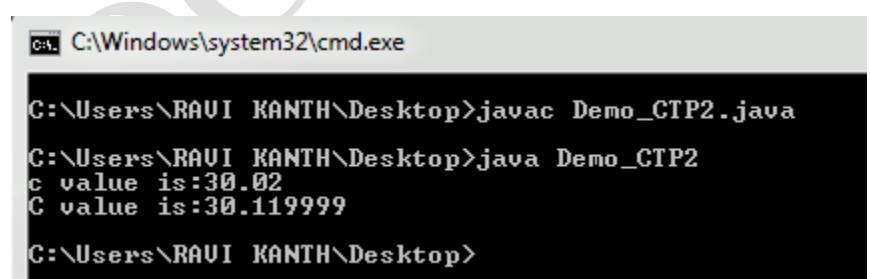
**Example2:**

```
// Demo on Method OverLoading [ Type of Arguemnts ]
class Student
{
    float a,b,c;
    void add(int x,float y)
    {
        a=x; b=y; c=a+b;
        System.out.println("c value is:"+c);
    }
    void add(int x,int y)
    {
        a=x; b=y; c=a+b;
        System.out.println("C value is:"+c);
    }
}
class Demo_CTP1
{
    public static void main(String arg[])
    {
        Student s1=new Student();
        s1.add(10,20.02f);
        s1.add(10,20);
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac Demo_CTP1.java
C:\Users\RAVI KANTH\Desktop>java Demo_CTP1
c value is:30.02
C value is:30.0
```

**Example3:**

```
// Demo on Method OverLoading
// [ Sequence of Arguemnts]
class Student
{
    float a,b,c;
    void add(int x,float y)
    {
        a=x; b=y; c=a+b;
        System.out.println("c value is:"+c);
    }
    void add(float x,int y)
    {
        a=x; b=y; c=a+b;
        System.out.println("C value is:"+c);
    }
}
class Demo_CTP2
{
    public static void main(String arg[])
    {
        Student s1=new Student();
        s1.add(10,20.02f);
        s1.add(10.12f,20);
    }
}
```

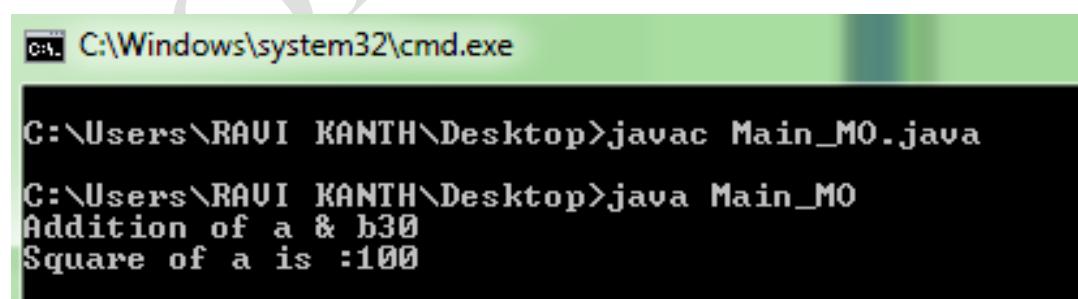


The screenshot shows a Windows Command Prompt window with the following text:

```
C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac Demo_CTP2.java
C:\Users\RAVI KANTH\Desktop>java Demo_CTP2
c value is:30.02
C value is:30.119999
C:\Users\RAVI KANTH\Desktop>
```

**Example4:**

```
// Method Overloading or Compile Time Poly or Static Binding
class Test
{
    int a,b;
    void calc(int x)
    {
        a=x;
        System.out.println("Square of a is :"+(a*a));
    }
    void calc(int x,int y)
    {
        a=x;
        b=y;
        System.out.println("Addition of a & b"+(a+b));
    }
}
class Main_M0
{
    public static void main(String ar[])
    {
        Test obj=new Test();
        obj.calc(10,20);
        obj.calc(10);
    }
}
```



The screenshot shows a Windows command prompt window. The title bar says "C:\Windows\system32\cmd.exe". The command line shows "C:\Users\RAVI KANTH\Desktop>javac Main\_M0.java". After compilation, the command line shows "C:\Users\RAVI KANTH\Desktop>java Main\_M0". The output of the program is displayed below the command line, showing the addition of 10 and 20, and the square of 10.

```
C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac Main_M0.java
C:\Users\RAVI KANTH\Desktop>java Main_M0
Addition of a & b30
Square of a is :100
```

## Over Riding or Redefining

Whatever methods parent has by default available to the child through inheritance.  
Sometimes child may not satisfy with parent method implementation.

Then child is allowed to redefine that method based on its requirement.

This process is called as Overriding

The parent class method which is overridden is called Overridden method

The child class method which is overriding is called Overriding method

### **Example:1**

#### **Example Method Overriding in Java**

```
class Walking
{
    void walk()
    {
        System.out.println("Man walking fastly");
    }
}
class OverridingDemo
{
    public static void main(String args[])
    {
        Man obj = new Man();
        obj.walk();
    }
}
```

### **Output**

Man walking

Problem is that I have to provide a specific implementation of walk() method in subclass that is why we use method overriding.

### Example of method overriding in Java

In this example, we have defined the walk method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

#### Example

```
class Walking
{
    void walk()
    {
        System.out.println("Man walking fastly");
    }
}

class Man extends walking
{
    void walk()
    {
        System.out.println("Man walking slowly");
    }
}

class OverridingDemo
{
    public static void main(String args[])
    {
        Man obj = new Man();
        obj.walk();
    }
}
```

**Output**

Man walking slowly

**Example:2**

```
class parent
{
    public void property( )
    {
        System.out.println("cash+land+gold");
    }

    public void study()
    {
        System.out.println(" U must study only B.Tech");
    }
}

class child extends parent // Child class methods are called Overriding methods
{
    public void study()
    {
        System.out.println("I shall study MBA");
    }
}
```

**Example:**

```

/*Demo on Method Overriding / Run-time Polymorphism/ Late Binding
 Dynamic Binding
 Defiend as refining the functionality of superclass/parent class or
 Base calss mehtod in a sub class/child class/derived class
 Signature of both the methods should be the same
 Super class method is kniwn as Overriden method and
 sub classss method is known as Overridding method
*/
class parent // Overridden class
{
    protected int x=10;
    void m1()
    {
        System.out.println("Iam in Parent m1()");
        System.out.println("Parent x:"+x);
    }
}
class child extends parent // Overriding claas
{
    public int y=20;
    void m1()
    {
        System.out.println("Iam in child m1()");
        System.out.println("Parent x:"+x);
        System.out.println("Child y:"+y);
    }
    void m1(int a)
    {
        System.out.println("Iam in child m1()");
        System.out.println("Parent x:"+x);
        System.out.println("Child y:"+y);
    }
}
class M_OR
{
    public static void main(String aa[])
    {
        child c=new child();
        c.m1();
    }
}

```

```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac M_OR.java

C:\Users\RAVI KANTH\Desktop>java M_OR
I am in child m1()
Parent x:10
Child y:20

C:\Users\RAVI KANTH\Desktop>
```

What is the difference between OverLoading & Over Ridding

| <b>Property</b>                             | <b>OverLoading</b>                 | <b>OverRiding</b>   |
|---|------------------------------------|---|
| <b>Method Names</b>                         | Must be Same                       | Must be Same  |
| <b>Arguments Types</b>                      | Must be different[ Atleast Order ] | Must be Same [ Including Order]   |
| <b>Method Signatures</b>                    | Must be different                  | Must be Same  |
| <b>Return Type</b>                          | No Restrictions                    | Must be same but this rule is applicable until 1.4version only. From 1.5version onwards Co-variant return types are allowed |
| <b>private ,static &amp; final methods</b>  | Can be overloaded                  | Cannot be overridden  |
| <b>Access Modifiers</b>                     | No Restrictions                    | We cannot reduce the scope of Access Modifiers but we can increase the scope of Access Modifiers                            |
| <b>Throws clause[ Exceptional Handling]</b> | No Restrictions                    | If child class method throws any checked exception compulsory parent class method   |

|                          |   |   |
|--------------------------|---|---|
|                          |   | should throw the same checked exception as its parent otherwise we will get compile time error but there are no restrictions for Unchecked Exceptions |
| <b>Method Resolution</b> | Always takes care by Compiler, based on reference type              | Always takes care by JVM based on Run-time Object   |
| <b>Also Known As</b>     | Compile time Polymorphism/<br>Static Polymorphism/<br>Early Binding | Run-time Polymorphism/<br>Dynamic Polymorphism/<br>Late Binding   |

Access Modifiers:

| Parent | Public | Default                  | Protected        | Private   |
|--------|--------|--------------------------|------------------|-----------|
| Child  | Public | Default/protected/public | Protected/public | No Access |

**Exception:**

Parent : public void m1() throws IOException/Exception

Child: public void m1() throws IOException

**Note:**

- In Overloading we have to check only method names( must be same) and argument types (must be different) except this the remaining like return types, access modifiers etc., are not required to check.
- But in Overriding everything we have to check like method names arguments types, return types, access modifiers etc.

**UNIT-III****Topics to be covered:****Final Keyword****Final keyword in java**

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level.

In java language final keyword can be used in following way.

1. **final at variable** : final variable value cannot be changed during the scope of the Program
2. **final at method** : final method cannot be overridden in other class
3. **final at class** : final class cannot be inherited

**1. Final at variable**

Final keyword is used to make a variable as a constant. This is similar to constants in other language. A variable declared with the final keyword cannot be modified or changed by the program after initialization.

This is useful to universal constants, such as "PI". If you still try to change the value it may result in Compilation Error.

**Final Keyword in java Example**

```
public class Circle
{
    public static final double PI=3.14159;

    public static void main(String[] args)
    {
        System.out.println(PI);
    }
}
```

}

```
// Demo on final Keyword with final variable

class final_var
{
    final int i=10;
    final_var()
    {
        i=i+20;
    }
}
class Main_final
{
    public static void main(String a[])
    {
        final_var f1=new final_var();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac Main_final.java
Main_final.java:9: error: cannot assign a value to final variable i
          i=i+20;
               ^
1 error
```

## 2. Final at method level

It makes a method final, meaning that sub classes can not override this method. The compiler checks and gives an error if you try to override the method.

When we want to restrict overriding, then make a method as a final.

### Example

```
public class A
{
    public void fun1()
```

```
{  
    .....  
}  
public final void fun2()  
{  
    .....  
}  
}  
class B extends A  
{  
    public void fun1()  
{  
        .....  
}  
    public void fun2()  
{  
        // it gives an error because we can not override final  
method  
    }  
}
```

```

// Demo on final Keyword with final method
class final_method // Parent class
{
    int i=10;
    final void m1()
    {
        System.out.println("Final method");
    }
}
// Overriding is not possible as m1() is declared as final
class child extends final_method // child class
{
    void m1()
    {
        System.out.println("Child method");
    }
}
class Main_final1
{
    public static void main(String a[])
    {
        final_method f1=new final_method();
    }
}

```

```

C:\Users\RAVI KANTH\Desktop>javac Main_final1.java
Main_final1.java:14: error: m1() in child cannot override m1() in final_method
    void m1()
               ^
  overridden method is final
1 error

```

### Example of final keyword at method level

#### Example

```

class Employee
{
    final void disp()
    {
        System.out.println("Hello Good Morning");
    }
}

```

```
        }
    }
class Developer extends Employee
{
    void disp()
    {
        System.out.println("How are you ?");
    }
}
class FinalDemo
{
    public static void main(String args[])
    {
        Developer obj=new Developer();
        obj.disp();
    }
}
```

**Output**

It gives an error

### 3. Final at class level

It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

#### Example

```
public final class A
{
    .....
    .....
}

public class B extends A
{
    // it gives an error, because we can not inherit final class
}
```

```
// Demo on final Keyword with final class
final class final_class // Parent class
{
    final_class()
    {
        System.out.println("Final parent class");
    }
}
class child extends final_class // child class
{
    void m1()
    {
        System.out.println("Child class");
    }
}
class Main_final2
{
    public static void main(String a[])
    {
        child c1=new child();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac Main_final2.java
Main_final2.java:9: error: cannot inherit from final final_class
class child extends final_class // child class
                           ^
1 error
```

### Example of final keyword at class level

#### Example

```
final class Employee
{
    int salary=10000;
}
class Developer extends Employee
{
    void show()
    {
        System.out.println("Hello Good Morning");
    }
}
class FinalDemo
{
    public static void main(String args[])
    {
        Developer obj=new Developer();
        obj.show();
    }
}
```

#### Output

Output:

It gives an error

## **UNIT-III**

### **Topics to be covered:**

#### **Inheritance:**

Defining a sub class, Deriving a sub class, Single Inheritance, Multilevel Inheritance, Hierarchical Inheritance Topic

#### **Inheritance in Java**

- Inheritance is the mechanism of deriving new class from old one.
- Old class is known as Super class/ Parent/ Base/ Generalized Class
- New class is known as Sub class/ Child/ Derived/ Specialized Class
- Subclass inherits all instance variables and methods defined by Super class and it also adds its own unique elements or new features.
- Inheritance uses is-a relationship in java
- We make use of extends keyword to access the parent class features in child class
- Java doesnot support Multiple Inheritance, we implement using interfaces

#### **Benefits of Inheritance:**

- Reusability of code
- Code Sharing
- Consistency in using an interface
- Application development time is very less.
- Redundancy (repetition) of the code is reducing. Hence we can get less

#### **Syntax of Inheritance**

```
class Child extends Parent
{
    //methods and fields
}
```

**Types of Inheritance:****• Single Inheritance**

```
public class A  
{  
.....  
}  
public class B extends A  
{  
.....  
}
```

**• Multi-Level Inheritance**

```
public class A  
{  
.....  
}  
public class B extends A  
{  
.....  
}  
public class C extends B  
{  
.....  
}
```

**• Hierarchical Inheritance**

```
public class A  
{  
.....  
}  
public class B extends A  
{  
.....  
}  
public class C extends A
```

```
{  
.....  
}
```

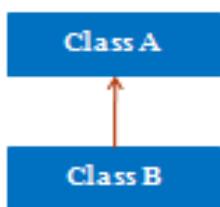
- **Hybrid Inheritance**

[Combination of more than one type either Single/Multi-level/Hierarchical]

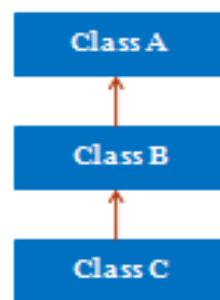
```
public class A  
{  
.....  
}  
public class B extends A  
{  
.....  
}  
public class C extends A  
{  
.....  
}  
public class D extends B  
{  
.....  
}
```

- **Multiple Inheritance [ Implemented using Interfaces]**

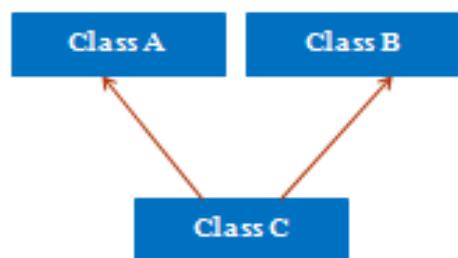
## Types of Inheritance



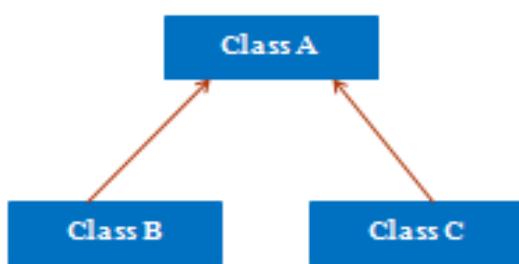
Single



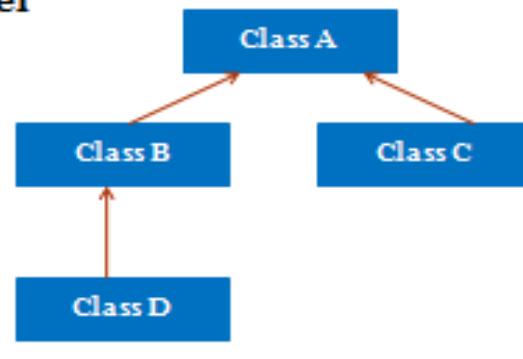
Multilevel



Multiple



Hierarchical

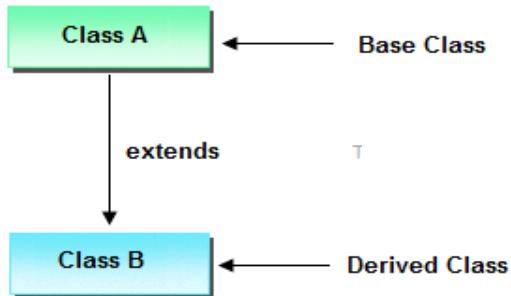


Hybrid

OOPS THRU

## 1. Single inheritance

In single inheritance there exists single base class and single derived class.



```

// Demo on Single Inheritance
class A
{
    A()
    {
        System.out.println("Hai");
    }
    void m1()
    {
        System.out.println("Class A");
    }
}
class B extends A
{
    B()
    {
        System.out.println("Hello");
    }
    void m2()
    {
        System.out.println("Class B");
    }
}
public class Main
{
    public static void main(String a[])
    {
        B b=new B();
        b.m2();
        b.m1();
    }
}
  
```

```
C:\Users\RAVI KANTH\Desktop>java Main
Hai
Hello
Class B
Class A
```

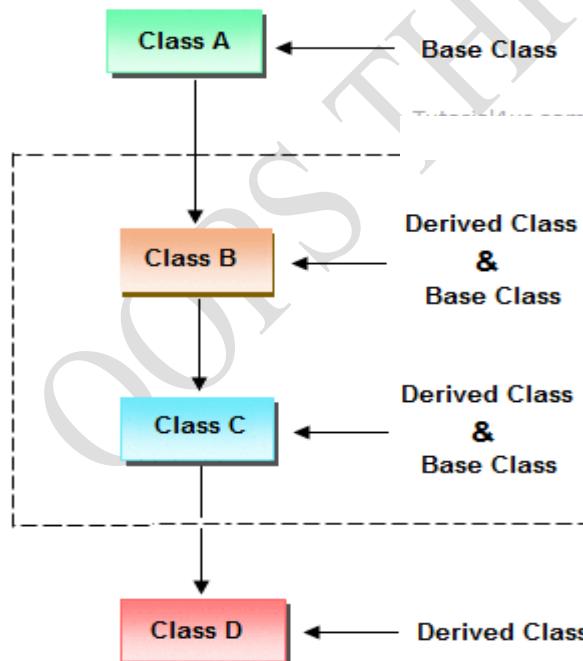
## **2. Multilevel inheritances in Java**

It is a ladder or hierarchy of single level inheritance. It means if Class A is extended by Class B and then further Class C extends Class B then the whole structure is termed as Multilevel Inheritance. Multiple classes are involved in inheritance, but one class extends only one. The lowermost subclass can make use of all super classes' members.

**Single base class + single derived class + multiple intermediate base classes.**

### **Intermediate base classes**

An intermediate base class is one in one context with access derived class and in another context same class access base class.



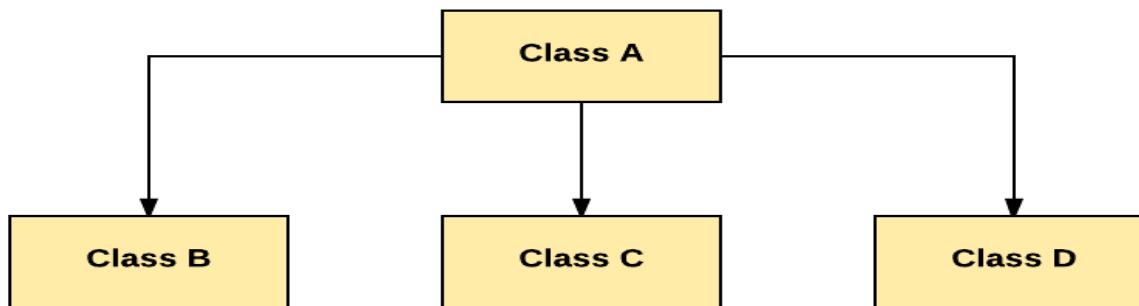
Hence all the above three inheritance types are supported by both classes and interfaces.

```
// Demo on Multi Level | Inheritance
class A
{
    A()
    {
        System.out.println("Class A");
    }
    void m1()
    {
        System.out.println("Class A m1");
    }
}
class B extends A
{
    B()
    {
        System.out.println("Class B");
    }
    void m2()
    {
        System.out.println("Class B m2 ");
    }
}
class C extends B
{
    C()
    {
        System.out.println("Class C");
    }
    void m3()
    {
        System.out.println("Class C m3");
    }
}
public class Main
{
    public static void main(String a[])
    {
        C c=new C();
        c.m1();
        c.m2();
        c.m3();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac Main.java
C:\Users\RAVI KANTH\Desktop>java Main
Class A
Class B
Class C
Class A m1
Class B m2
Class C m3
```

### **3. Hierarchical inheritance**

Hierarchical Inheritance is one in which there exists single base class and n number of derived classes.



```
// Demo on Hierarchical Inheritance
class A
{
    A()
    {
        System.out.println("Class A");
    }
    void m1()
    {
        System.out.println("Class A m1");
    }
}
class B extends A
{
    B()
    {
        System.out.println("Class B");
    }
    void m2()
    {
        System.out.println("Class B m2 ");
    }
}
class C extends A
{
    C()
    {
        System.out.println("Class C");
    }
    void m3()
    {
        System.out.println("Class C m3");
    }
}
```

```
public class Main
{
    public static void main(String a[])
    {
        C c=new C();
        c.m1();
        B b=new B();
        b.m1();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>java Main
Class A
Class C
Class A m1
Class A
Class B
Class A m1
```

```
// Demo on Hierarchical Inheritance
class A
{
    A()
    {
        System.out.println("Class A");
    }
    void m1()
    {
        System.out.println("Class A m1");
    }
}
class B
{
    B()
    {
        System.out.println("Class B");
    }
    void m1()
    {
        System.out.println("Class B m2 ");
    }
}
class C extends A,B // Java Doesnot support Multiple Inheritance
{
    C()
    {
        System.out.println("Class C");
    }
    void m3()
    {
        System.out.println("Class C m3");
    }
}
```

ERROR

```
class Parent
{
    int i=10; // default AM
    void m1()
    {
        System.out.println("Value of i"+i);
    }
}
class Child extends Parent
{
    void m1()
    {
        System.out.println("Value of i in Child"+i);
    }
}

class Main
{
    public static void main(String a[])
    {
        Child c1=new Child();
        c1.m1();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Main.java
C:\Users\RAVI KANTH\Desktop>java Main
Value of i in Child10
```

```

class Parent
{
    public int i=10; // default AM
    protected int j=20;
    void m1()
    {
        System.out.println("Value of i"+i);
        System.out.println("Value of j"+j);
    }
}
class Child extends Parent
{
    void m1()
    {
        System.out.println("Value of i in Child"+i);
        System.out.println("Value of j in Child"+j);
    }
}

class Main
{
    public static void main(String a[])
    {
        Child c1=new Child();
        c1.m1();
    }
}

```

```

C:\Users\RAVI KANTH\Desktop>java Main
Value of i in Child10
Value of j in Child20

```

```

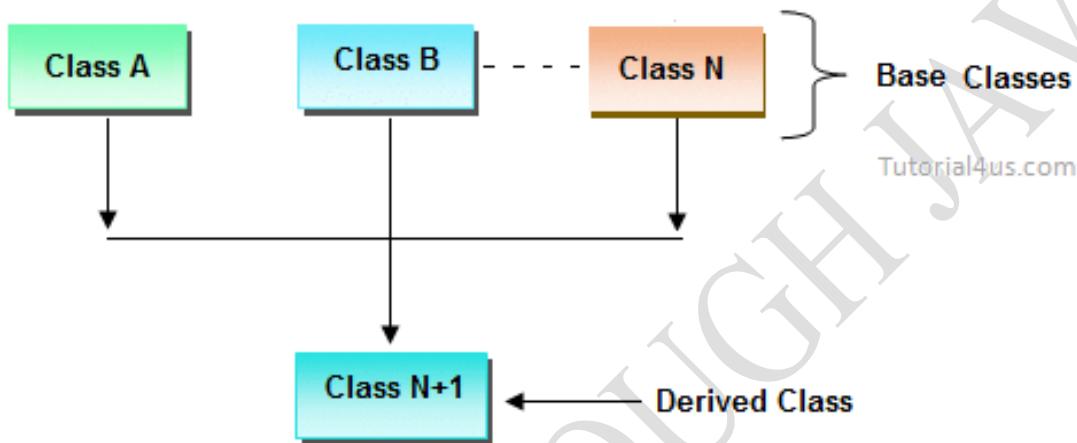
class Parent
{
    private int i=10; // default AM
    protected int j=20;
    void m1()
    {
        System.out.println("Value of i"+i);
        System.out.println("Value of j"+j);
    }
}
class Child extends Parent
{
    void m1()
    {
        System.out.println("Value of i in Child"+i);
        System.out.println("Value of j in Child"+j);
    }
}

```

```
C:\Users\RAVI KANTH\Desktop>javac Main.java
Main.java:16: error: i has private access in Parent
        System.out.println("Value of i in Child"+i);
               ^
1 error
```

#### 4. Multiple inheritance

In multiple inheritance there exist multiple classes and single derived class.

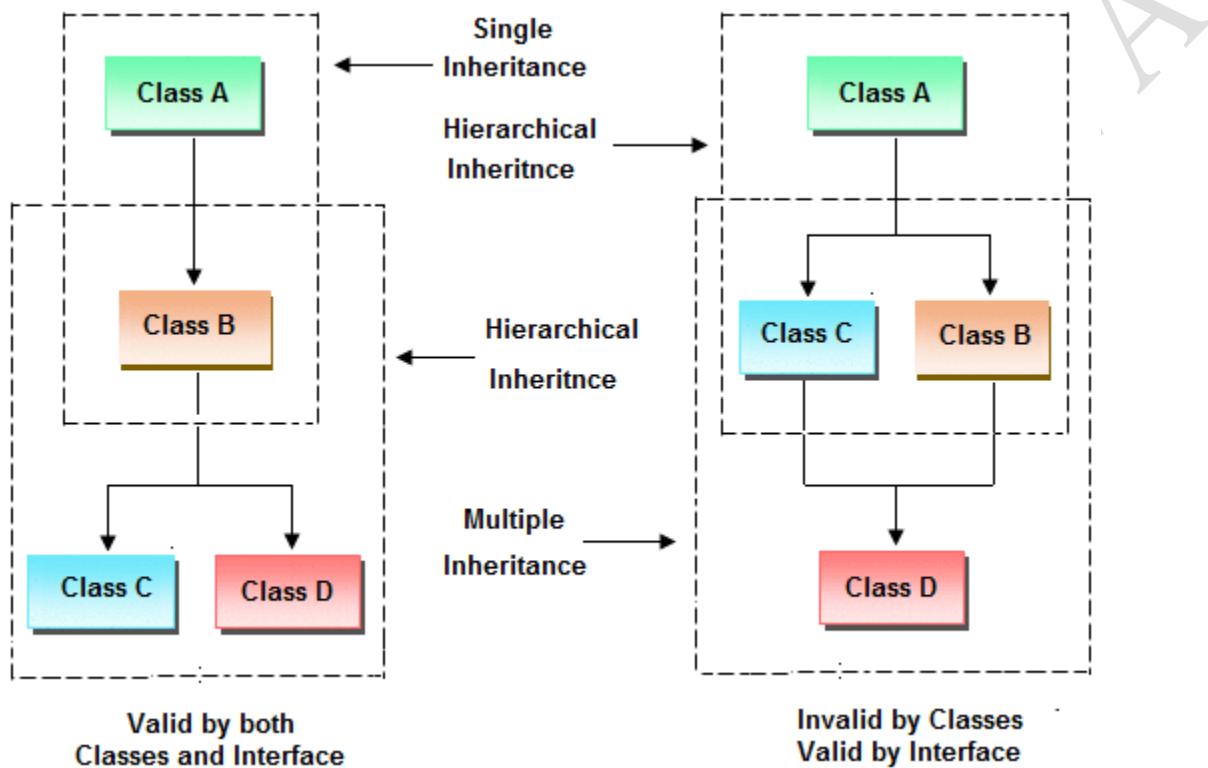


The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

## 5. Hybrid inheritance

Combination of any inheritance type

In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.



### **UNIT-III**

#### **Topics to be covered:**

##### **super Keyword**

super is a reference variable that is used to refer immediate parent class object

**Note:** Super keyword is always pointing to base class (scope outside the class) features and "this" keyword is always pointing to current class (scope is within the class) features.

##### **Uses of Super Keyword**

1. Super( ) is used to invoke immediate Parent class constructors
2. Super is used to invoke immediate Parent class method
3. Super is used to refer immediate Parent class variable

```
// Demo on Super Keyword ...calling parent class variable

class parent
{
    int i=10;
}
class child extends parent
{
    int i=20;
    void m1()
    {
        System.out.println("Value of i in child :" + i);
        System.out.println("Value of i in Parent :" + super.i);
    }
}
class Main_Super
{
    public static void main(String ar[])
    {
        child c1=new child();
        c1.m1();
    }
}
```



```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Main_Super.java

C:\Users\RAVI KANTH\Desktop>java Main_Super
Value of i in child :20
Value of i in Parent :10
```

```
// Demo on Super Keyword ...calling parent class Constructor

class parent
{
    parent()
    {
        System.out.println("Iam Parent Constructor");
    }
}

class child extends parent
{
    child()
    {
        System.out.println("Iam child Constructor");
        //super(); // Error
        // First line of constructor would be written as
        //super(); ... which is written by JVM
    }
}

class Main_Super1
{
    public static void main(String ar[])
    {
        child c1=new child();
    }
}
```

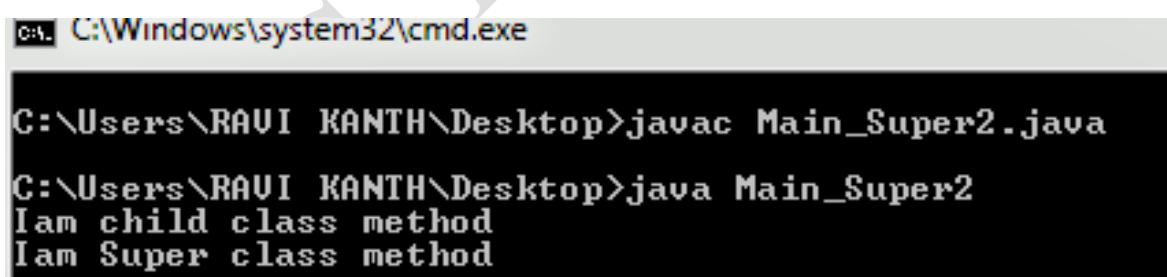
```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Main_Super1.java

C:\Users\RAVI KANTH\Desktop>java Main_Super1
Iam Parent Constructor
Iam child Constructor
```

```
// Demo on Super Keyword ...calling parent class method

class parent
{
    void m1()
    {
        System.out.println("Iam Super class method");
    }
}
class child extends parent
{
    void m1()
    {
        System.out.println("Iam child class method");
        super.m1();
    }
}
class Main_Super2
{
    public static void main(String ar[])
    {
        child c1=new child();
        c1.m1();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac Main_Super2.java
C:\Users\RAVI KANTH\Desktop>java Main_Super2
Iam child class method
Iam Super class method
```

\*\*\*\*\*

**UNIT-III****Topics to be covered:**

**Interfaces** - Interfaces vs Abstract classes, defining an interface, implementing interfaces, accessing implementations through interface references, extending interfaces.

**Interface in Java**

- In Java programming language, an interface is a reference type, similar to a class that can contain only constants, method signatures and nested types.
- There are no method bodies. Hence Interfaces cannot be instantiated- they can only be implemented by classes or extended by other interfaces
- Interfaces are acting as the reference of an object
- An interface is similar to a class
- An interface contains number of data members and member functions
- All the member data declared in an interface are static and final by default
- All the members functions declared in an interface should be abstract
- An interface should be created by using keyword “interface”
- An interface can be implemented to a class by using keyword “implements”
- More than one interface can be implemented to a class
- When ever you implement an interface to a class, all the member functions declared in the interfaces should be defined in the class otherwise that class becomes an abstract class.

**Use of Interface:**

- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.

**Properties of Interface:**

- It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.

- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

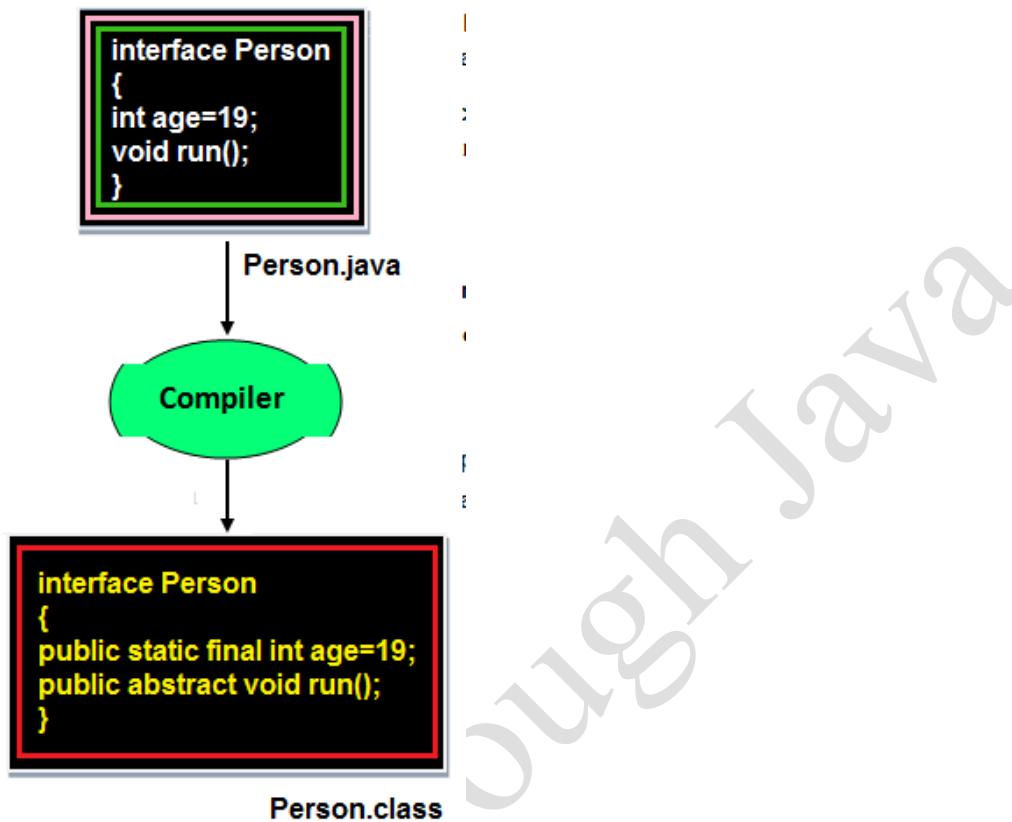
### How interface is different from class:

|                        | <b>Class</b>        | <b>Interface</b>            |
|------------------------|---------------------|-----------------------------|
| <b>Fields</b>          | Normal or Constants | Only Constants              |
| <b>Methods</b>         | With complete body  | Abstract methods [ no body] |
| <b>Object</b>          | We can create       | We cannot create object     |
| <b>Access Modifier</b> | We can use all AM   | By default it is public     |
| <b>Usage</b>           | Extends             | Implements                  |

### NOTE:

- You cannot instantiate an interface. It does not contain any constructors.
- All methods in an interface are abstract.
- Interface cannot contain instance fields. Interface only contains public static final variables.
- Interface is cannot extended by a class; it is implemented by a class.
- Interface can extend multiple interfaces. It means interface support multiple inheritance

### Behavior of compiler with Interface program



- By default compiler adds public static final before any variable and public abstract before any method.
- Because **Interface** is design for fulfill universal requirements and to achieve fully abstraction.

### Declaring Interfaces:

The **interface** keyword is used to declare an interface.

**interface interface\_name**

### Example

```

interface Person
{
    datatype variablename=value;
        //Any number of final, static fields [ constant]
    returntype methodname(list of parameters or no parameters);
        //Any number of abstract method declarations
}
    
```

**Example**

public static final datatype variable name=value; ----> for data member

public abstract returntype methodname(parameters)---> for method

Ex: public int add(int x,int y);

// No need to write Body , as by default they are Abstract

**How to use implements keyword:**

```
class B implements A // here A is an interface and B is a class
{
.....
}
```

**Implementing Interfaces:**

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

**Example**

```
interface Person
{
    void run();
}

class Employee implements Person
{
    public void run()
    {
        System.out.println("Run fast");
    }
}
```

**When to use abstract and when we use Interface**

- If we do not know about any things about implementation just we have requirement specification then we should be go for **Interface**
- If we are talking about implementation but not completely (partially implemented) then we should be go for **abstract**

### Rules for implementation interface

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

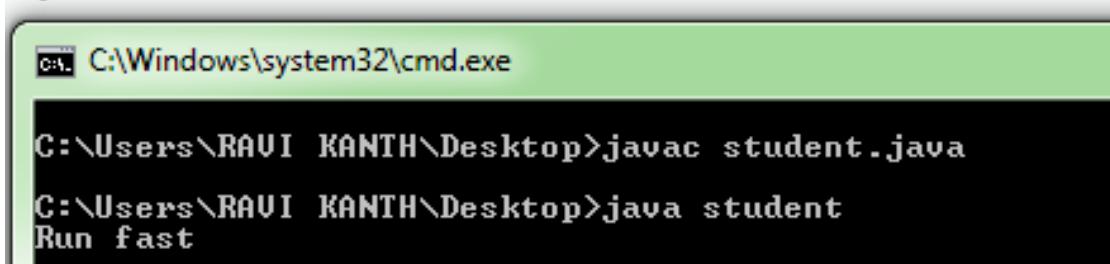
### Difference between Abstract class and Interface

|   | <b>Abstract class</b>   | <b>Interface</b>   |
|---|---|--|
| 1 | It is collection of abstract method and concrete methods.   | It is collection of abstract method.   |
| 2 | There properties can be reused commonly in a specific application.  | There properties commonly usable in any application of java environment.                                     |
| 3 | It does not support multiple inheritance.   | It support multiple inheritance.   |
| 4 | Abstract class is preceded by abstract keyword.   | It is preceded by Interface keyword.   |
| 5 | The default access specifier of abstract class methods are default.   | There default access specifier of interface method are public.   |
| 6 | These class properties can be reused in other class using extend keyword.   | These properties can be reused in any other class using implements keyword.                                  |
| 7 | Inside abstract class we can take constructor.  | Inside interface we can not take any constructor.  |
| 8 | For the abstract class there is no restriction like initialization of variable at the time of variable declaration. | For the interface it should be compulsory to initialization of variable at the time of variable declaration. |

|    |  |   |
|----|--|---|
| 9  | There are no any restriction for abstract class variable.  | For the interface variable can not declare variable as private, protected, transient, volatile.                       |
| 10 | There are no any restriction for abstract class method modifier that means we can use any modifiers. | For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized. |

### Example of Interface

```
interface Person
{
    void run(); // abstract method
}
class student implements Person
{
    public void run()
    {
        System.out.println("Run fast");
    }
    public static void main(String args[])
    {
        student obj = new student();
        obj.run();
    }
}
```



C:\Windows\system32\cmd.exe

```
C:\Users\RAVI KANTH\Desktop>javac student.java
C:\Users\RAVI KANTH\Desktop>java student
Run fast
```

**Ex:**

```
// demo on interface
interface A
{
    public int a=10,b=20; // By default they are public,static and final
    public int addNum(int x,int y);
    public int subNum(int x,int y);
}
class B implements A
{
    public int addNum(int x, int y)
    {
        return x+y;
    }
    public int subNum(int x, int y)
    {
        return x-y;
    }
}
class Test
{
    public static void main(String aa[])
    {
        A a1=new B();
        System.out.println(a1.a+"\t"+a1.b);
        System.out.println(A.a+"\t"+A.b);
        //a1.a=100; // Compilation error , the member data declared in an interface are final
        System.out.println("Sum="+a1.addNum(10,20));
        System.out.println("Diff="+a1.subNum(10,20));
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac Test.java
C:\Users\RAVI KANTH\Desktop>java Test
10      20
10      20
Sum=30
Diff=-10
```

**Ex: Multiple Inheritance through Interfaces**

Interface I3 extends I1,I2

```
{  
}  
}
```

Class Impl implements I3

```
{  
  
// should define all the methods of I1,I2,I3 interfaces  
  
}
```

## Multiple Inheritance using interface

### Example

```
interface Developer
{
    void disp();
}

interface Manager
{
    void show();
}

class Employee implements Developer, Manager
{
    public void disp()
    {
        System.out.println("Hello Good Morning");
    }
    public void show()
    {
        System.out.println("How are you ?");
    }
    public static void main(String args[])
    {
        Employee obj=new Employee();
        obj.disp();
        obj.show();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Employee.java
C:\Users\RAVI KANTH\Desktop>java Employee
Hello Good Morning
How are you ?
```

### Possible Combinations

```
class A
{
    class B
    {
    }
}

class A
{
    interface B
    {
    }
}

interface A
{
    interface B
    {
    }
}
```

```
interface A
{
    class B
    {
    }
}
```

## Note:

1. The interface which is declared inside a class is always static whether we are declaring or not
2. The interface which is declared inside interface is always public and static whether we are declaring or not
3. The class which is declared inside interface is always public and static whether we are declaring or not

## TASK:

- Write a Java Program to create an abstract class Animal with name and species as fields and initialize the values through constructor and create a concrete method eat and abstract method sound. Now create a sub-class animal and call the features of Animal class
- Write a Java Program to create an interface by named Calculator and declare abstract methods add,subtract,multiply,divide and implement all the abstract methods of interface in Normal\_Calcualtor class by creating indirect object of interface

\*\*\*\*\*

## **UNIT-III**

### **Topics to be covered:**

**Interfaces** - Interfaces vs Abstract classes, defining an interface, implementing interfaces, accessing implementations through interface references, extending interfaces.

### **Interface in Java**

- In Java programming language, an interface is a reference type, similar to a class that can contain only constants, method signatures and nested types.
- There are no method bodies. Hence Interfaces cannot be instantiated- they can only be implemented by classes or extended by other interfaces
- Interfaces are acting as the reference of an object
- An interface is similar to a class
- An interface contains number of data members and member functions
- All the member data declared in an interface are static and final by default
- All the members functions declared in an interface should be abstract
- An interface should be created by using keyword “interface”
- An interface can be implemented to a class by using keyword “implements”
- More than one interface can be implemented to a class
- When ever you implement an interface to a class, all the member functions declared in the interfaces should be defined in the class otherwise that class becomes an abstract class.

### **Use of Interface:**

- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.

### **Properties of Interface:**

- It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.

- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

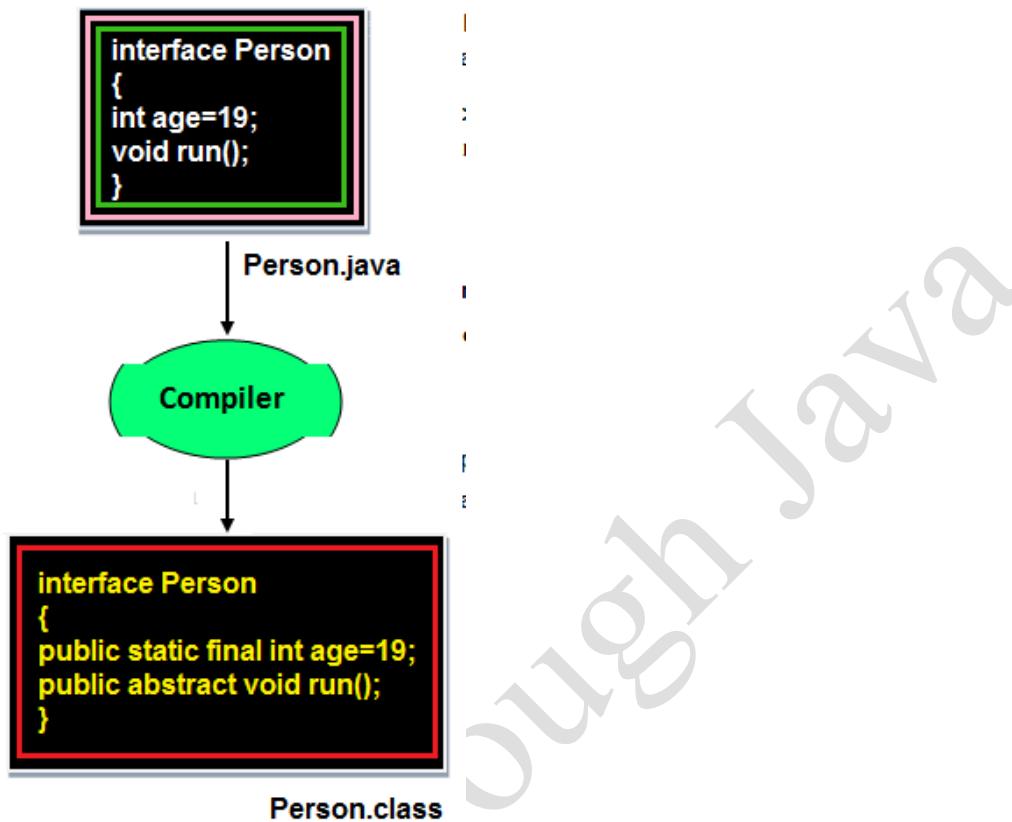
### How interface is different from class:

|                        | <b>Class</b>        | <b>Interface</b>            |
|------------------------|---------------------|-----------------------------|
| <b>Fields</b>          | Normal or Constants | Only Constants              |
| <b>Methods</b>         | With complete body  | Abstract methods [ no body] |
| <b>Object</b>          | We can create       | We cannot create object     |
| <b>Access Modifier</b> | We can use all AM   | By default it is public     |
| <b>Usage</b>           | Extends             | Implements                  |

### NOTE:

- You cannot instantiate an interface. It does not contain any constructors.
- All methods in an interface are abstract.
- Interface cannot contain instance fields. Interface only contains public static final variables.
- Interface is cannot extended by a class; it is implemented by a class.
- Interface can extend multiple interfaces. It means interface support multiple inheritance

### Behavior of compiler with Interface program



- By default compiler adds public static final before any variable and public abstract before any method.
- Because **Interface** is design for fulfill universal requirements and to achieve fully abstraction.

### Declaring Interfaces:

The **interface** keyword is used to declare an interface.

**interface interface\_name**

### Example

```

interface Person
{
    datatype variablename=value;
        //Any number of final, static fields [ constant]
    returntype methodname(list of parameters or no parameters);
        //Any number of abstract method declarations
}

```

**Example**

public static final datatype variable name=value; ----> for data member

public abstract returntype methodname(parameters)---> for method

Ex: public int add(int x,int y);

// No need to write Body , as by default they are Abstract

**How to use implements keyword:**

```
class B implements A // here A is an interface and B is a class
{
.....
}
```

**Implementing Interfaces:**

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

**Example**

```
interface Person
{
    void run();
}

class Employee implements Person
{
    public void run()
    {
        System.out.println("Run fast");
    }
}
```

**When to use abstract and when we use Interface**

- If we do not know about any things about implementation just we have requirement specification then we should be go for **Interface**
- If we are talking about implementation but not completely (partially implemented) then we should be go for **abstract**

### Rules for implementation interface

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

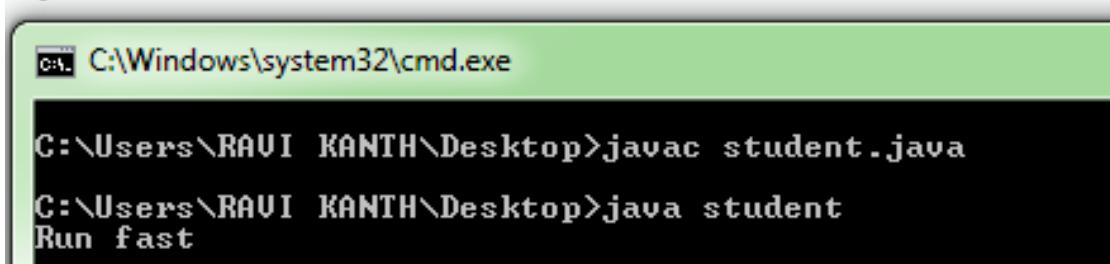
### Difference between Abstract class and Interface

|   | <b>Abstract class</b>   | <b>Interface</b>   |
|---|---|--|
| 1 | It is collection of abstract method and concrete methods.   | It is collection of abstract method.   |
| 2 | There properties can be reused commonly in a specific application.  | There properties commonly usable in any application of java environment.                                     |
| 3 | It does not support multiple inheritance.   | It support multiple inheritance.   |
| 4 | Abstract class is preceded by abstract keyword.   | It is preceded by Interface keyword.   |
| 5 | The default access specifier of abstract class methods are default.   | There default access specifier of interface method are public.   |
| 6 | These class properties can be reused in other class using extend keyword.   | These properties can be reused in any other class using implements keyword.                                  |
| 7 | Inside abstract class we can take constructor.  | Inside interface we can not take any constructor.  |
| 8 | For the abstract class there is no restriction like initialization of variable at the time of variable declaration. | For the interface it should be compulsory to initialization of variable at the time of variable declaration. |

|    |  |   |
|----|--|---|
| 9  | There are no any restriction for abstract class variable.  | For the interface variable can not declare variable as private, protected, transient, volatile.                       |
| 10 | There are no any restriction for abstract class method modifier that means we can use any modifiers. | For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized. |

### Example of Interface

```
interface Person
{
    void run(); // abstract method
}
class student implements Person
{
    public void run()
    {
        System.out.println("Run fast");
    }
    public static void main(String args[])
    {
        student obj = new student();
        obj.run();
    }
}
```



C:\Windows\system32\cmd.exe

```
C:\Users\RAVI KANTH\Desktop>javac student.java
C:\Users\RAVI KANTH\Desktop>java student
Run fast
```

**Ex:**

```
// demo on interface
interface A
{
    public int a=10,b=20; // By default they are public,static and final
    public int addNum(int x,int y);
    public int subNum(int x,int y);
}
class B implements A
{
    public int addNum(int x, int y)
    {
        return x+y;
    }
    public int subNum(int x, int y)
    {
        return x-y;
    }
}
class Test
{
    public static void main(String aa[])
    {
        A a1=new B();
        System.out.println(a1.a+"\t"+a1.b);
        System.out.println(A.a+"\t"+A.b);
        //a1.a=100; // Compilation error , the member data declared in an interface are final
        System.out.println("Sum="+a1.addNum(10,20));
        System.out.println("Diff="+a1.subNum(10,20));
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac Test.java
C:\Users\RAVI KANTH\Desktop>java Test
10      20
10      20
Sum=30
Diff=-10
```

**Ex: Multiple Inheritance through Interfaces**

Interface I3 extends I1,I2

```
{  
}  
}
```

Class Impl implements I3

```
{  
  
// should define all the methods of I1,I2,I3 interfaces  
  
}
```

## Multiple Inheritance using interface

### Example

```
interface Developer
{
    void disp();
}

interface Manager
{
    void show();
}

class Employee implements Developer, Manager
{
    public void disp()
    {
        System.out.println("Hello Good Morning");
    }
    public void show()
    {
        System.out.println("How are you ?");
    }
    public static void main(String args[])
    {
        Employee obj=new Employee();
        obj.disp();
        obj.show();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Employee.java
C:\Users\RAVI KANTH\Desktop>java Employee
Hello Good Morning
How are you ?
```

### Possible Combinations

```
class A
{
    class B
    {
    }
}

class A
{
    interface B
    {
    }
}

interface A
{
    interface B
    {
    }
}
```

```
interface A
{
    class B
    {
    }
}
```

## Note:

1. The interface which is declared inside a class is always static whether we are declaring or not
2. The interface which is declared inside interface is always public and static whether we are declaring or not
3. The class which is declared inside interface is always public and static whether we are declaring or not

## TASK:

- Write a Java Program to create an abstract class Animal with name and species as fields and initialize the values through constructor and create a concrete method eat and abstract method sound. Now create a sub-class animal and call the features of Animal class
- Write a Java Program to create an interface by named Calculator and declare abstract methods add,subtract,multiply,divide and implement all the abstract methods of interface in Normal\_Calcualtor class by creating indirect object of interface

\*\*\*\*\*

**UNIT-III****Topics to be covered:**

Abstract methods and Abstract classes

**Abstract class in Java**

- A class that is declared with abstract keyword, is known as **abstract class**.
- An abstract class is just like a normal class but it will be an incomplete class
- An abstract class is one which contains Data Members and Member functions  
[containing some defined method and some undefined method].
- In java programming undefined methods are known as un-Implemented, or abstract method.
- Abstract classes cannot be instantiated, which means we cannot create a direct object
- If a class consists of at least one abstract method then it is known as abstract class
- We need to create a child class for an abstract class in that child class we should define all the abstract methods, else that class will become an abstract class
- We can invoke the abstract class members by creating the object of child class

**Syntax**

```
abstract class className
{
    .....
}
```

**Example**

```
abstract class A
{
    .....
}
```

If any class have any abstract method then that class become an abstract class.

**Example**

```
class Vechile
{
    abstract void Bike();
}
```

Class Vechile is an abstract class because it have abstract Bike() method.

**Make class as abstract class**

To make the class as abstract class, whose definition must be preceded by a abstract keyword.

**Example**

```
abstract class Vechile
{
    .....
}
```

**Abstract method**

An abstract method is one which contains only declaration or prototype but it never contains body or definition.

In order to make any undefined method as abstract whose declaration is must be predefined by abstract keyword.

**Syntax**

```
abstract ReturnType methodName(List of formal parameter);
```

**Example**

```
abstract void sum();
abstract void diff(int a, int b);
```

**Example of abstract class**

```

abstract class Vechile
{
    abstract void speed(); // abstract method
}
class Bike extends Vechile
{
    void speed()
    {
        System.out.println("Speed limit is 40 km/hr..");
    }
    public static void main(String args[])
    {
        Vechile obj = new Bike(); //indirect object creation
        obj.speed();
    }
}

```

```

C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac Bike.java
C:\Users\RAVI KANTH\Desktop>java Bike
Speed limit is 40 km/hr..

```

**Create an Object of abstract class**

An object of abstract class cannot be created directly, but it can be created indirectly. It means you can create an object of abstract derived class.

**Example**

```
Vechile obj = new Bike(); //indirect object creation
```

**Important Points about abstract class**

- Abstract class of Java always contains common features.
- Every abstract class participates in inheritance.
- Abstract class definitions should not be made as final because abstract classes always participate in inheritance classes.

- An object of abstract class cannot be created directly, but it can be created indirectly.
- All the abstract classes of Java makes use of polymorphism along with method overriding for business logic development and makes use of dynamic binding for execution logic.

**Example:**

```
// Demo on Static Binding/Compile-time Binding
class A
{
    static void m1()
    {
        System.out.println("HAI");
    }
}
class B extends A
{
    static void m1()
    {
        System.out.println("Bye");
    }
}
class Static_Binding
{
    public static void main(String aa[])
    {
        A a=new B();
        a.m1();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac ma.java
C:\Users\RAVI KANTH\Desktop>java Static_Binding
HAI
C:\Users\RAVI KANTH\Desktop>
```

```
// Demo on Dynamic Binding/Run-time Binding
class A
{
    void m1()
    {
        System.out.println("HAI");
    }
}
class B extends A
{
    void m1()
    {
        System.out.println("Bye");
    }
}
class Static_Binding
{
    public static void main(String aa[])
    {
        A a=new B();
        a.m1();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac ma.java
```

```
C:\Users\RAVI KANTH\Desktop>java Static_Binding
Bye
```

**Example of abstract class having method body**

```
abstract class Vechile
{
    abstract void speed();
    void mileage()
    {
        System.out.println("Mileage is 60 km/ltr..");
    }
}
class Bike1 extends Vechile
{
    void speed()
    {
        System.out.println("Speed limit is 40 km/hr..");
    }
    public static void main(String args[])
    {
        Vechile obj = new Bike1();
        obj.speed();
        obj.mileage();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Bike1.java
C:\Users\RAVI KANTH\Desktop>java Bike1
Speed limit is 40 km/hr..
Mileage is 60 km/ltr..
```

**Example of abstract class having constructor, data member, methods**

```

abstract class Vechile
{
    int limit=40;
    Vechile()
    {
        System.out.println("constructor is invoked");
    }
    void getDetails()
    {
        System.out.println("it has two wheels");
    }
    abstract void run();
}
class Bike2 extends Vechile
{
    void run()
    {
        System.out.println("running safely..");
    }
    public static void main(String args[])
    {
        Vechile obj = new Bike2();
        obj.run();
        obj.getDetails();
        System.out.println(obj.limit);
    }
}

```

```

C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac Bike2.java
C:\Users\RAVI KANTH\Desktop>java Bike2
constructor is invoked
running safely..
it has two wheels
40

```

\*\*\*\*\*

## **UNIT-III**

### **Topics to be covered:**

#### **Access Modifiers in Java**

- Access Modifier set the accessibility of class, package, variable, method, interface, so on..
- They also control the accessibility and restrict the accessibility
- It also tells us where to access and where not to access
- In languages like C++ public,private,protected,default are considered as Access Sepcifiers. Except that the remaining ( like static, abstract so on ) are considered as Access Modifiers.
- Where as in Java there is nothing like specifiers, all are by default considered as Access Modifiers.

#### **Keywords such as:**

public, private, protected, default, final, static, synchronized, abstract, native, strictfp, transient, volatile

Access modifiers are always used for, how to reuse the features within the package and access the package between class to class, interface to interface and interface to a class.

Access modifiers provide features accessing and controlling mechanism among the classes and interfaces.

**Note:**

If we are not using private, protected and public keywords, then JVM is by default taking as default access modifiers.

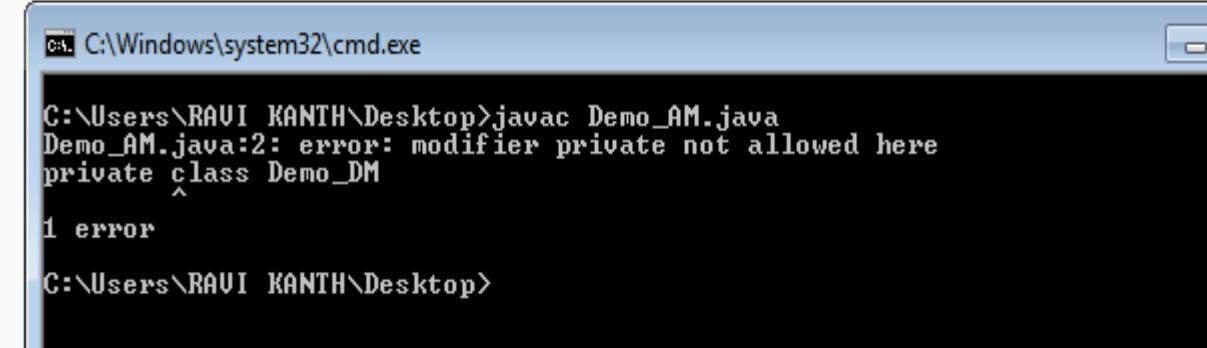
**Rules for access modifiers:**

| Access Modifiers | Within the same class | Within other class of same package | Within Derived classes of other Packages or Subclasses | External Classes of other Packages or To the outside world |
|------------------|-----------------------|------------------------------------|--|--|
| private          | YES                   | NO                                 | NO   | NO   |
| Default          | YES                   | YES                                | NO   | NO   |
| protected        | YES                   | YES                                | YES  | NO   |
| Public           | YES                   | YES                                | YES  | YES  |

**Proof:****Class must be public**

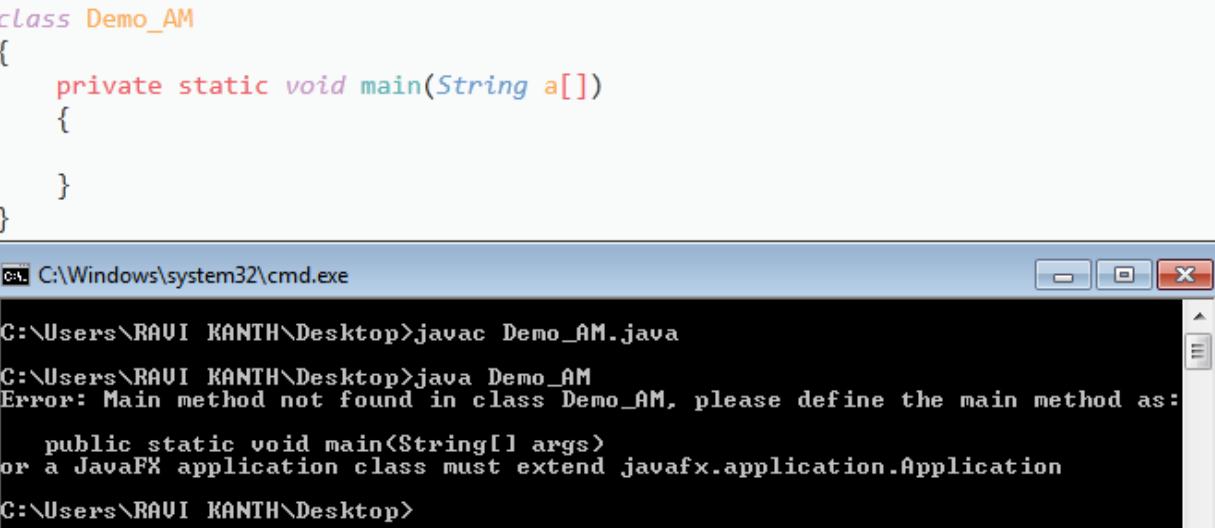
```
//Top Level Class : public , default,final abstract, strictfp are allowed
private class Demo_DM
{
}

// If you still trying to use private then we get compile time error
// C.T.E: Modifer private not allowed here
```



C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Demo\_AM.java  
Demo\_AM.java:2: error: modifier private not allowed here  
private ^ class Demo\_DM  
1 error  
C:\Users\RAVI KANTH\Desktop>

**Main method must be public**


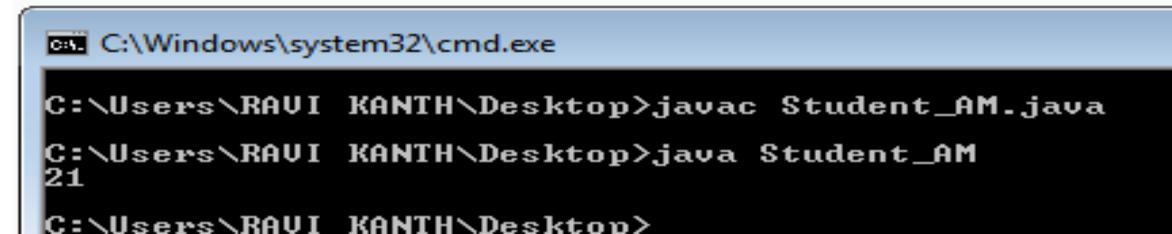
```
class Demo_AM
{
    private static void main(String a[])
    {
    }
}
```

C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac Demo\_AM.java  
C:\Users\RAVI KANTH\Desktop>java Demo\_AM  
Error: Main method not found in class Demo\_AM, please define the main method as:  
 public static void main(String[] args)  
or a JavaFX application class must extend javafx.application.Application  
C:\Users\RAVI KANTH\Desktop>

**Examples:**

```
public class Student_AM
{
    private int age;
    void display()
    {
        System.out.println(age);
    }
    public static void main(String a[])
    {
        Student_AM s1=new Student_AM();
        System.out.println(s1.age=21);
    }
}
```



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The user has navigated to their desktop directory ('C:\Users\RAVI\_KANTH\Desktop') and run two commands: 'javac Student\_AM.java' and 'java Student\_AM'. The output of the second command is '21', indicating the value of the 'age' variable.

```
C:\Windows\system32\cmd.exe
C:\Users\RAVI_KANTH\Desktop>javac Student_AM.java
C:\Users\RAVI_KANTH\Desktop>java Student_AM
21
C:\Users\RAVI_KANTH\Desktop>
```

## USE of Private

Use of Set & Get methods to access Private variables outside the class

Setter methods can also be used for assigning or modifying values of instance variables

**private:** Private members of class in not accessible anywhere in program these are only accessible within the class. Private are also called class level access modifiers.

### Example

```
class Hello
{
    private int a=20;
    private void show()
    {
        System.out.println("Hello java");
    }
}
public class Test
{
    public static void main(String args[])
    {
        Hello obj=new Hello();
        System.out.println(obj.a);
        //Compile Time Error, you can't access private data
        obj.show();
        //Compile Time Error, you can't access private methods
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac Test.java
Test.java:14: error: a has private access in Hello
        System.out.println(obj.a); //Compile Time Error, you can't access private data
                                ^
Test.java:15: error: show() has private access in Hello
        obj.show(); //Compile Time Error, you can't access private methods
                    ^
2 errors
```

**public:** Public members of any class are accessible anywhere in the program in the same class and outside of class, within the same package and outside of the package. Public are also called universal access modifiers.

### Example

```
class Hello
{
    public int a=20;
    public void show()
    {
        System.out.println("Hello java");
    }
}
public class Test
{
    public static void main(String args[])
    {
        Hello obj=new Hello();
        System.out.println(obj.a);
        obj.show();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>java Test
20
Hello java
```

**protected:** Protected members of the class are accessible within the same class and another class of the same package and also accessible in inherited class of another package. Protected are also called derived level access modifiers.

In below the example we have created two packages pack1 and pack2. In pack1, class A is public so we can access this class outside of pack1 but method show is

declared as a protected so it is only accessible outside of package pack1 only through inheritance.

### Example

```
// save A.java
package pack1;
public class A
{
    protected void show()
    {
        System.out.println("Hello Java");
    }
}

//save B.java
import pack1.A;
class B extends A
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.show();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\RAVI KANTH\Desktop>javac -d . A.java
C:\Users\RAVI KANTH\Desktop>javac A.java
C:\Users\RAVI KANTH\Desktop>javac B.java
C:\Users\RAVI KANTH\Desktop>java B
Hello Java
```

**default:** Default members of the class are accessible only within the same class and another class of the same package. The default are also called package level access modifiers.

### Example

```
//save by A.java
package pack;
class A
{
    void show()
    {
        System.out.println("Hello Java");
    }
}

//save by B.java
package pack2;
import pack1.*;
class B
{
    public static void main(String args[])
    {
        A obj = new A(); //Compile Time Error, can't access outside the package
        obj.show(); //Compile Time Error, can't access outside the package
    }
}
```

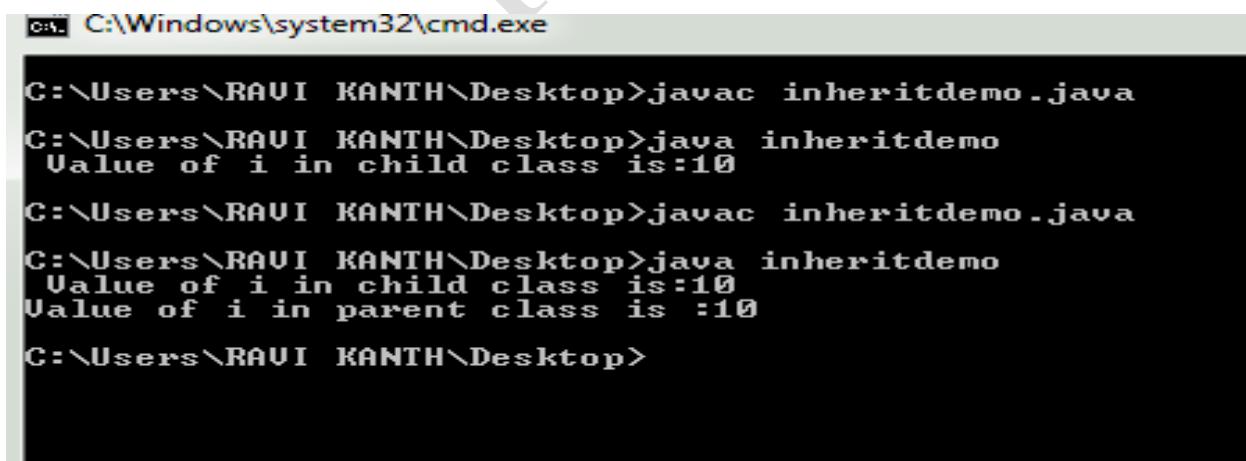
### Output

Hello Java

**Note:**

1. The modifiers which are applicable for inner classes but not for outer classes  
--- private, protected, static
2. The modifiers which are applicable for classes but not for interface are final
3. The modifiers which are applicable for classes but not for enums are final and abstract
4. The modifiers which are applicable only for methods and which we can't use anywhere else native
5. The only modifiers which are applicable for constructors are ...public,private,protected ,default
6. The only applicable modifier for local variable if final

```
// Example over Default Access modifier
class parent1
{
    int i=10;
    void m1()
    {
        System.out.println("Value of i in parent class is :" + i);
    }
}
class child1 extends parent1
{
    void m2()
    {
        System.out.println(" Value of i in child class is:" + i);
    }
}
class inheritdemo
{
    public static void main(String a[])
    {
        child1 obj1=new child1();
        obj1.m2();
        obj1.m1();
    }
}
```



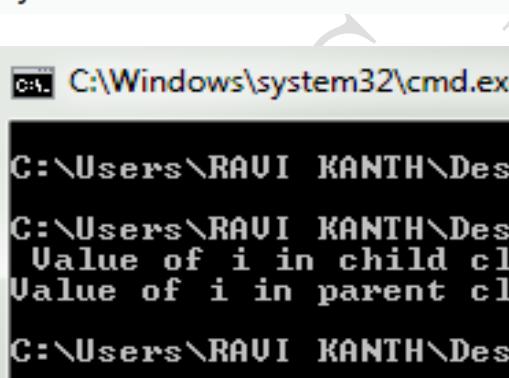
The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The user has run two separate Java sessions. In the first session, they compile the 'inheritdemo.java' file using 'javac'. In the second session, they run the compiled class 'inheritdemo'. The output of the second session shows two println statements: one from the child class 'm2()' which prints 'Value of i in child class is:10', and one from the parent class 'm1()' which prints 'Value of i in parent class is :10'.

```
C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac inheritdemo.java
C:\Users\RAVI KANTH\Desktop>java inheritdemo
Value of i in child class is:10
C:\Users\RAVI KANTH\Desktop>javac inheritdemo.java
C:\Users\RAVI KANTH\Desktop>java inheritdemo
Value of i in child class is:10
Value of i in parent class is :10
C:\Users\RAVI KANTH\Desktop>
```

```
// Example over Public Access modifier
class parent1
{
    public int i=10;
    void m1()
    {
        System.out.println("Value of i in parent class is :" + i);
    }
}
class child1 extends parent1
{
    void m2()
    {
        System.out.println(" Value of i in child class is:" + i);
    }
}
class inheritdemo
{
    public static void main(String a[])
    {
        child1 obj1=new child1();
        obj1.m2();
        obj1.m1();
    }
}
```

```
C:\Users\RAVI KANTH\Desktop>javac inheritdemo.java
C:\Users\RAVI KANTH\Desktop>java inheritdemo
Value of i in child class is:10
Value of i in parent class is :10
C:\Users\RAVI KANTH\Desktop>
```

```
// Example over Protected Access modifier
class parent1
{
    protected int i=10;
    void m1()
    {
        System.out.println("Value of i in parent class is :" + i);
    }
}
class child1 extends parent1
{
    void m2()
    {
        System.out.println(" Value of i in child class is:" + i);
    }
}
class inheritdemo
{
    public static void main(String a[])
    {
        child1 obj1=new child1();
        obj1.m2();
        obj1.m1();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\RAVI KANTH\Desktop>javac inheritdemo.java
C:\Users\RAVI KANTH\Desktop>java inheritdemo
Value of i in child class is:10
Value of i in parent class is :10
C:\Users\RAVI KANTH\Desktop>
```

```
// Example over Private Access Modifier
class parent1
{
    private int i=10;
    void m1()
    {
        System.out.println("Value of i in parent class is :" + i);
    }
}
class child1 extends parent1
{
    void m2()
    {
        System.out.println(" Value of i in child class is:" + i);
    }
}
class inheritdemo
{
    public static void main(String a[])
    {
        child1 obj1=new child1();
        obj1.m2();
        obj1.m1();
    }
}
```

## Access



C:\Windows\system32\cmd.exe

```
C:\Users\RAVI KANTH\Desktop>javac inheritdemo.java
inheritdemo.java:14: error: i has private access in parent1
    System.out.println(" Value of i in child class is:" + i);
                                         ^
1 error
C:\Users\RAVI KANTH\Desktop>
```

\*\*\*\*\*

## UNIT-III

### Topics to be covered:

**Inner classes** - uses of inner classes, local inner classes, anonymous inner classes, static inner classes, examples.

### Nested Classes

- Java language allows us to define a class within another class. Such classes are called as nested class
- In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within another is called the **nested class**, and the class that holds the inner class is called the **outer class**.

### **Syntax:**

- Following is the syntax to write a nested class. Here, the class **OuterClass** is the outer class and the class **NestedClass** is the nested class.

```
class OuterClass
```

```
{
```

```
.....
```

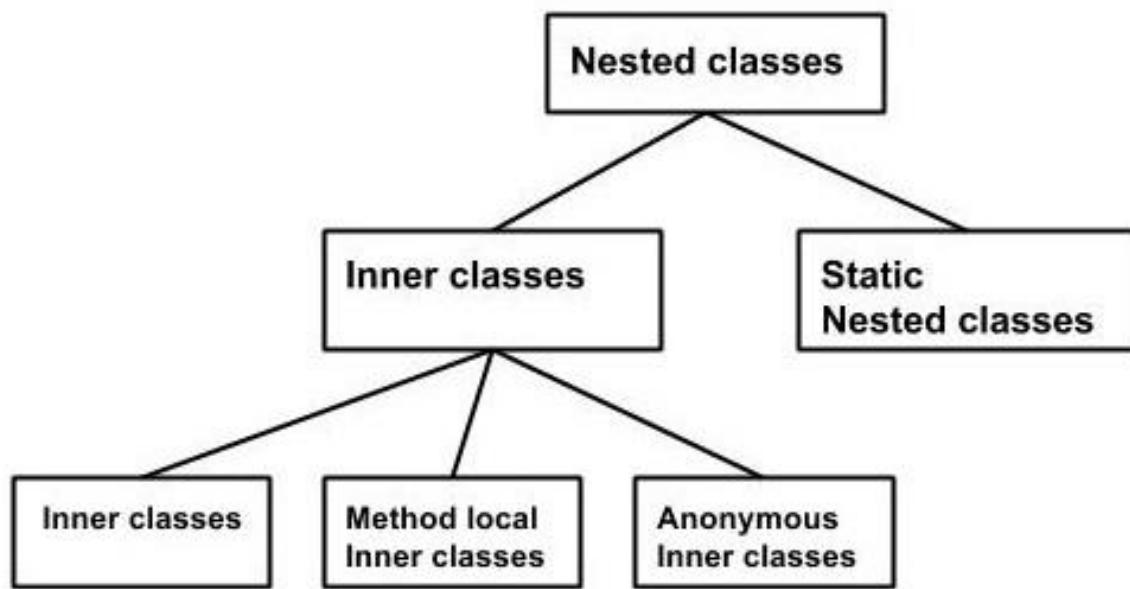
```
class NestedClass // They are also called Inner Classes
```

```
{
```

```
....
```

```
    }  
}  
  
}
```

- Nested class is a class within a class.
- Nested classes are divided into two categories: Static and Non-Static
- Nested classes that are declared static are simply called static nested classes
- Non-static nested classes are called inner classes
- **Static nested classes** – These are the static members of a class
- **Non-static nested classes** – These are the non-static members of a class.



**Uses of inner classes in java. They are as follows:**

- 1) Inner classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.
- 2) Inner classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- 3) **Code Optimization:** It requires less code to write.
- 4) Provides **Encapsulation**

**Inner class**

- Inner classes are regular members of Outer class
- We can apply all the access modifier to inner classes [ private, protected, public, default]
- With respect to non-access modifiers we can access static, abstract, final, strictfp
- Inner classes are a security mechanism in java. An outer class cannot be associated as private access modifier, whereas inner class can be made as private
- Types of Inner Classes ( non-static )
  - Inner classes
  - Method Local Inner Classes
  - Anonymous Inner Classes

```
// Demo on Inner Classes

class Outer_Class
{
    private class Inner_Class
    {
        public void print()
        {
            System.out.println("This is an Inner Class");
        }
    }
    void display_Inner()
    {
        Inner_Class inClass=new Inner_Class();
        inClass.print();
    }
}
public class InnerClasses
{
    public static void main(String args[])
    {
        // Instantiating the outer class
        Outer_Class outClass=new Outer_Class();
        // Accessing the dispaly_Inner() method
        outClass.display_Inner();
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac InnerClasses.java
C:\Users\IIIT\Desktop\RaviKanth>java InnerClasses
This is an Inner Class
```

```
// Demo on Inner Class1
class TestOuter
{
    private int data=30;
    class Inner
    {
        void msg()
        {
            System.out.println("Data is "+data);
        }
    }
    public static void main(String args[])
    {
        TestOuter obj=new TestOuter();
        TestOuter.Inner in=obj.new Inner();
        in.msg();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\IIIT\Desktop\RaviKanth>javac TestOuter.java

C:\Users\IIIT\Desktop\RaviKanth>java TestOuter
Data is 30
```

**Method Local Inner Class:**

In Java, we can write a class within a method and this will be a local type. Like local variables, the scope of the inner class is restricted within the method.

A method-local inner class can be instantiated only within the method where the inner class is defined.

```
// Demo on Method Local Inner Class
public class MethodLocalInnerClass
{
    int num=20;
    // instance method of the outer class
    public void Demo()
    {
        // method-Local inner class

        class Inner_class
        {
            public void display()
            {
                System.out.println("This is method inner class "+num);
            }
        } // end of inner class

        // assessing the inner class
        Inner_class inclass = new Inner_class();
        inclass.display();
    }
    public static void main(String args[])
    {
        MethodLocalInnerClass obj = new MethodLocalInnerClass();
        obj.Demo();
    }
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac MethodLocalInnerClass.java
C:\Users\IIIT\Desktop\RaviKanth>java MethodLocalInnerClass
This is method inner class 20
```

### Java Anonymous Inner Class

An inner class declared without a class name is known as an Anonymous Inner Class.

We declare and instantiate them at the same time

Generally used whenever we need to override the method of a class or an interface.

**Syntax:**

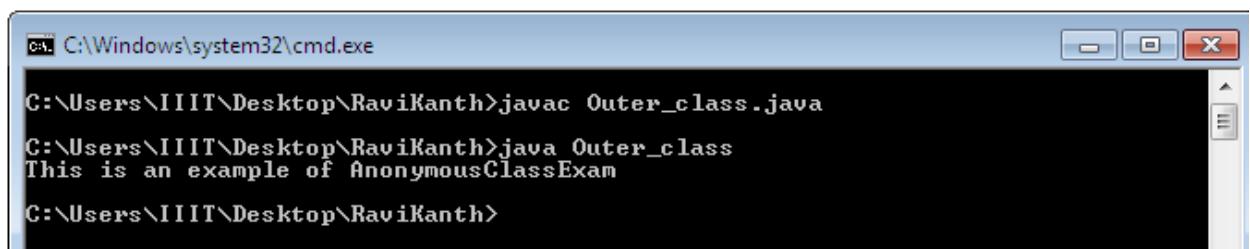
```
AnonymousInner an_inner = new AnonymousInner()  
{  
    Public void my_method()  
    {  
        .....  
        .....  
    }  
};
```

Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

## Java anonymous inner class example using class

```
// Demo over Anonymous Class
abstract class AnonymousClassExam
{
    public abstract void display();
}
public class Outer_class
{
    public static void main(String args[])
    {
        AnonymousClassExam  inclass=new AnonymousClassExam()
        {
            public void display()
            {
                System.out.println("This is an example of AnonymousClassExam");
            }
        };
        inclass.display();
    }
}
```



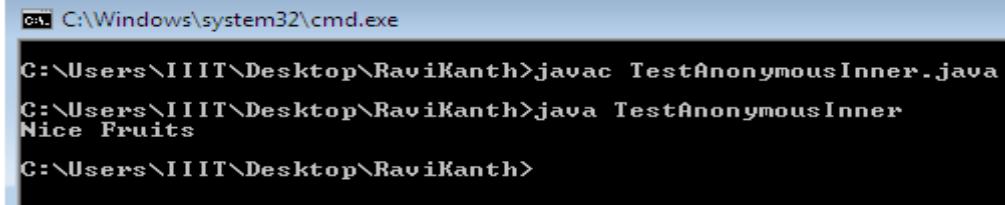
```
ch. C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac Outer_class.java
C:\Users\IIIT\Desktop\RaviKanth>java Outer_class
This is an example of AnonymousClassExam
C:\Users\IIIT\Desktop\RaviKanth>
```

## Java anonymous inner class example using interface

```
// Demo over Anonymous Class using interface

interface Eatable
{
    void eat();
}

class TestAnonymousInner
{
    public static void main(String args[])
    {
        Eatable e=new Eatable()
        {
            public void eat()
            {
                System.out.println("Nice Fruits");
            }
        };
        e.eat();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac TestAnonymousInner.java
C:\Users\IIIT\Desktop\RaviKanth>java TestAnonymousInner
Nice Fruits
C:\Users\IIIT\Desktop\RaviKanth>
```

## Java Static Nested Class

A static Inner class is a nested class which is a static member of the outer class

It can be accessed without instantiating the outer class, using other static members

Just like static members, a static nested class does not have access to the instance variables and methods of the outer class

It cannot access non-static data members and methods. It can be accessed by outer class name.

- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

Syntax:

```
class MyOuter
{
    static class Nested_Demo
    {
    }
}
```

```
// Demo on static nested class
public class StaticNestedClass
{
    private static class Nested_Class
    {
        public void display()
        {
            System.out.println("this is a static nested class");
        }
    }
    public static void main(String args[])
    {
        StaticNestedClass.Nested_Class nested=new StaticNestedClass.Nested_Class();
        nested.display();
    }
}
```



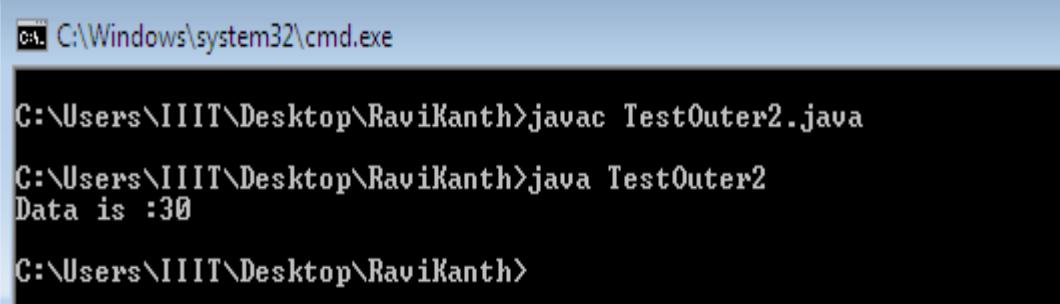
```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac StaticNestedClass.java
C:\Users\IIIT\Desktop\RaviKanth>java StaticNestedClass
this is a static nested class
C:\Users\IIIT\Desktop\RaviKanth>
```

### Java static nested class example with static method

If you have the static member inside static nested class, you don't need to create instance of static nested class.

```
// Java static nested class example with static method

class TestOuter2
{
    static int data=30;
    static class Inner
    {
        static void msg()
        {
            System.out.println("Data is :" + data);
        }
    }
    public static void main(String args[])
    {
        TestOuter2.Inner.msg();
        //no need to create the instance of static nested class
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac TestOuter2.java
C:\Users\IIIT\Desktop\RaviKanth>java TestOuter2
Data is :30
C:\Users\IIIT\Desktop\RaviKanth>
```

## UNIT-III

### Topics to be covered:

#### Finalizer methods

- Java language keeps track of all the objects we create and that we destroy when they are no longer required
- Objects are allocated memory from the heap memory and when they are not needed their memory should be reclaimed
- The Java Runtime Environment deletes objects when it determines that they are no longer required. It has its own set of algorithms for deciding when the memory has to be freed.
- This entire process of reclaiming memory is known as **Garbage Collection**

**NOTE:** An object is eligible for garbage collection when no references exists on that object. References can be either implicitly dropped when it goes out of scope or explicitly dropped by assigning null to an object reference.

#### Garbage Collector

- The java runtime environment has a garbage collector that periodically frees the memory used by objects that are no longer required or needed.
- Two basic approaches used by garbage collections are **Reference Counting and Tracing**
  - **Reference counting** maintains a reference count for every object. A newly created object will have count as 1. Throughout its lifetime, the object will be referred to by many other objects, thus incrementing the

reference count and as the referencing object moves to other objects, the reference count for that particular object is decremented. When reference count of a particular object becomes 0 zero, the object can be garbage collected.

- **Tracing technique** traces the entire set of objects (starting from root) and all objects having references on them are marked. Tracing Garbage Collector algorithm popularly known as **mark and sweep**.
- The garbage collector runs either synchronously or asynchronously in a low priority **daemon thread**. The garbage collector executes synchronously when the system runs out of memory or asynchronously when the system is idle.
- The garbage collector can be invoked to run at any time by calling **System.gc( )** or **Runtime.gc( )**.

### **Advantages:**

- Makes Java Memory Efficient
- Automatically done by JVM

### **What is the meaning of unreferenced object**

By nulling the reference

```
A a=new A();
```

```
a=null;
```

By assigning reference to another object

```
A b=new A();
```

```
a=b;
```

By anonymous Object

```
new A();
```

### Finalization

- Before an object gets garbage collected, the garbage collector gives the object an opportunity to clean up itself through a call to the object's **finalize()** method. This process is known as finalization.
- It releases all the occupied resources ( Sockets, files, etc...) can be freed
- The **finalize()** method is a member function of the predefined class **java.lang.Object**
- A class must override the **finalize()** method to perform any clean up action required by the object
- **finalize()** method is invoked each time before the object is garbage collected, which is used to perform clean-up action and defined **Object** class.
- **gc()** method is used to invoke garbage collection, available in **System** and **Runtime** class

- Syntax:

```
public void finalize()
{
}
```

### Example:

```
// Demo on Garbage Collection---finalize method & gc method
public class Garbage_Collection
{
    public void demo()
    {
        System.out.println("Iam Demo ");
    }
    public void finalize() // finalize method
    {
        System.out.println("Object is Garbage Collected");
    }
    public static void main(String args[])
    {
        Garbage_Collection gc1=new Garbage_Collection(); // Creation of Object
        gc1.demo(); // Calling demo method
        gc1=null;// sets object to null
        //gc1.demo(); // after null object this gives us Error
        System.gc(); // call Garbage Collector method
        gc1.demo(); // it results in error
    }
}
```

```
C:\Users\IIIT\Desktop\RaviKanth>javac Garbage_Collection.java
C:\Users\IIIT\Desktop\RaviKanth>java Garbage_Collection
Iam Demo
Object is Garbage Collected
Exception in thread "main" java.lang.NullPointerException
        at Garbage_Collection.main(Garbage_Collection.java:19)
```

**Note: You can even create multiple objects and check the program**

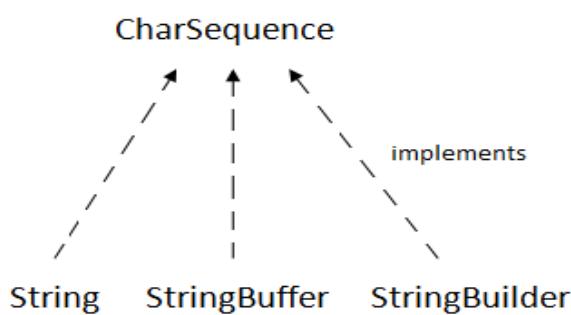
**UNIT-III****Topics to be covered:****Strings:**

Exploring String class, String Class Methods, String Buffer Class, Simple programs

**Strings in Java:****Definition:-**

Strings are sequences of Unicode characters. In many programming languages strings are stored in *arrays* of characters. However, in Java strings are a separate object type, String. The "+" operator is used for concatenation, but all other operations on strings are done with methods in the String class.

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

**Java provides 3 types of string handling classes**

**String class** -defined in “java.lang” package

- Once an object is created, no modifications can be done on that.

**StringBuffer class and StringBuilder class**— defined in “java.lang” package

- Modifications can be allowed on created object.

**StringTokenizer class** - defined in “java.util” package

- Used to divide the string into substrings based on tokens.

### **Declaration & Creation of String:**

**Syntax:** String string\_name; //Decl  
string\_name=new String(“string”); //creation

Ex: String sname;

sname=new String(“ Java”);  
(or)

String sname=new String(“java”); //Both

Java Strings can be concatenated using the ‘+’ operator

Ex: String Fullname=Name1+Name2;

If Name1=“RGUKT”, Name2=“BASAR”, we get

Fullname=“RGUKTBASAR”

### **String Arrays:**

Java also supports the creation & usage of arrays that contain strings

**Syntax:** String str\_array\_name[ ]=new String[size];

Ex: String ItemArray[ ]=new String[6];

**String Constructors:-**

1. String s= new String(); //To create an empty String call the default constructor.
2. char chars[]={‘a’,’b’,’c’,’d’};  
String s=new String(chars);
3. String(String str); //Construct a string object by passing another string object.  
String str = "abcd";  
String str2 = new String(str);
4. String( char chars[], int startIndex, int numChars)  
//To create a string by specifying positions from an array of characters  
char chars[]={‘a’,’b’,’c’,’d’,’e’,’f’};  
String s=new String(chars,2,3); //This initializes s with characters “cde”.
5. String(byte asciiChars[ ]) // Construct a string from subset of byte array.
6. String(byte asciiChars[ ], int startIndex, int numChars)

**Examples:-****// Construct one String from another.**

```
class MakeString {
    public static void main(String args[]) {
        char c[] = {'J', 'a', 'v', 'a'};
        String s1 = new String(c);
        String s2 = new String(s1);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

**Output:**

Java

Java

**// Construct string from subset of char array.**

```
class SubStringCons {
    public static void main(String args[]) {
        byte ascii[] = {65, 66, 67, 68, 69, 70 };
        String s1 = new String(ascii);
        System.out.println(s1);
        String s2 = new String(ascii, 2, 3);
        System.out.println(s2);
    }
}
```

**Output:**

ABCDEF

CDE

**String Methods:-**

| Method Call             | Return Type | Task Performed   |
|-------------------------|-------------|--|
| s2=s1.toLowerCase()     | String      | Converts the string s1 to all lowercase  |
| s2=s1.toUpperCase()     | String      | Converts the String s1 to all Uppercase  |
| s2=s1.replace('x','y')  | String      | Replaces all appearances of 'x' with 'y'   |
| s2=s1.trim()            | String      | Remove white spaces at the beginning & end of the string s1  |
| s1.equals(s2)           | Boolean     | Returns 'true' if s1 is same as s2, same characters in same order (case-sensitive)<br>'false' otherwise                            |
| s1.equalsIgnoreCase(s2) | Boolean     | " " , ignoring the case of characters  |
| s1.length()             | Int         | Returns the no. of characters in s1  |
| s1.charAt(n)            | char        | Returns the nth characters in s1   |
| s1.getChars(m,n,c,0)    | Void        | Extracts more than one character at a time & copies into a character array 'c'<br>m-source start<br>n-source end<br>0-Target start |
| s1.compareTo(s2)        | Int         | Returns zero, if s1=s2 (case sensitive)<br>-ve , if s1<s2<br>+ve , if s1>s2  |

|   |         |   |
|---|---------|---|
| s1.compareToIgnoreCase(s2)                        | Int     | ignoring case sensitive   |
| s1.regionMatches(m,s2,n,c)                        | Boolean | <p>C.compares a specific region inside a string with another specific region in another string</p> <p>True if same<br/>Else False</p> <p>m-start index of s1<br/>n-start index of s2<br/>c-no. of characters to compare</p> |
| s1.regionMatches(B,m,s2,n,c)                      | Boolean | if B is true ,ignores case  |
| s1.indexOf('ch')<br>s1.indexOf("str")             | Int     | Searches for the 1st occurrence of a character or string in s1 & returns the index value else returns '-1'  |
| s1.indexOf('ch',n)<br>s1.indexOf("str",n)         | Int     | “ ” from the nth position in s1   |
| s1.lastIndexOf('ch')<br>s1.lastIndexOf("str")     | Int     | Searches for last occurrence of a character or string in s1. The search is performed from the end of the string towards beginning & returns the index value else '-1'   |
| S1.lastIndexOf('ch',n)<br>S1.lastIndexOf("str",n) | Int     | “ ” from the nth position from end to begin   |

|                      |          |   |
|----------------------|----------|---|
| S1.substring(n)      | String   | Gives substring starting from nth character                                 |
| S1.substring(n,m)    | String   | “ ” upto mth character ( not including mth character )                      |
| S1.startsWith("str") | Boolean  | Determines whether a given string 's1' begins with a specific string or not |
| S1.endsWith("str")   | Boolean  | “ ” “ ” ends  |
| S1.concat(s2)        | String   | Concatenates s1&s2 stored   |
| S1.toCharArray()     | Char [ ] | Returns a character array containing a copy of the characters in a string   |
| o.toString()         | String   | Converts all objects into string objects                                    |
| String.valueOf(o)    | String   | Creates a string object of the parameter 'o' (Simple type or Object type)   |
| String.valueOf(var)  | String   | Converts the parameter value to String representation                       |

**Character Extraction:-**

```
class getCharsDemo
{
    public static void main(String args[])
    {
        String s = "This is a demo of the getChars method.";
        int start = 10;
        int end = 14;
        char buf[] = new char[end - start];
        s.getChars(start, end, buf, 0);
        System.out.println (buf);
    }
}
```

**Output:**

demo

```

class GetBytesDemo
{
    public static void main(String[] args)
    {
        String str = "abc" + "ABC";
        byte[] b = str.getBytes();
        //char[] c=str.toCharArray();
        System.out.println(str);
        for(int i=0;i<b.length;i++)
        {
            System.out.print(b[i]+" ");
            //System.out.print(c[i]+" ");
        }
    }
}

```

Output:

|           |           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|-----------|
| <b>97</b> | <b>98</b> | <b>99</b> | <b>65</b> | <b>66</b> | <b>67</b> |
| //a       | b         | c         | A         | B         | C         |

### **String Comparison:**

1. boolean **equals(Object str)**

To compare two strings for equality. It returns true if the strings contain the same characters in the same order, and false otherwise.

2. boolean **equalsIgnoreCase(String str)**

To perform a comparison that ignores case differences.

```
// Demonstrate equals() and equalsIgnoreCase().  
class equalsDemo  
{  
    public static void main(String args[])  
    {  
        String s1 = "Hello";  
        String s2 = "Hello";  
        String s3 = "Good-bye";  
        String s4 = "HELLO";  
        System.out.println(s1.equals(s2));  
        System.out.println(s1.equals(s3));  
        System.out.println(s1.equals(s4));  
        System.out.println(s1.equalsIgnoreCase(s4));  
    }  
}
```

Output:

```
true  
false  
false  
true
```

**String Comparison using regionMatches Method:-**

```
boolean regionMatches(int startIndex, String str2, int str2startIndex, int numChars)
```

```
boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2startIndex, int numChars)
```

- The regionMatches( ) method compares a specific region inside a string with another specific region in another string.
- startIndex specifies the index at which the region begins within the invoking String object.
- The String being compared is specified by str2. The index at which the comparison will start within str2 is specified by str2startIndex.
- The length of the substring being compared is passed in numChars.

**Example:-**

```
class RegionTest
{
    public static void main(String args[])
    {
        String str1 = "This is Test";
        String str2 = "THIS IS TEST";
        if(str1.regionMatches(5,str2,5,3))
        {
            // Case, pos1,secdString,pos1,len
            System.out.println("Strings are Equal");
        }
        else
        {
            System.out.println("Strings are NOT Equal");
        }
    }
}
```

**Output:** Strings are NOT Equal

//Sorting of String using compareTo method.

```
class SortString
{
    static String arr[] = { "is", "the", "time", "for", "all", "good", "men", "to",
                           "come", "to", "the", "aid", "of", "their", "country" };
    public static void main(String args[])
    {
        for(int j = 0; j < arr.length; j++)
        {
            for(int i = j + 1; i < arr.length; i++)
            {
                if(arr[i].compareTo(arr[j]) < 0)
                {
                    String t = arr[j];
                    arr[j] = arr[i];
                    arr[i] = t;
                }
            }
            System.out.println(arr[j]);
        }
    }
}
```

Output:-

```
aid all come country for good is men of
the the their time to to
```

**Changing the Case of Characters Within a String :-**

String toLowerCase( ):- converts all the characters in a string from uppercase to lowercase.

String toUpperCase( ):- converts all the characters in a string from lowercase to uppercase

// Demonstrate toUpperCase() and toLowerCase().

```
class ChangeCase
{
    public static void main(String args[])
    {
        String s = "This is a test.";
        System.out.println("Original: " + s);
        String upper = s.toUpperCase();
        String lower = s.toLowerCase();
        System.out.println("Uppercase: " + upper);
        System.out.println("Lowercase: " + lower);
    }
}
```

**Output:**

Original: This is a test.

Uppercase: THIS IS A TEST.

Lowercase: this is a test.

**StringBuffer Class:**

- StringBuffer is mutable means one can change the value of the object .
- The object created through StringBuffer is stored in the heap. StringBuffer has the same methods as the StringBuilder , but each method in StringBuffer is synchronized that is StringBuffer is thread safe . Due to this it does not allow two threads to simultaneously access the same method. Each method can be accessed by one thread at a time .
- StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

**StringBuffer Constructors**

1. StringBuffer( ): It reserves room for 16 characters without reallocation.  
`StringBuffer s=new StringBuffer();`
2. StringBuffer( int size)It accepts an integer argument that explicitly sets the size of the buffer.  
`StringBuffer s=new StringBuffer(20);`
3. StringBuffer(String str): It accepts a String argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.  
`StringBuffer s=new StringBuffer("RGUKTIIIT");`

**Methods**

1. StringBuffer append() method

The append() method concatenates the given argument with this string.

```

class StringBufferExample
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("Hello ");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
    }
}

```

## 2. StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

```

class StringBufferExample2
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("Hello ");
        sb.insert(1,"Java");//now original string is changed
        System.out.println(sb);//prints HJavaello
    }
}

```

## 3. StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBufferExample3
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("Hello");
        sb.replace(1,3,"Java");
        System.out.println(sb);//prints HJava
    }
}
```

#### 4. StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class StringBufferExample4
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("Hello");
        sb.delete(1,3);
        System.out.println(sb);//prints Hlo
    }
}
```

## 5. StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
class StringBufferExample5
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("Hello");
        sb.reverse();
        System.out.println(sb);//prints olleH
    }
}
```

## 6. StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by  $(\text{oldcapacity} * 2) + 2$ . For example if your current capacity is 16, it will be  $(16 * 2) + 2 = 34$ .

```
class StringBufferExample6
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer();
        System.out.println(sb.capacity());//default 16
        sb.append("Hello");
        System.out.println(sb.capacity());//now 16
    }
}
```

```

        sb.append("java is my favourite language");

        System.out.println(sb.capacity());//now (16*2)+2=34 i.e (
            oldcapacity*2)+2

    }

}

```

#### 7. StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

```

class StringBufferExample7
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer();

        System.out.println(sb.capacity());//default 16

        sb.append("Hello");

        System.out.println(sb.capacity());//now 16

        sb.append("java is my favourite language");

        System.out.println(sb.capacity());//now (16*2)+2=34 i.e (
            oldcapacity*2)+2

        sb.ensureCapacity(10);//now no change

        System.out.println(sb.capacity());//now 34

        sb.ensureCapacity(50);//now (34*2)+2

        System.out.println(sb.capacity());//now 70
    }
}

```

### **StringBuilder**

- StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified .
- The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe. StringBuilder is fast as it is not thread safe.

### **Constructors:**

- `StringBuilder demo2= new StringBuilder("Hello");`  
The above object too is stored in the heap and its value can be modified
- `demo2=new StringBuilder("Bye");`  
Above statement is right as it modifies the value which is allowed in the StringBuilder

### **String Tokenizer class:-**

- The string tokenizer class allows an application to break a string into tokens.
- The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.
- StringTokenizer class is used to split a String into different tokens as by defined delimiter.(space is the default delimiter).

### **Constructors:-**

1. `StringTokenizer(String str) :`  
str is string to be tokenized. Considers default delimiters like new line, space, tab, carriage return and form feed.
2. `StringTokenizer(String str, String delim) :`  
delim is set of delimiters that are used to tokenize the given string.

### 3. StringTokenizer(String str, String delim, boolean flag):

The first two parameters have same meaning.

The flag serves following purpose.

If the flag is false, delimiter characters serve to separate tokens. For example, if string is "hello RGUKT" and delimiter is " ", then tokens are "hello" and "RGUKT".

If the flag is true, delimiter characters are considered to be tokens. For example, if string is "hello RGUKT" and delimiter is " ", then tokens are "hello", " " and "RGUKT".

#### Methods

- int **countTokens()** //returns number of tokens in the string.
- boolean **hasMoreTokens()** //checks whether tokens are there or not
- String **nextToken()** //returns the token in the string

#### Example:-

```
import java.util.StringTokenizer;
public class StringTokenizer_Test
{
    static String str = "Hello,Welcome,to,Java,Programming";
    public static void main(String args[])
    {
        StringTokenizer st = new StringTokenizer(str);
        StringTokenizer st1 = new StringTokenizer(str,",");
        StringTokenizer st2 = new StringTokenizer(str,",",true);
        while(st.hasMoreTokens())
```

```
{  
    String tokens = st.nextToken();  
    System.out.print(tokens + "\n");  
}  
while(st1.hasMoreTokens())  
{  
    String tokens = st1.nextToken();  
    System.out.print(tokens + "\n");  
}  
while(st2.hasMoreTokens())  
{  
    String tokens = st2.nextToken();  
    System.out.print(tokens + "\n");  
}  
}  
}
```

**Output:**

Hello,Welcome,to,Java,Programming

Hello

Welcome

to

Java

Programming

Hello

,

Welcome

,

to  
,  
Java  
,  
Programming

**Exercise1:**

Write a java program that reads a line of integers and then displays each integer and find the sum of the integers (using StringTokenizer)?

```
import java.util.Scanner;
import java.util.*;
class Token
{
    static int sum=0;
    public static void main(String sree[])
    {
        Scanner s=new Scanner(System.in);
        System.out.print("Enter sum of integers: ");
        String str=s.next();
        StringTokenizer st=new StringTokenizer(str,"+");
        while(st.hasMoreTokens())
        {
            sum=sum+Integer.parseInt(st.nextToken());
        }
        System.out.println("Sum of "+str+"is: "+sum);
    }
}
```

**Input:**

Enter sum of integers: 10+20+30+40

**Output:**

Sum of 10+20+30+40 is: 100

**String****1. Immutable Type [ Concat method]**

`String s1=new String("RGUKT"); // s1 is going to point to RGUKT`

`s1.concat("IIIT-Basar"); RGUKTIIIT-BASAR`

`S.OP(s1); RGUKT`

**2. Slow[ SPA]****3. Consumes more memory when u concat too many strings****4. Overrides equals() method****String Buffer****1. Mutable Type [ append method]**

`StringBuffer sb1=new StringBuffer("RGUKT");`

`Sb1.append("IIIT-Basar");`

`S.op(sb1); // RGUKTIIIT-Basar`

**2. Fast****3. Consumes less memory****4. Doesnot override equals() method**

**Note:**

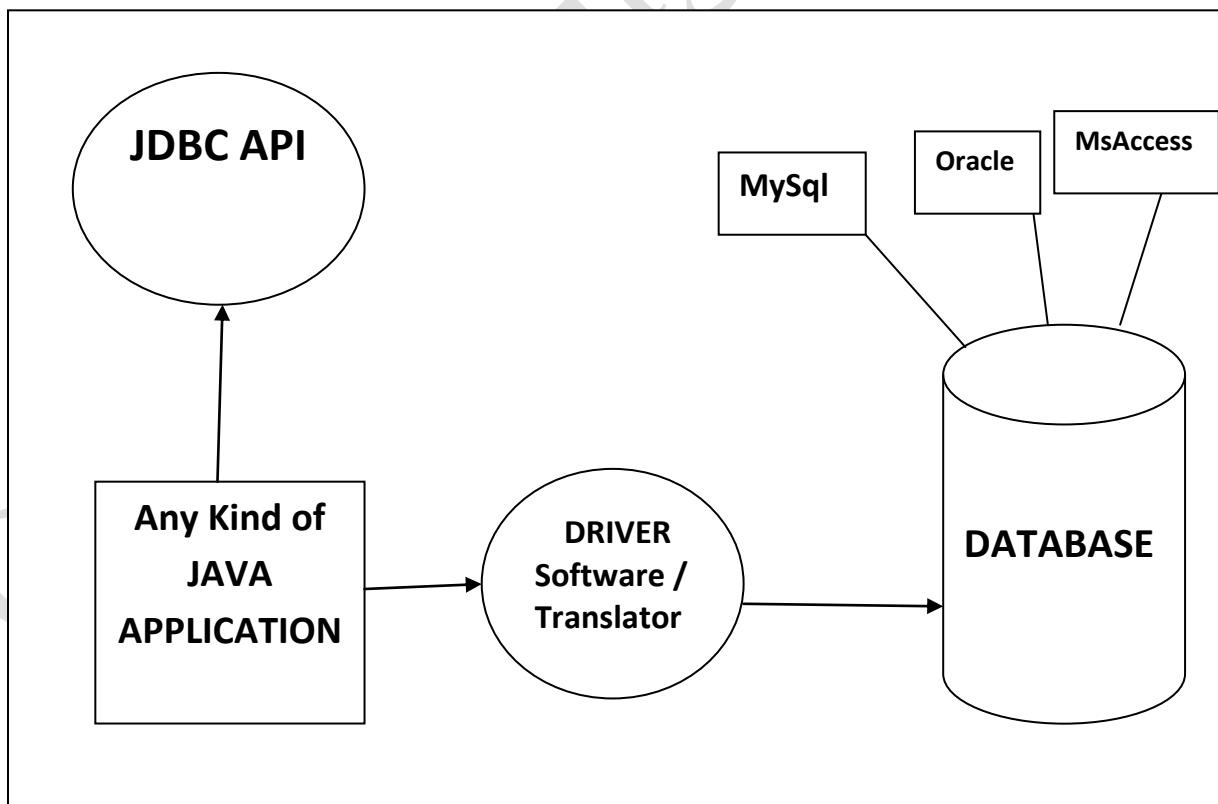
- 1. Once we create a string object we cannot perform any changes in the existing object.**
  - 2. If we are trying to perform any changes with those changes a new object will be created. This non changeable nature is nothing but immutability of string in Java**
  - 3. Once we create a StringBuffer obj. we can perform any type of changes in the existing object. This change is nothing but mutability of SB object**
- 
-

**SYLLABUS:****UNIT-5:****JDBC, ODBC Drivers:**

JDBC ODBC Bridges, Seven Steps to JDBC, Importing java SQL Packages, Loading & Registering the drivers, Establishing connection. Creating & Executing the statement.

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver



We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.

The **java.sql package** contains classes and interfaces for JDBC API.

A list of popular interfaces of **JDBC API** are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

### Drawback in JDBC-ODBC Driver

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

### **JDBC-ODBC bridge driver**

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

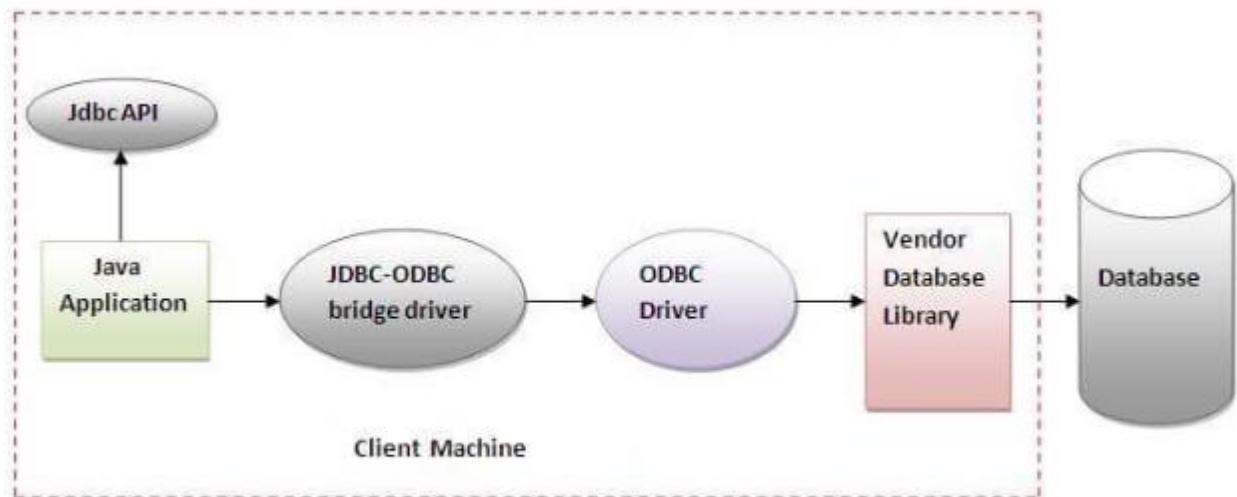


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- o easy to use.
- o can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

## Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

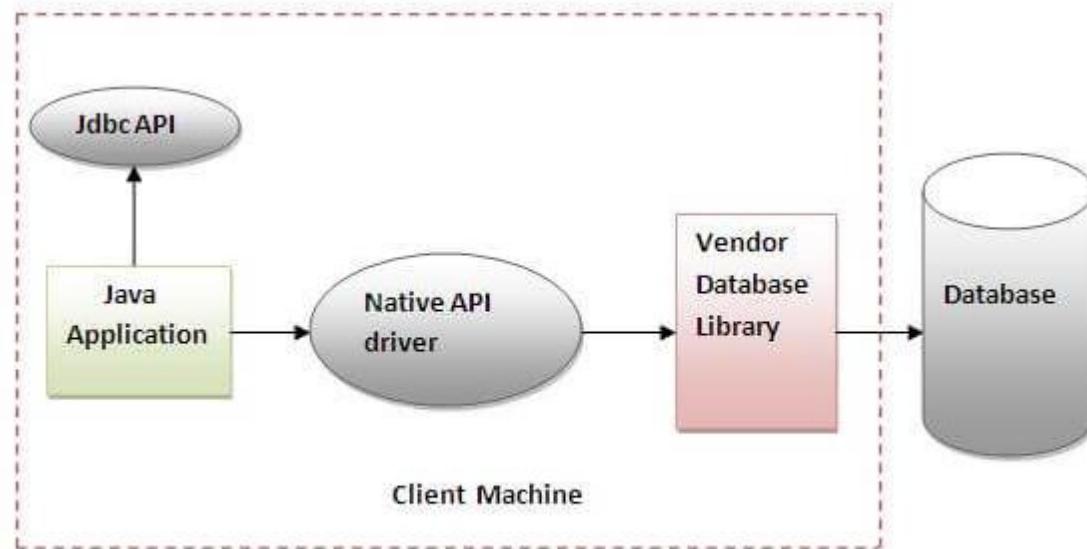


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

### Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

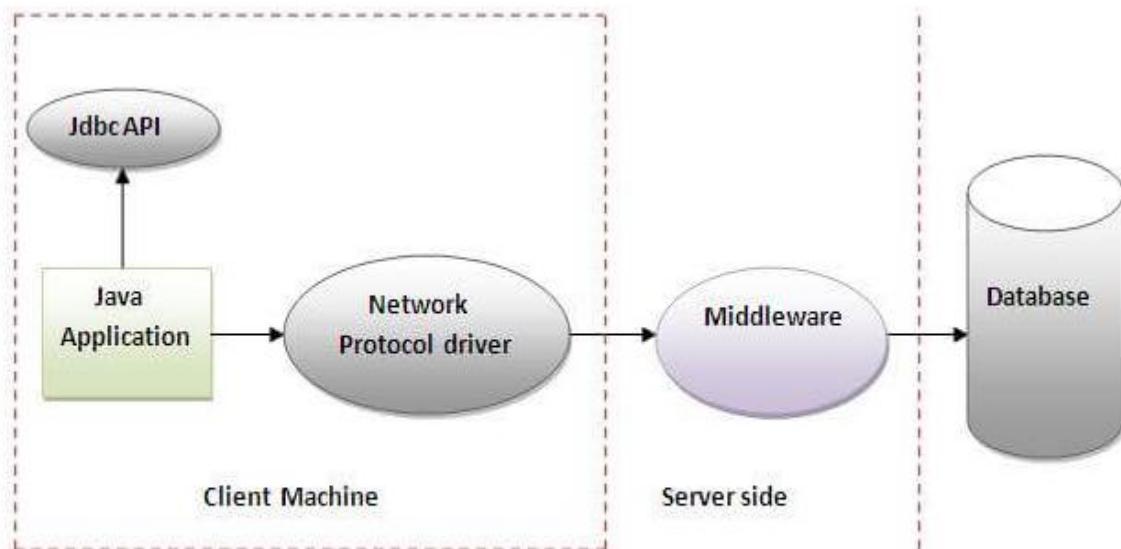


Figure- Network Protocol Driver

#### Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

#### Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

### Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

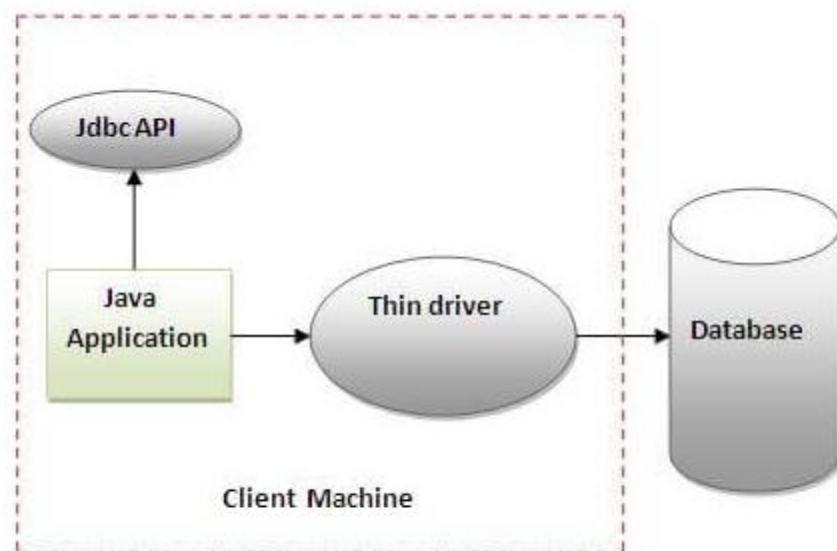


Figure- Thin Driver

#### Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

#### Disadvantage:

- Drivers depend on the Database.

- **What is MySQLdb?**

- MySQLdb is an interface for connecting to a MySQL database server to Java. It implements the Java Database API v2.0 and is built on top of the MySQL C API.

- **How do I Install MySQLdb?**

- Before proceeding, you make sure you have MySQLdb installed on your machine.
- STEP-1: Firstly choose application mysql-essential-5.0.41-win32.msi
- STEP-2: Run the file
- STEP-3: Next
- STEP-4: Choose complete and click on next
- STEP-5: Click Install
- STEP-6: Click Next and Next
- STEP-7: Close the file

And click on windows button

Type mysql choose second one

Open (Mysql server instance config wizard)

- STEP-8: Cilck next
  - STEP-9: Click next
- Upto Modify security settings and set password for your mysql database and click next
- STEP-10: Click on execute
  - #If does not exexute all configuration go to (STEP-7)
  - Then another file name Is mysql connectivity
  - Click to install

Then click next

Then next

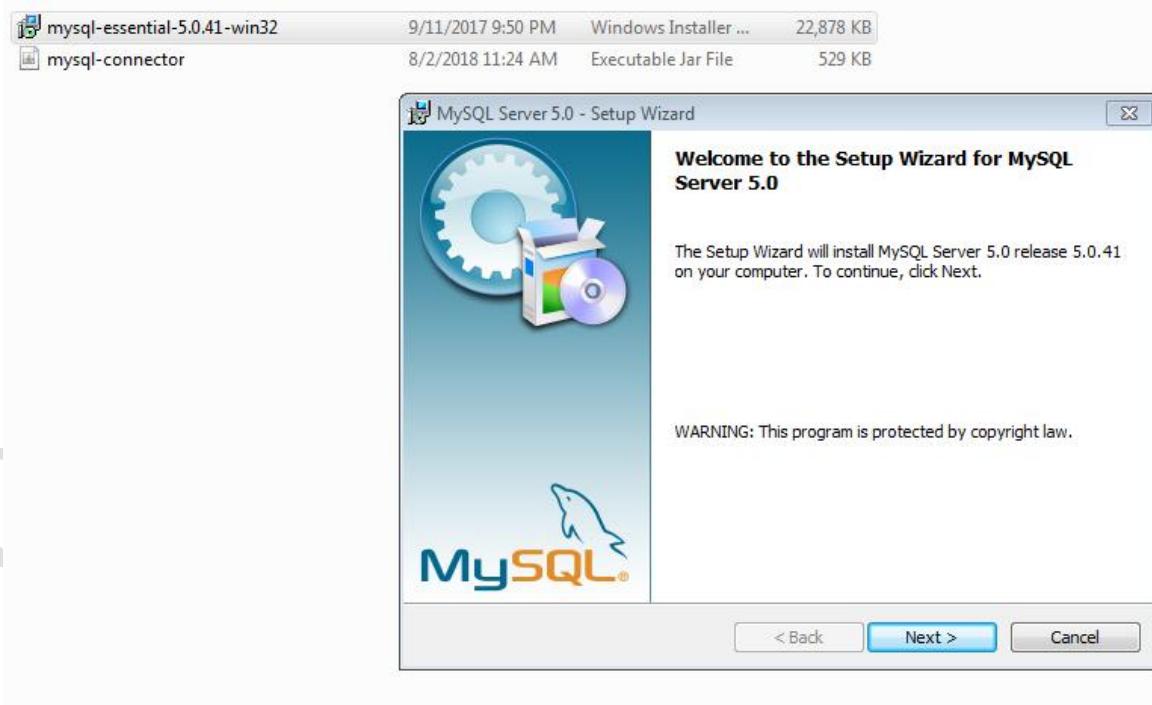
Then install

Ok enjoy with mysql server

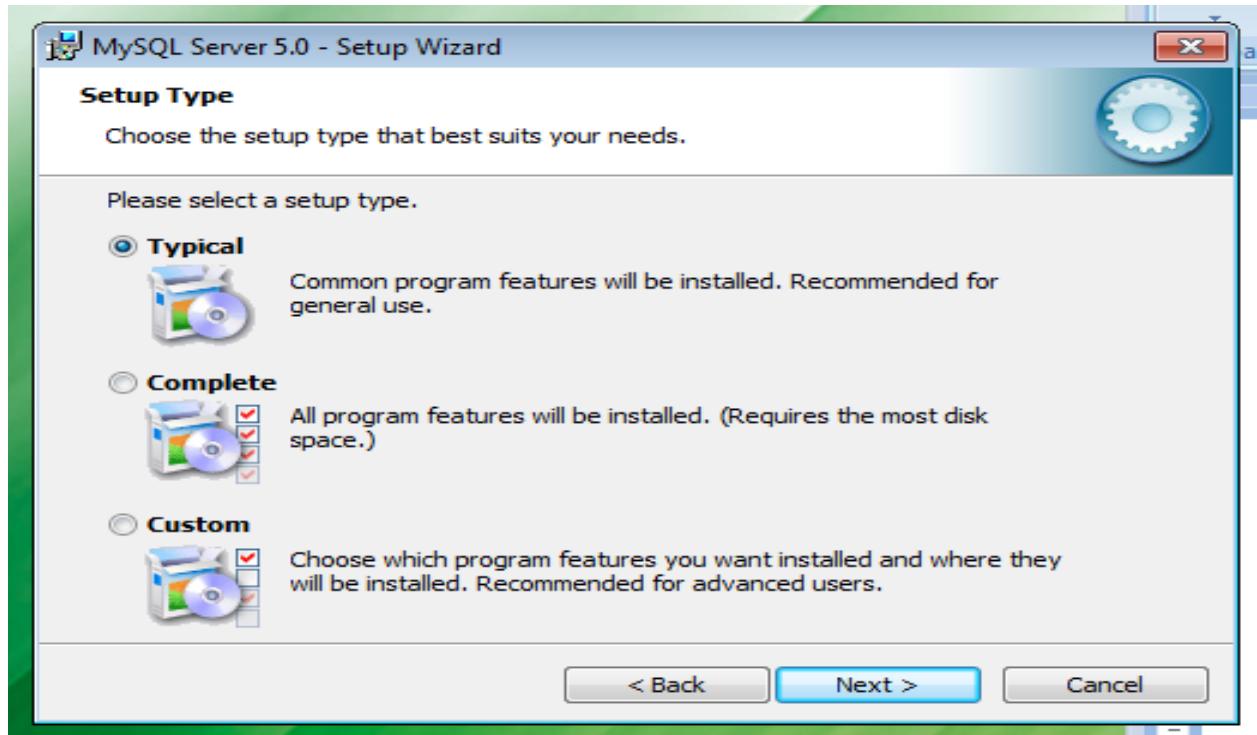
- **Procedure to work with Java Connection with MySQL Database**

- Check whether Database is ready on to your system[ Install MySQL]
- Do remember the Username & Password and Database name
- Create Database by User Defined Name
- Create Table & Insert the values and Update & Delete the values
- Check the Data by Select command
- Hope now you are ready with your Database

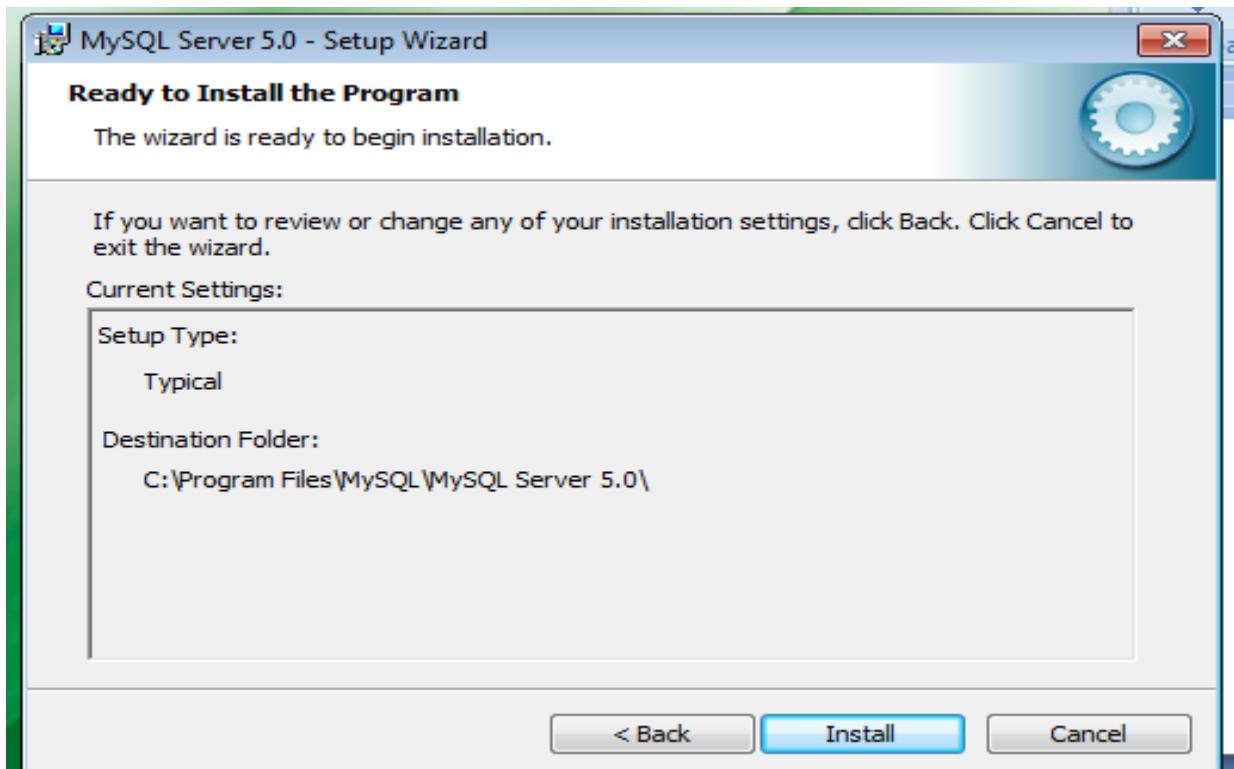
- **Install mysql DB software**



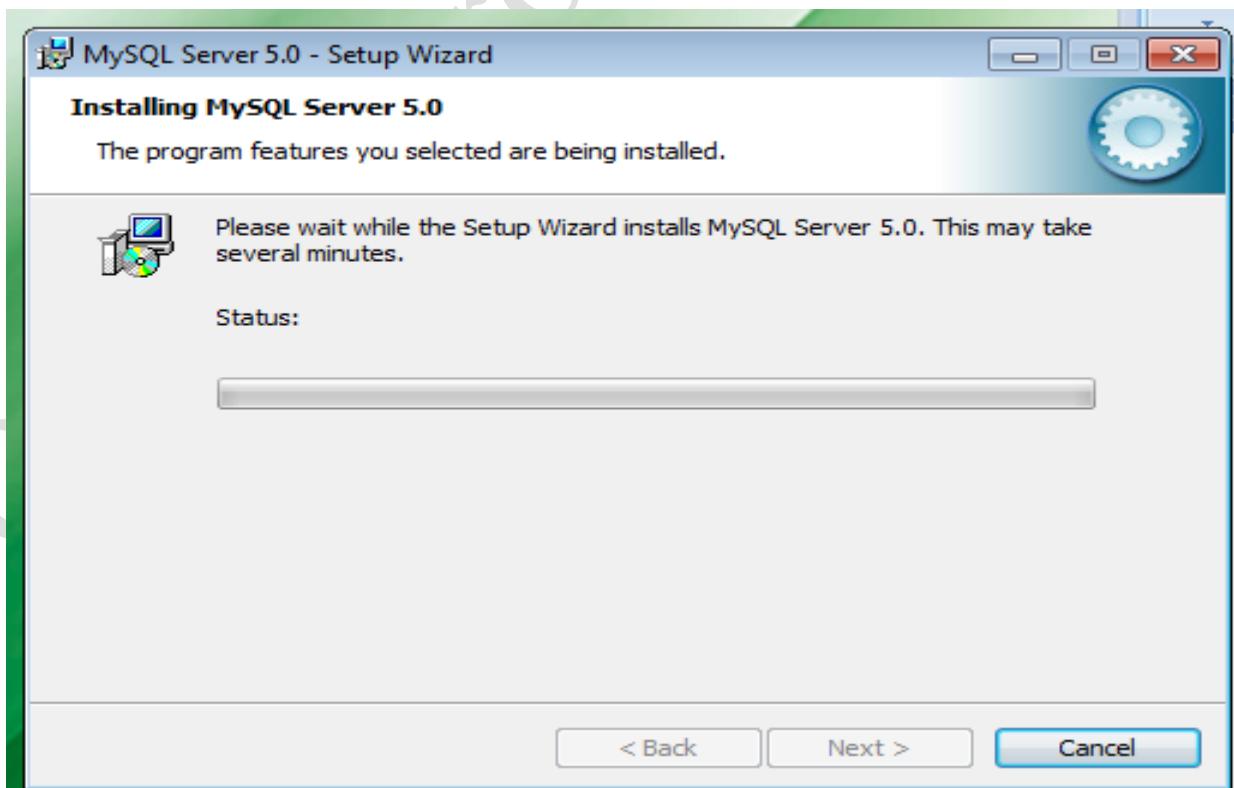
- Click on Next Button



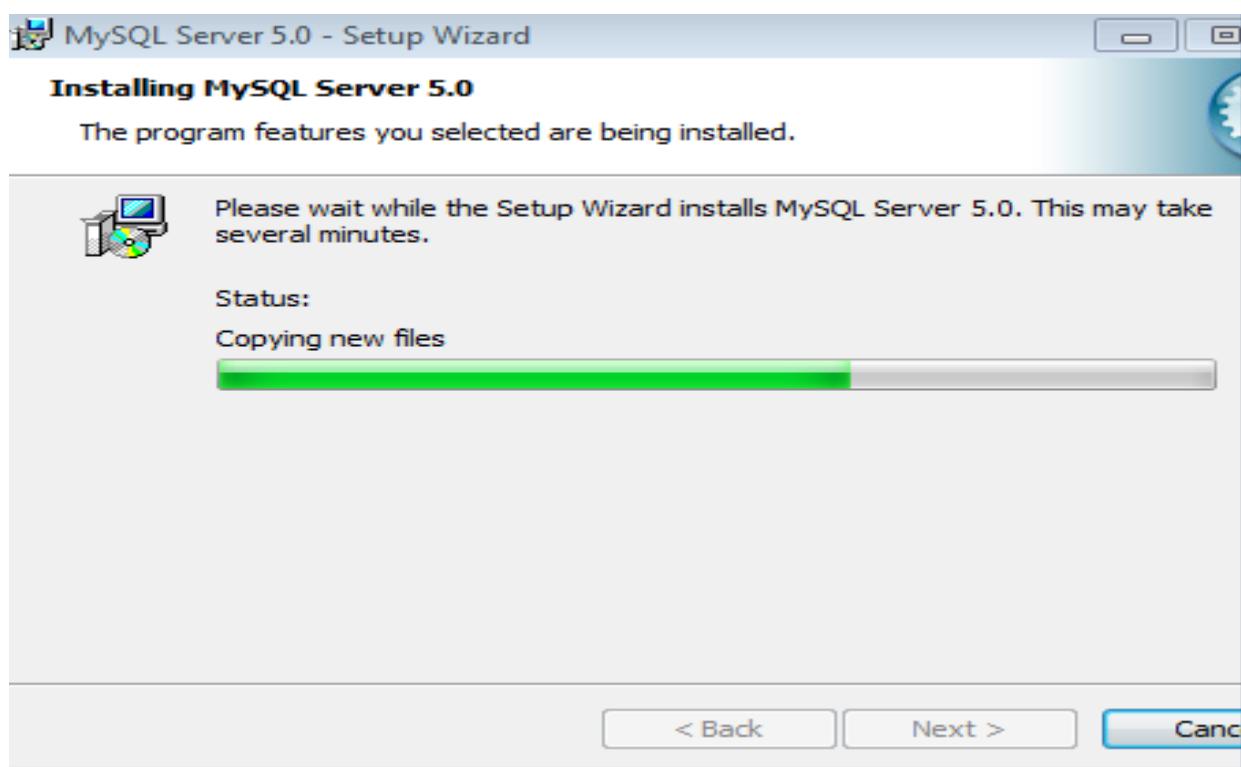
- Click on Next Button

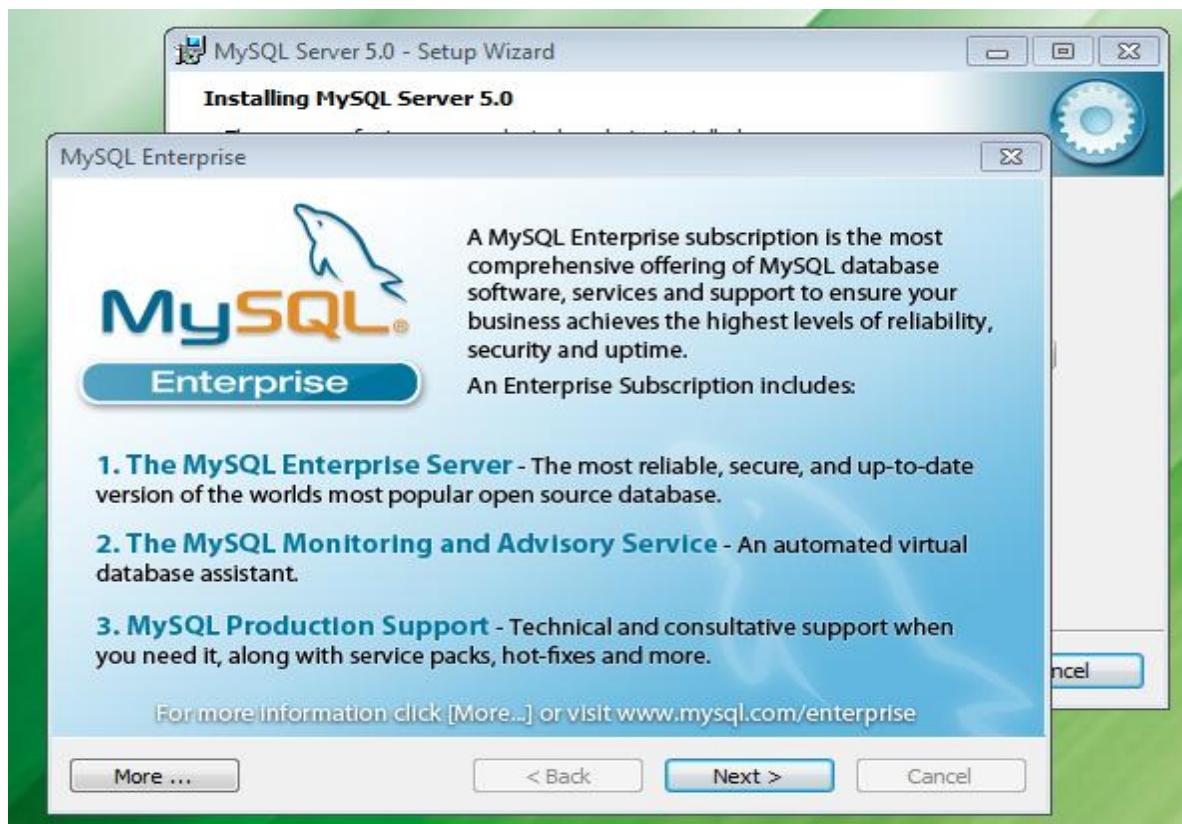


- Click on Install Button

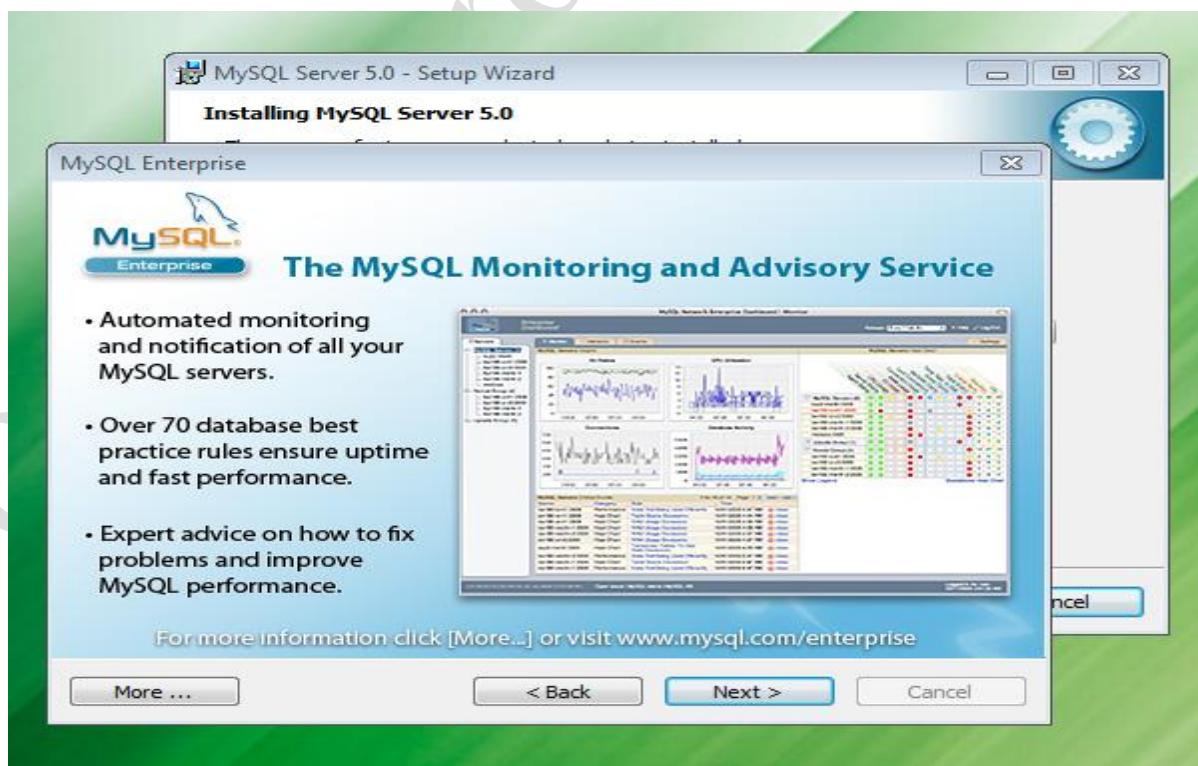


- Wait for the process to run.. Have Patience





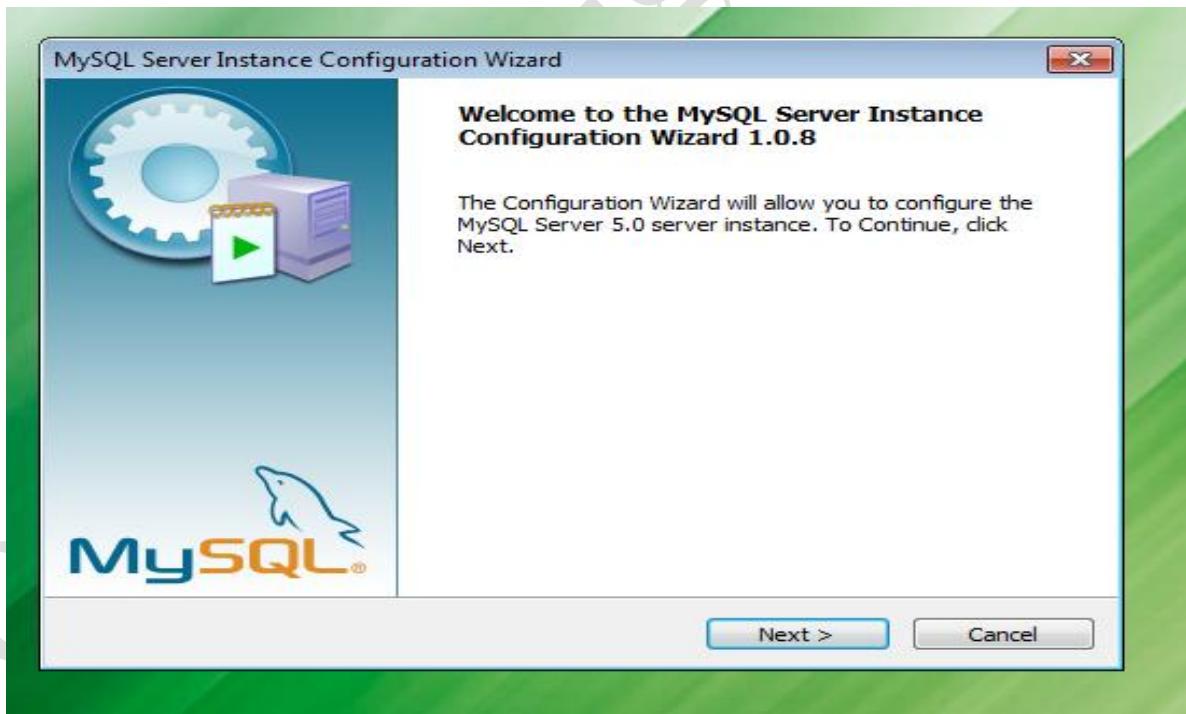
- Again click on next button



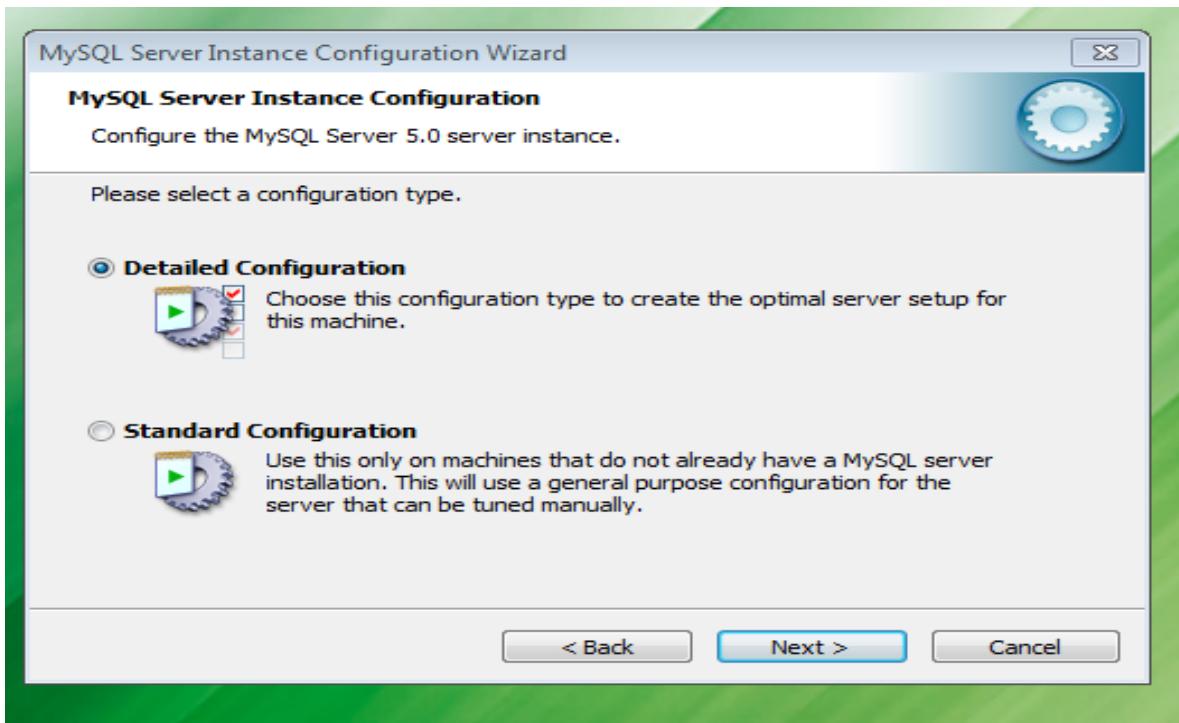
- Click on next button



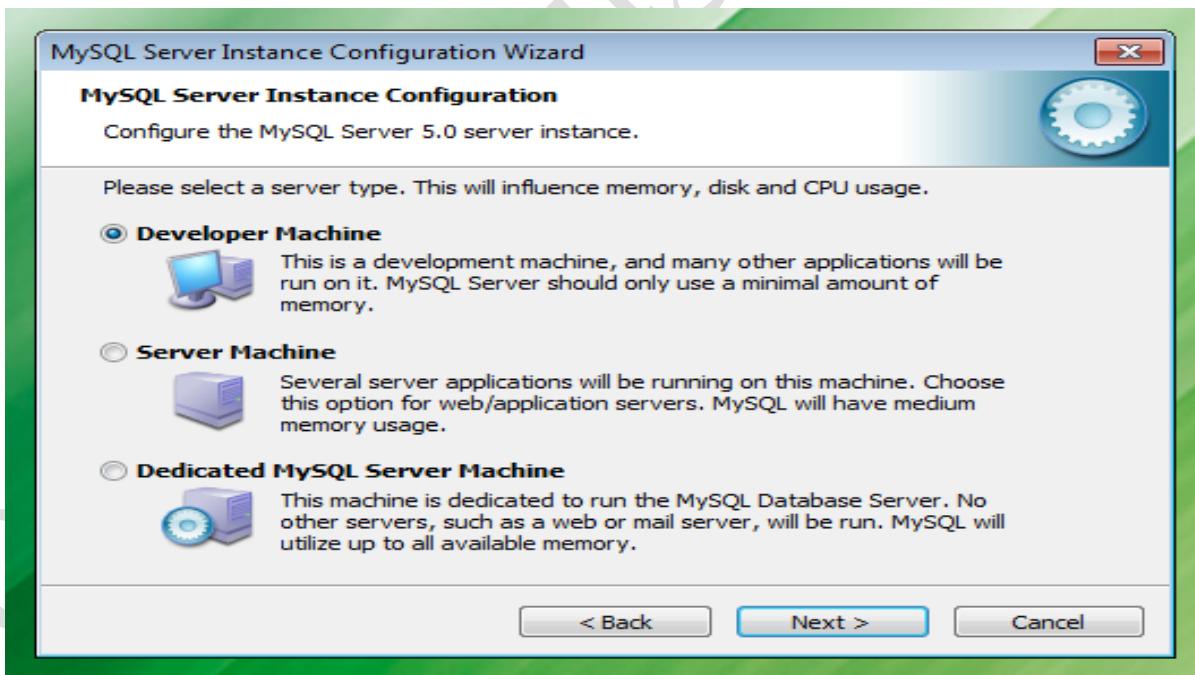
- Click on finish button



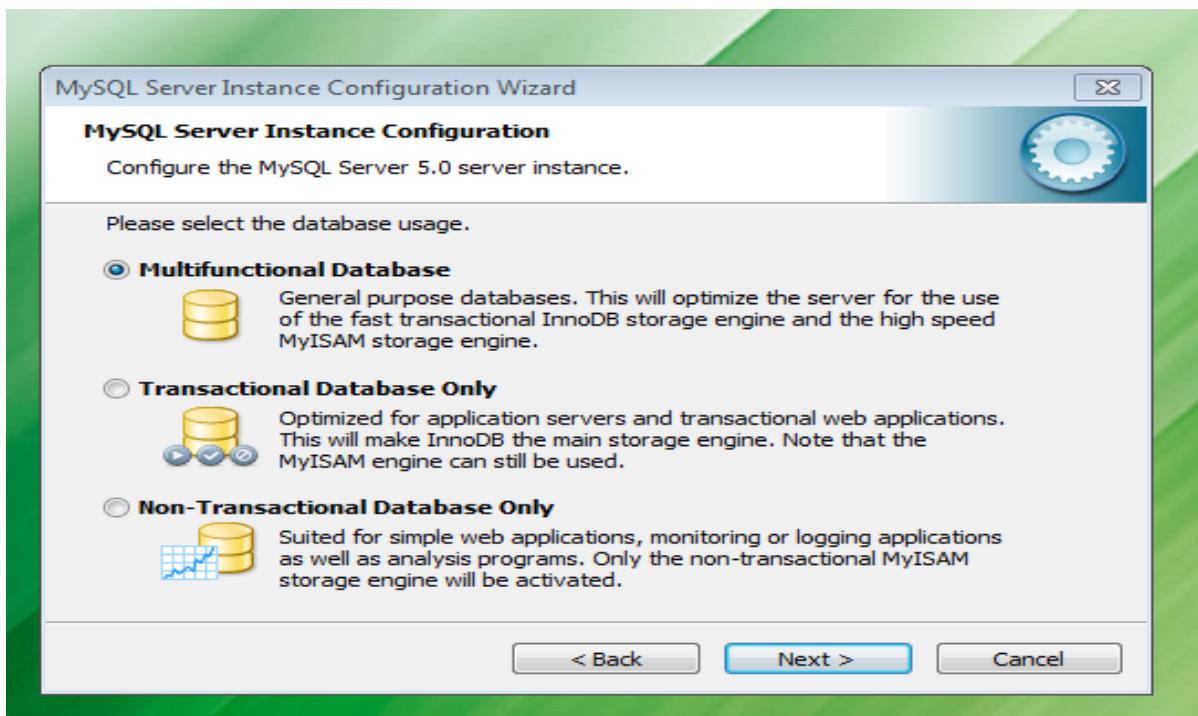
- Now it is checking for configuration setting ...Click on next



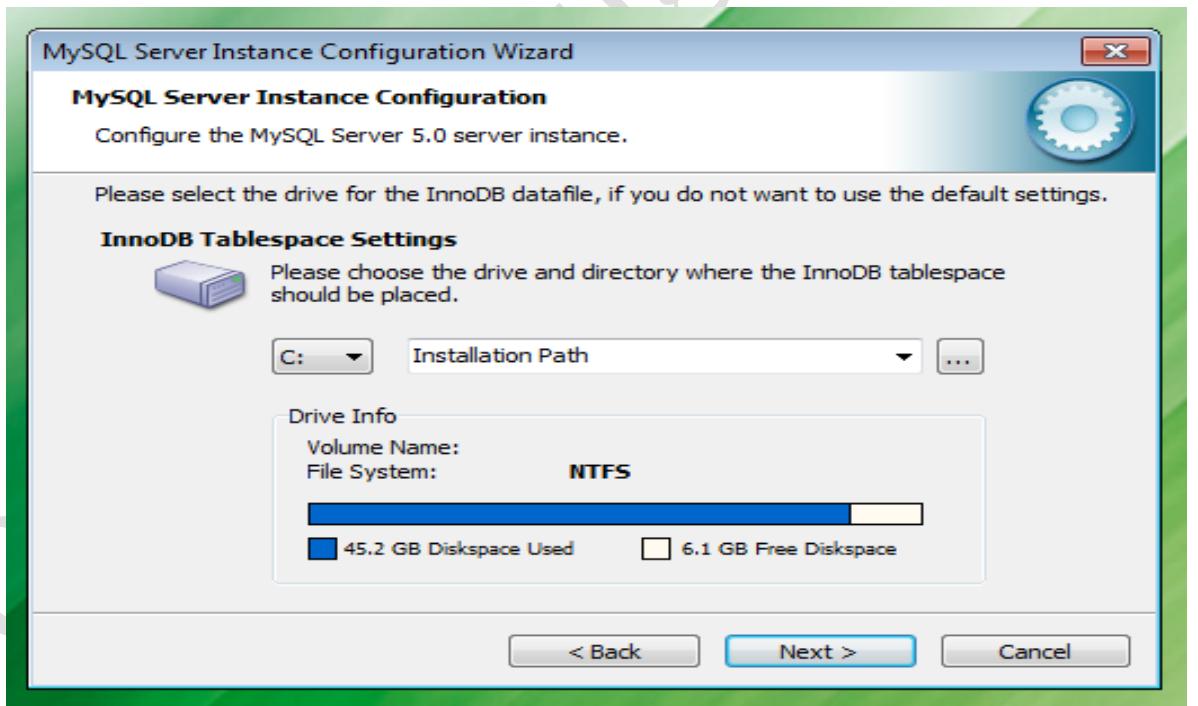
- Click on next



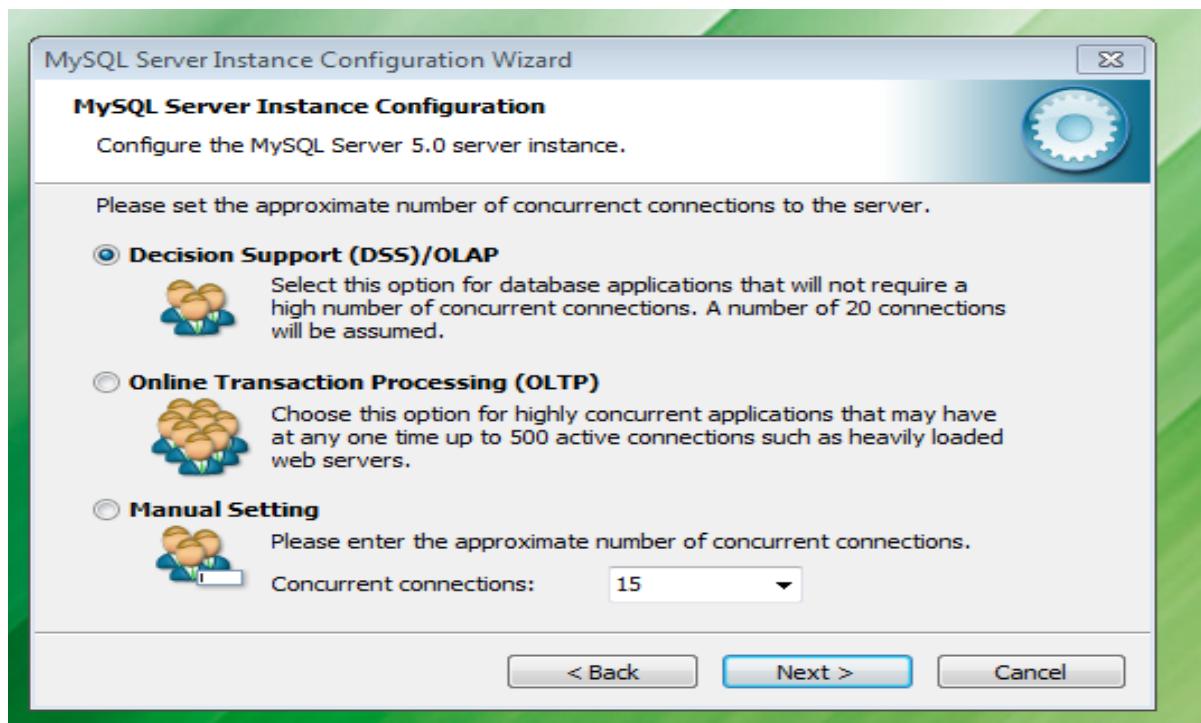
- Again click on next



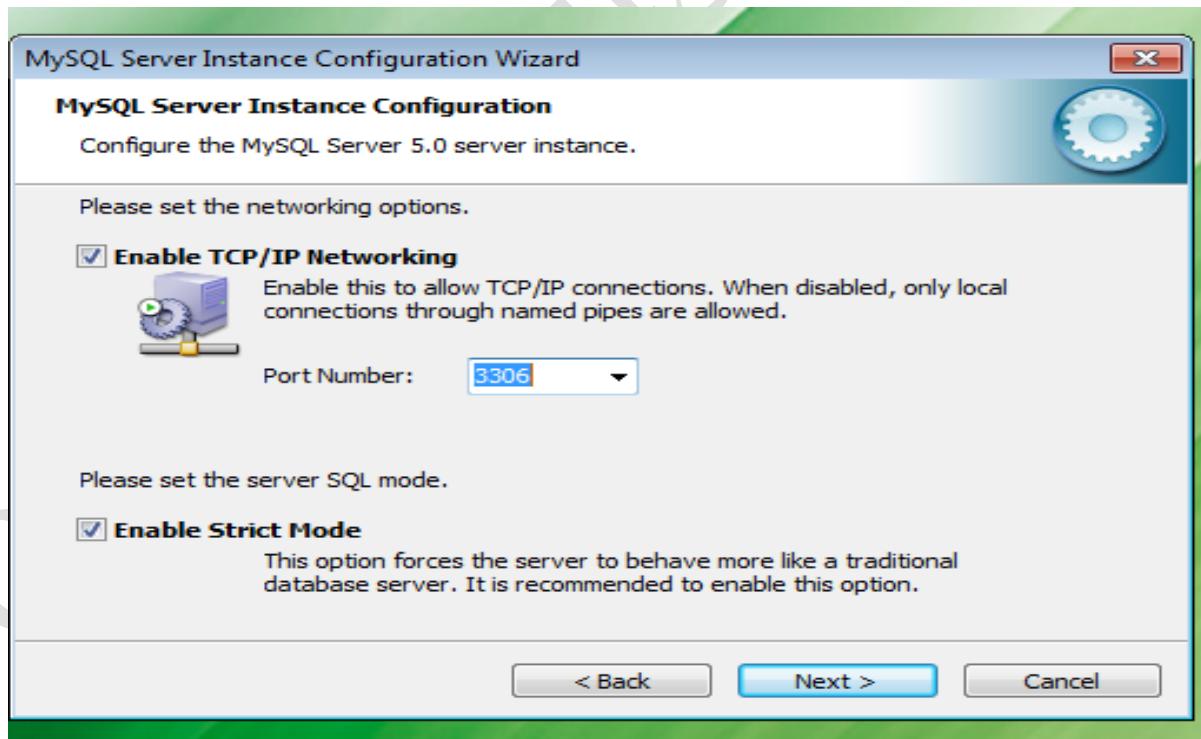
- Again click on next



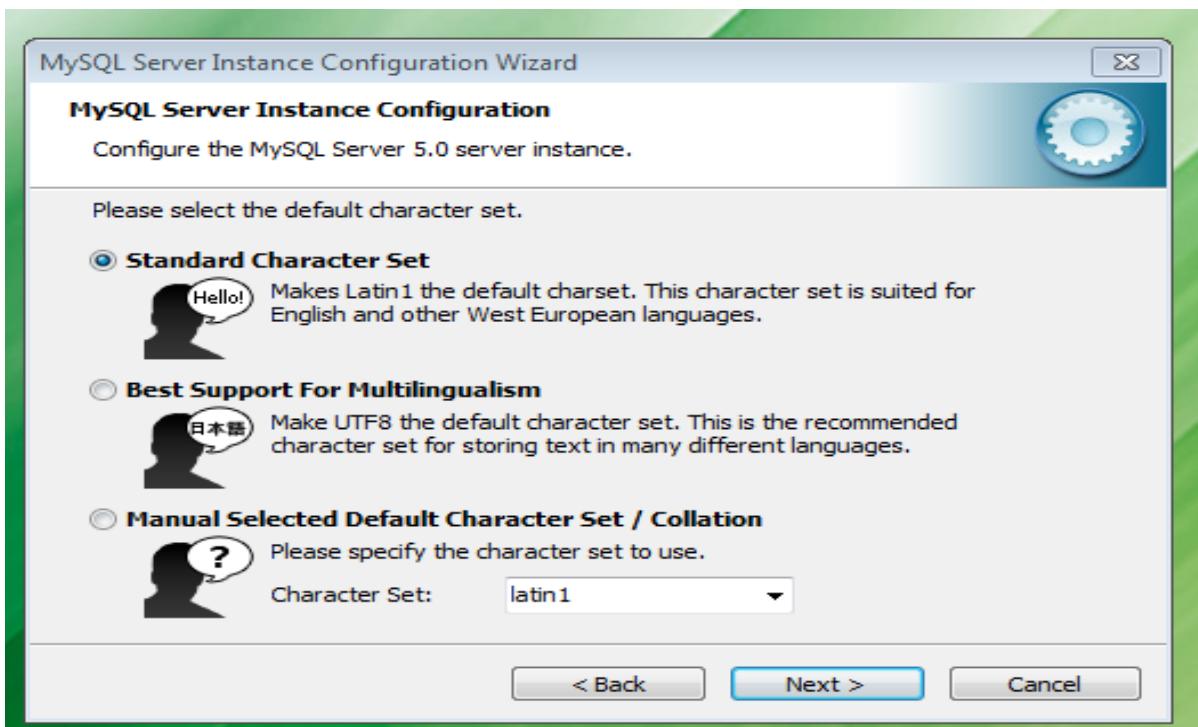
- Checking for installation path ...click next



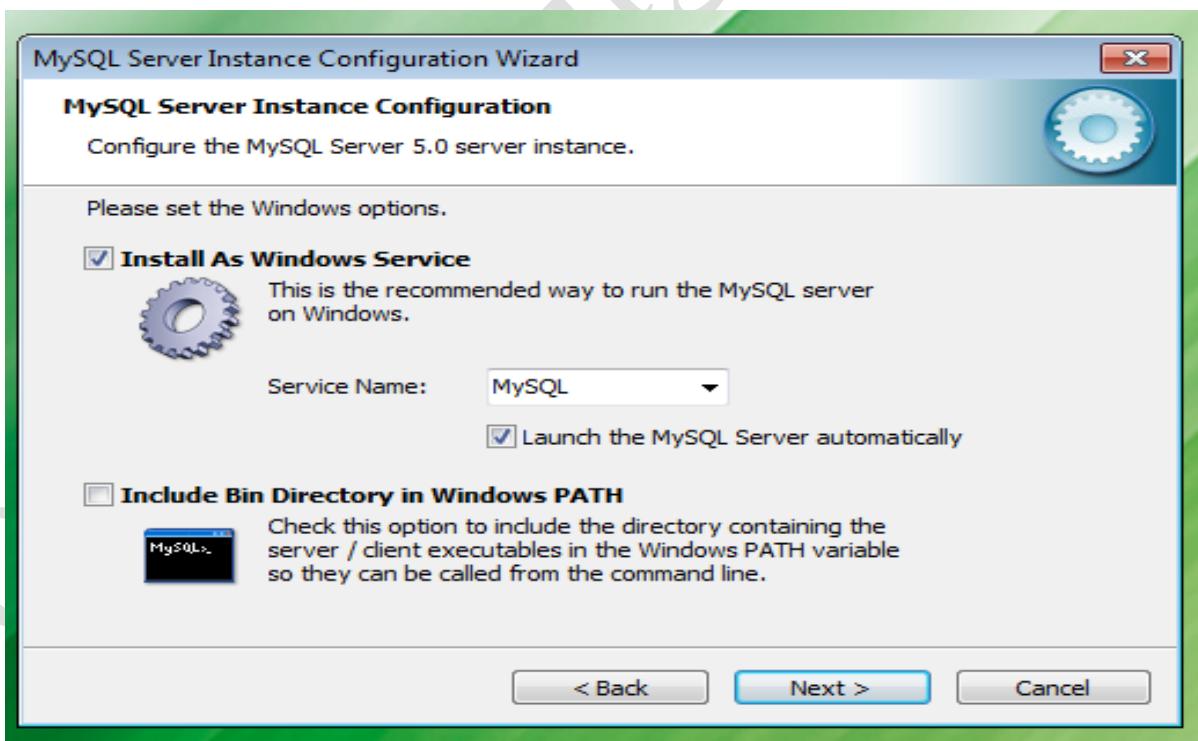
- Click next



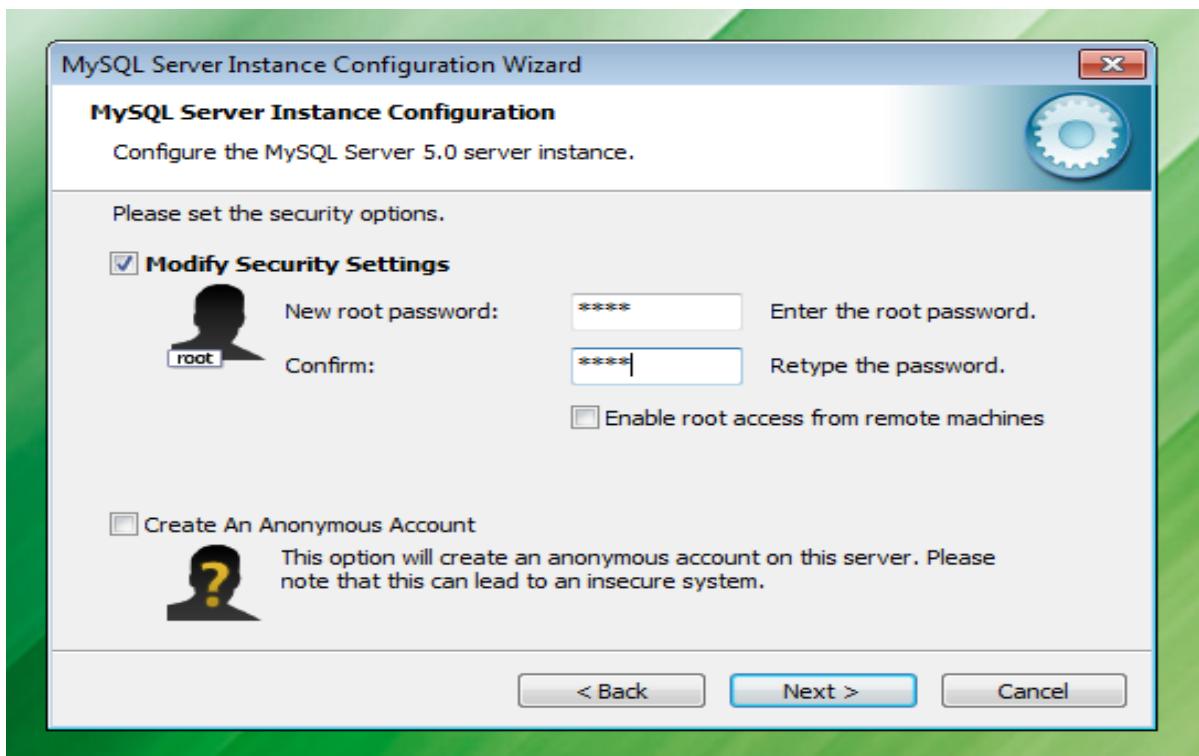
- Setting the Port number by default... click next



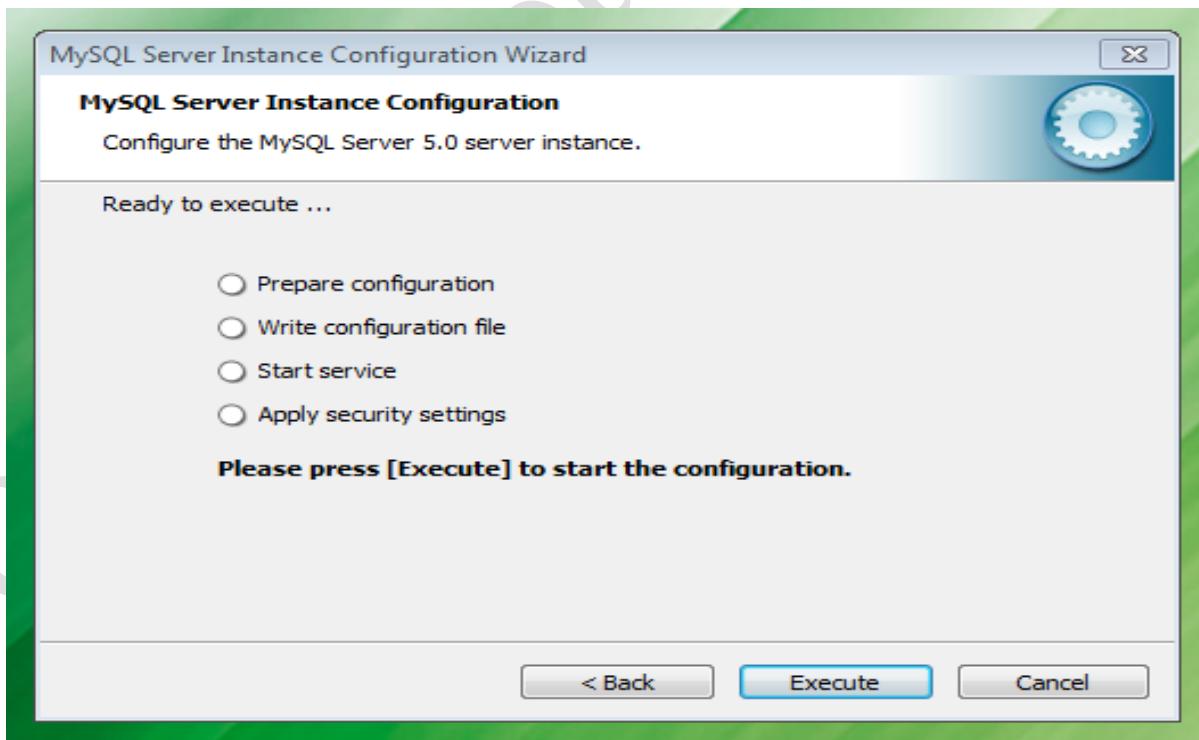
- Click next



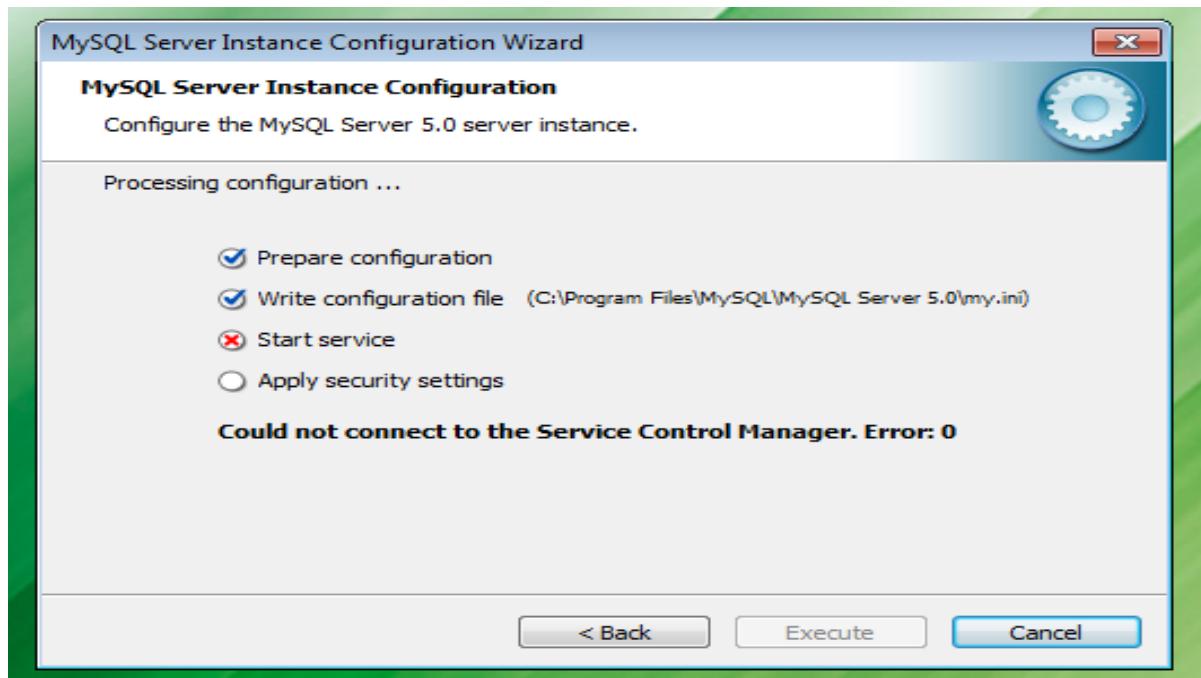
- Click next



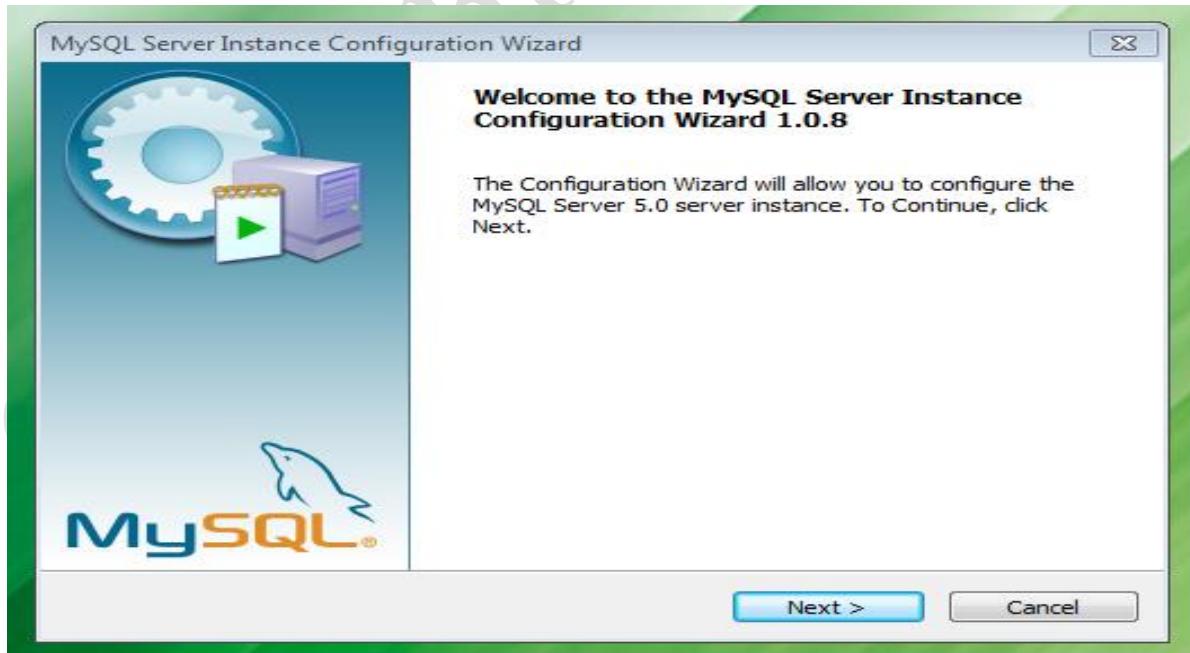
- Give the Password and do remember the password and click on next



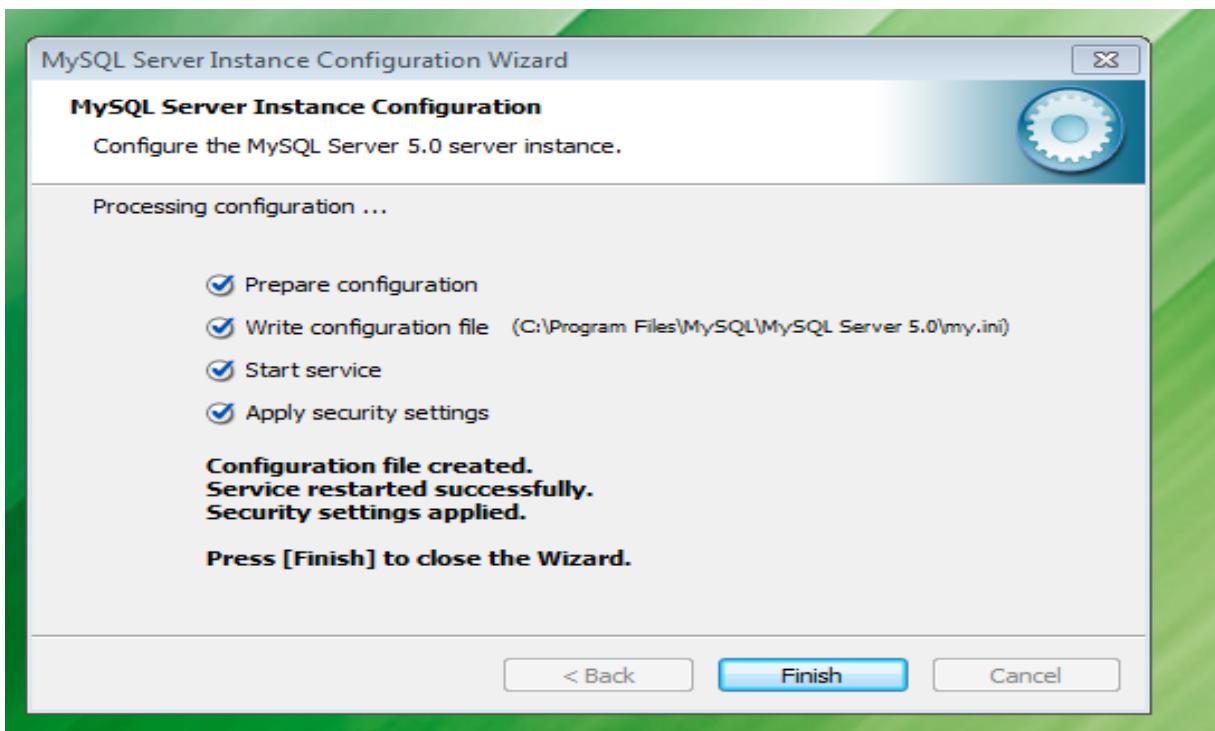
- Click on Execute



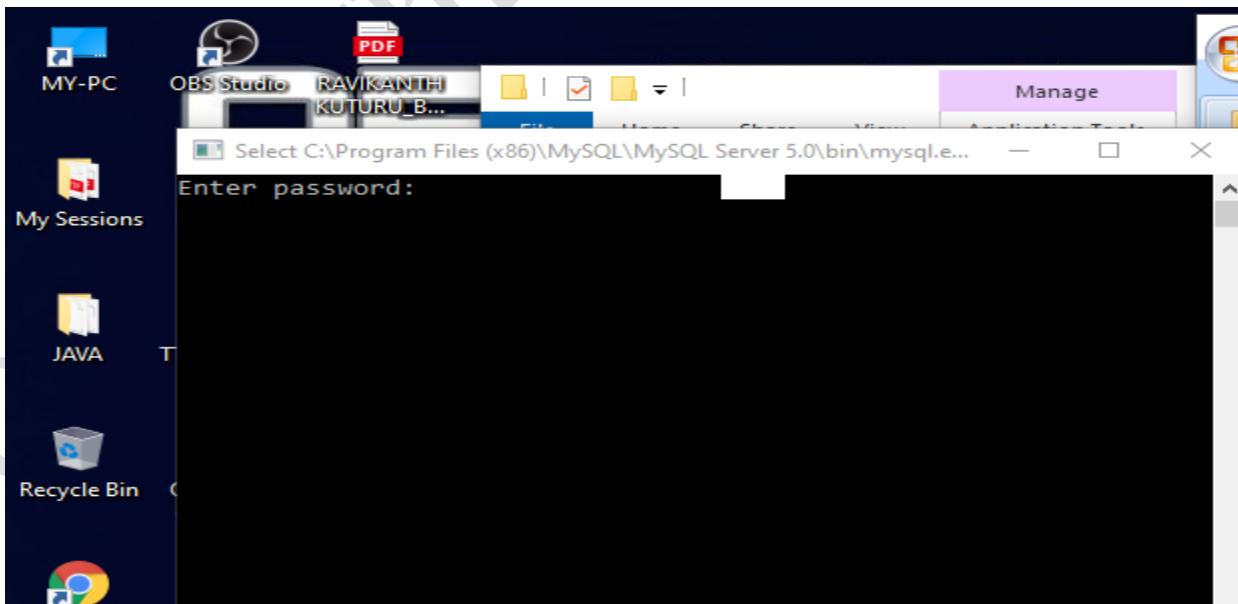
- If you have got this Error ,
- click on Cancel and open Start Menu then type mysql and click on
- MySql Server Instance ConfigWizard



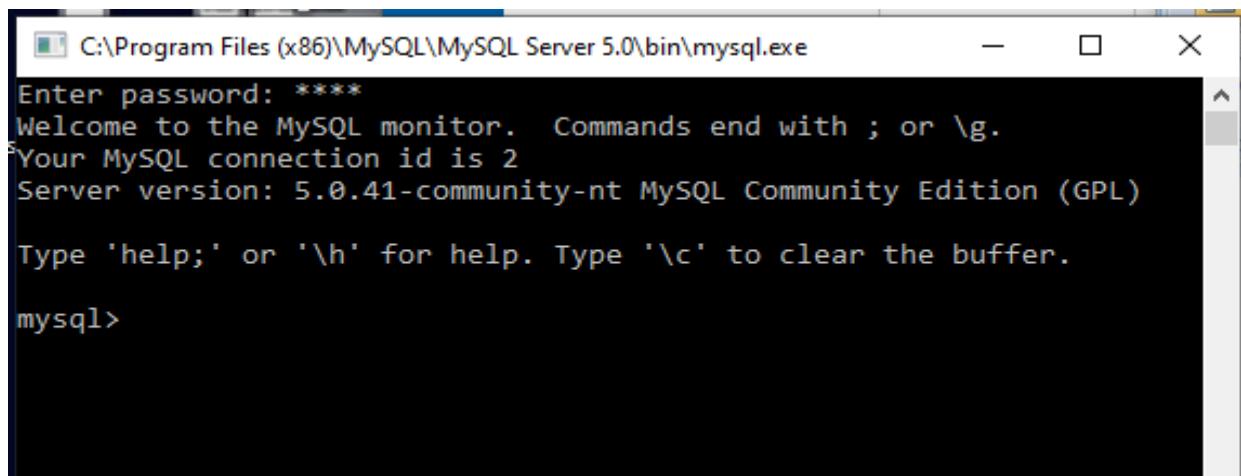
- Click on next and repeat the process



- Finally we will get this screen.. Successfully installed ... click on finish button
- If you get any error repeat the process again.
- Now go to start menu and type MySql Command Line Client



- Type the password which you have given earlier in the configuration settings
- If successful you will get this screen

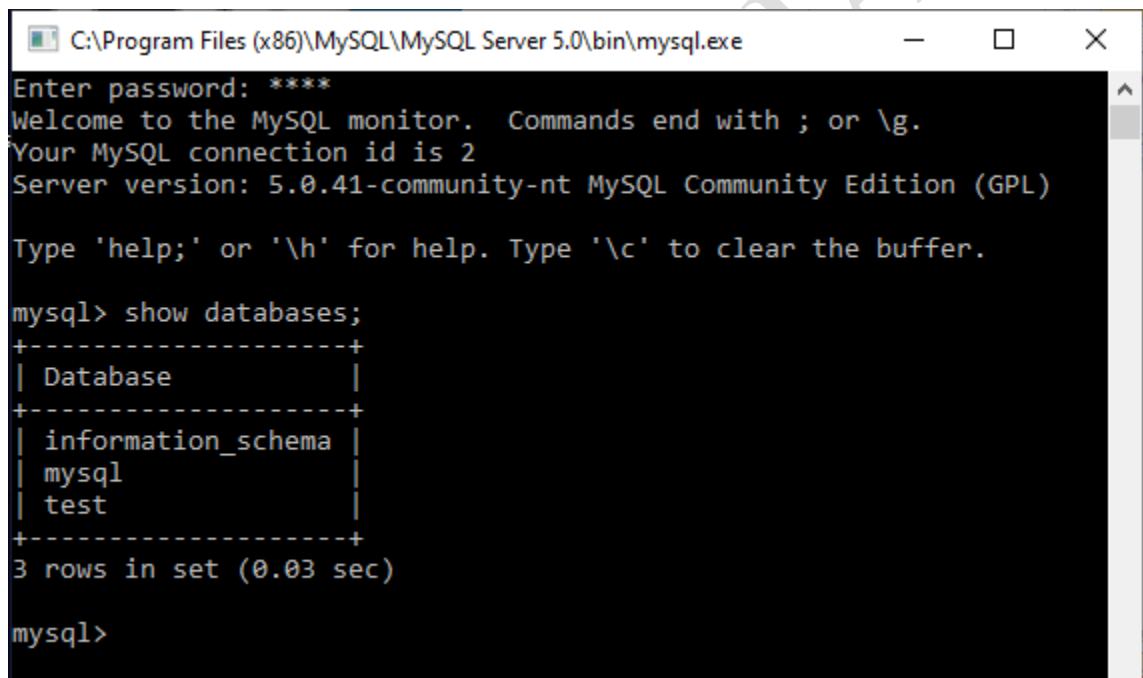


```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.0.41-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

- Follow the Screenshot procedure and create the database and tables and insert the records



```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.0.41-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| test           |
+-----+
3 rows in set (0.03 sec)

mysql>
```

```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.0.41-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| employees |
| mysql |
| rgukt_cse |
| rgukt_employees |
| test |
+-----+
6 rows in set (0.00 sec)

mysql> create database RGUKT_ECE_E2;
Query OK, 1 row affected (0.05 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| employees |
| mysql |
| rgukt_cse |
| rgukt_ece_e2 |
| rgukt_employees |
| test |
+-----+
7 rows in set (0.00 sec)

mysql>
```

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| employees      |
| mysql          |
| rgukt_cse     |
| rgukt_ece_e2   |
| rgukt_employees |
| test           |
+-----+
7 rows in set (0.00 sec)

mysql> use rgukt_ece_e2;
Database changed
mysql>
```

```
mysql> use rgukt_ece_e2;
Database changed
mysql> create table ECE_OOPS(Student_Id int(10),Student_Name varchar(20), ContactNumber int(10), Gender varchar(6), Marks int(5));
Query OK, 0 rows affected (0.22 sec)

mysql> desc ece_oops;
+-----+-----+-----+-----+-----+
| Field        | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Student_Id   | int(10)    | YES  |     | NULL    |         |
| Student_Name | varchar(20) | YES  |     | NULL    |         |
| ContactNumber | int(10)    | YES  |     | NULL    |         |
| Gender        | varchar(6) | YES  |     | NULL    |         |
| Marks         | int(5)     | YES  |     | NULL    |         |
+-----+-----+-----+-----+-----+
5 rows in set (0.09 sec)

mysql>
```

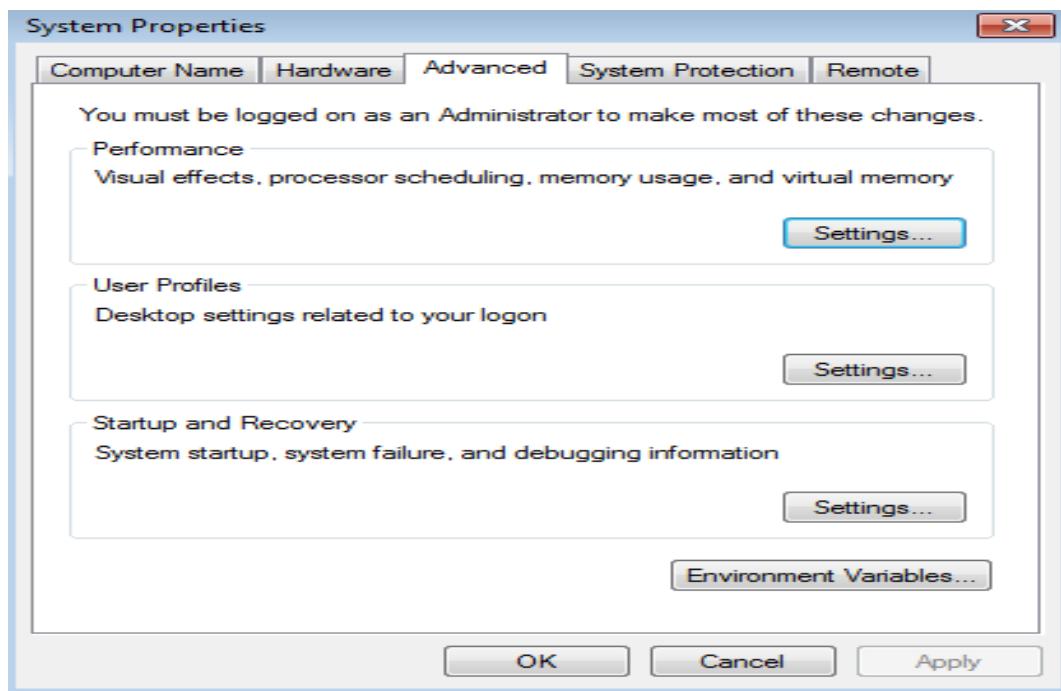
```
mysql> insert into ECE_OOPS values(1230,'RAVIKANTH',924744876,'MALE',96);
Query OK, 1 row affected (0.05 sec)

mysql> select * from ECE_OOPS;
+-----+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+-----+
| 1230 | RAVIKANTH | 924744876 | MALE | 96 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

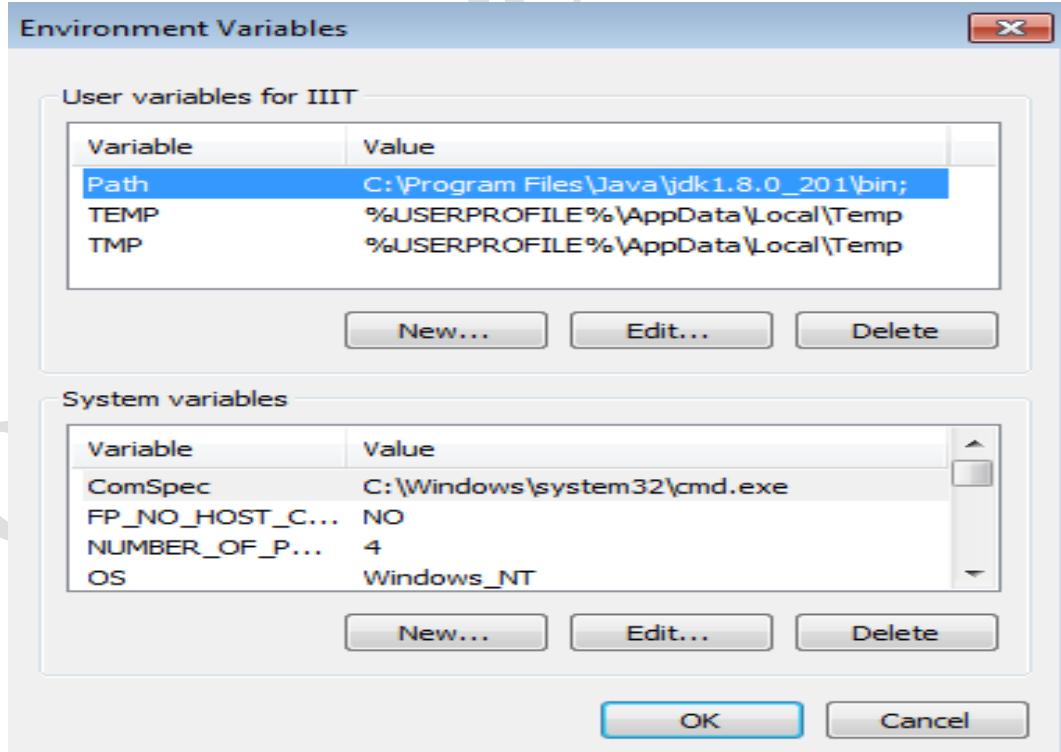
mysql>
```

- Successfully we have created database and table in MySql DB.
- Dear Students if you want to learn more about Database you may refer <https://www.javatpoint.com/what-is-database>

- Now go to start menu and click on Environment System variables and set the CLASS PATH

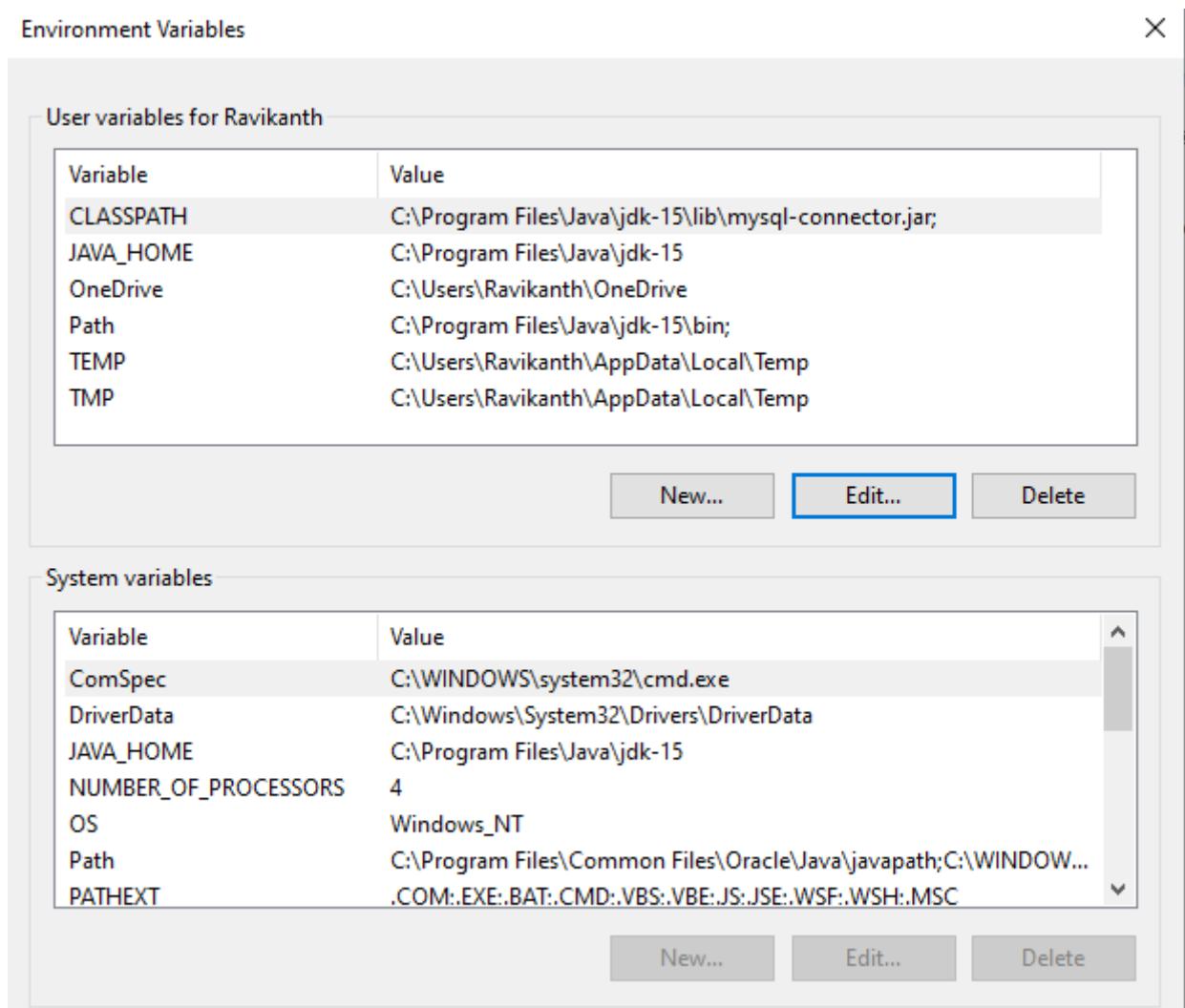


- Click on Environment variables



- Paste your file in this location:

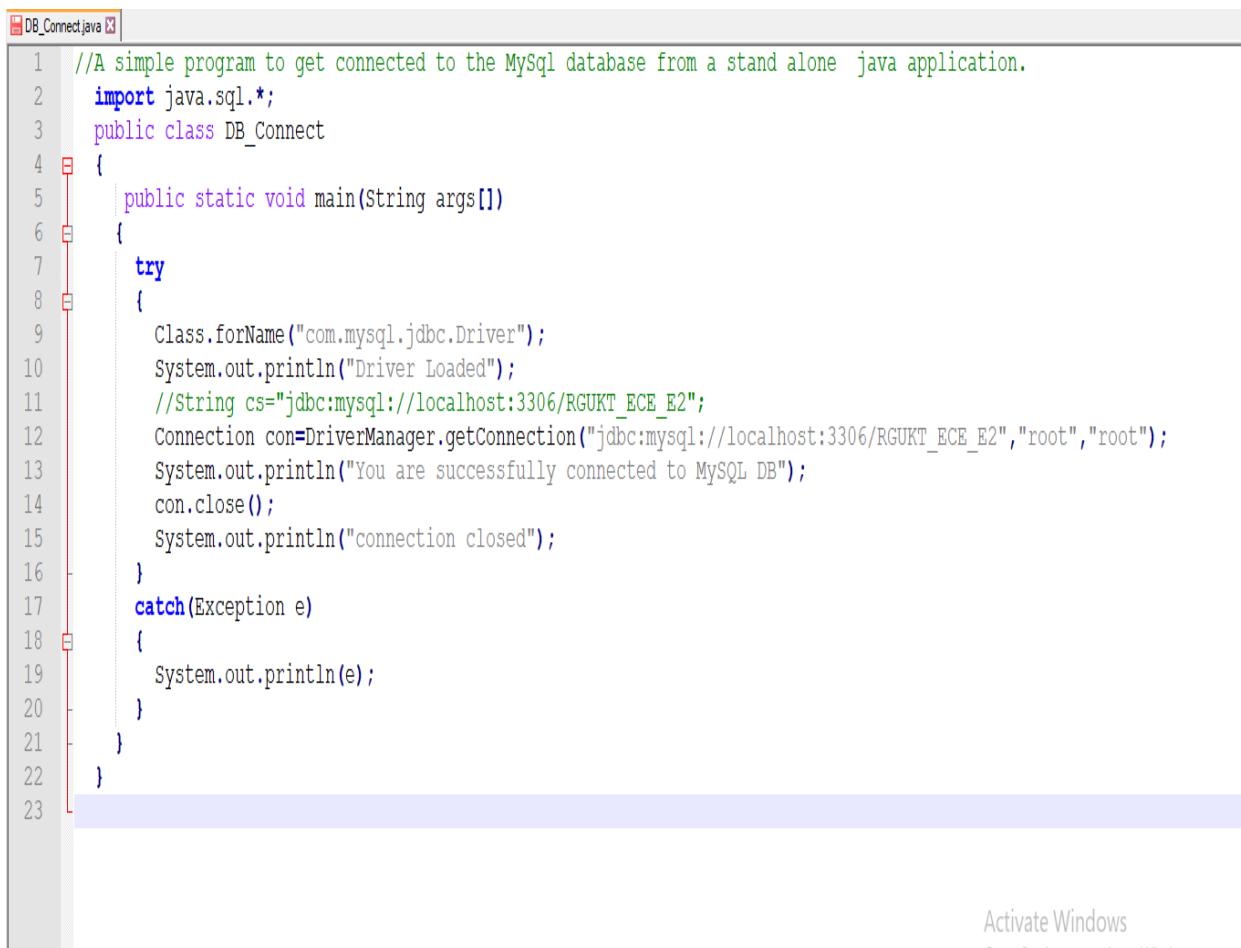
- C:\Program Files\Java\jdk-15\lib\mysql-connector.jar



CLASSPATH set Successfully

## JDBC Driver

- You can choose the right database for your application. Java Database API supports a wide range of database servers such as –  
Ex: MySQL , Oracle, MS-Access, Sybase so on...
- You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database packages
- This API includes the following:
  - Importing the API module.
  - Acquiring a connection with the database.
  - Issuing SQL statements and stored procedures.
  - Closing the connection

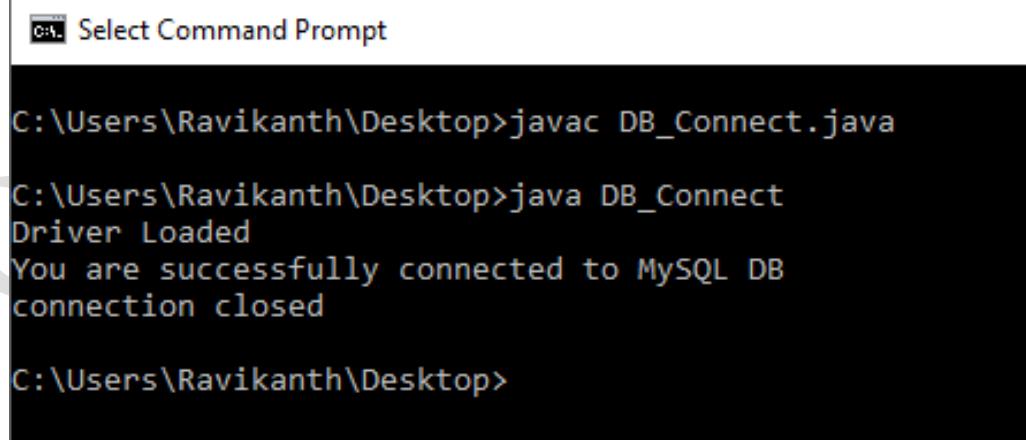
**Example 1: TO connect a Java Program to MySql Database**


```

DB_Connect.java
1 //A simple program to get connected to the MySql database from a stand alone java application.
2 import java.sql.*;
3 public class DB_Connect
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            String cs="jdbc:mysql://localhost:3306/RGUKT_ECE_E2";
12            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/RGUKT_ECE_E2","root","root");
13            System.out.println("You are successfully connected to MySQL DB");
14            con.close();
15            System.out.println("connection closed");
16        }
17        catch(Exception e)
18        {
19            System.out.println(e);
20        }
21    }
22 }
23

```

Activate Windows  
Go to Settings to activate Windows

**Lets us Execute**


```

Select Command Prompt

C:\Users\Ravikanth\Desktop>javac DB_Connect.java
C:\Users\Ravikanth\Desktop>java DB_Connect
Driver Loaded
You are successfully connected to MySQL DB
connection closed

C:\Users\Ravikanth\Desktop>

```

\*\*\*\*\*

### **SYLLABUS:**

#### **UNIT-5:**

##### **JDBC, ODBC Drivers:**

###### **JDBC Driver**

- You can choose the right database for your application. Java Database API supports a wide range of database servers such as –  
Ex: MySQL , Oracle, MS-Access, Sybase so on...
- You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database packages
- This API includes the following:
  - Importing the API module.
  - Acquiring a connection with the database.
  - Issuing SQL statements and stored procedures.
  - Closing the connection

## Seven Steps to JDBC [ Java Database Connectivity], Importing java SQL Packages, Loading & Registering the drivers, Establishing connection, Creating & Executing the statement.

Java JDBC is a Java API to connect and execute query with the database  
Steps to connect to database to Java Program

### **Step1: Import the required package**

```
import java.sql.*;
```

### **Step2: Load and Register with the Driver Class**

The **forName( )** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

forName() method is used to load the Driver Class

**Syntax:** Class.forName("Connection String"); // Depends on DB

Ex:Class.forName("com.mysql.jdbc.Driver"); // **For MySQL**

Ex: Class.forName("Oracle.jdbc.driver.OracleDriver"); // **For Oracle**

Ex: Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // **For MsAccess**

### **Step3: Create the Connection Object**

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

The **getConnection()** method of DriverManager class is used to establish connection with the database.

**Syntax:** public static Connection getConnection(String url, String name, String password)

```
Connection con=DriverManager.getConnection(jdbc:mysql://localhost/"Database Name","Username",Password"); // My SQL
```

**Step4: Create Statement Object**

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

The `createStatement()` method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

It is used to execute queries with the Database

```
Statement st=con.createStatement();
```

**Commonly used methods of Statement interface:**

The important methods of Statement interface are as follows:

- 1) **public ResultSet executeQuery(String sql):** is used to execute SELECT query.  
It returns the object of ResultSet.
- 2) **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- 3) **public boolean execute(String sql):** is used to execute queries that may return multiple results.

**Step5: Retrieve the Results**

The object of ResultSet interface maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

**We can move the cursor by using**

**next() – next method**

**,previous()- Moves the cursor to the previous row of the resultset**

**,first()- Moves the cursor to the first row of the resultset**

**,last()- Moves the cursor to the last row of the resultset**

**,getInt()-**

**,getString() methods**

**Absolute(int row)**

**getRow()- returns the current row number**

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

```
Reultset rs=st.executeQuery("Select * from rgukt_ece_e2"); // Sending the SQL Query
```

### **Step6: Process the Results**

Execute the Query

```
while(rs.next())
```

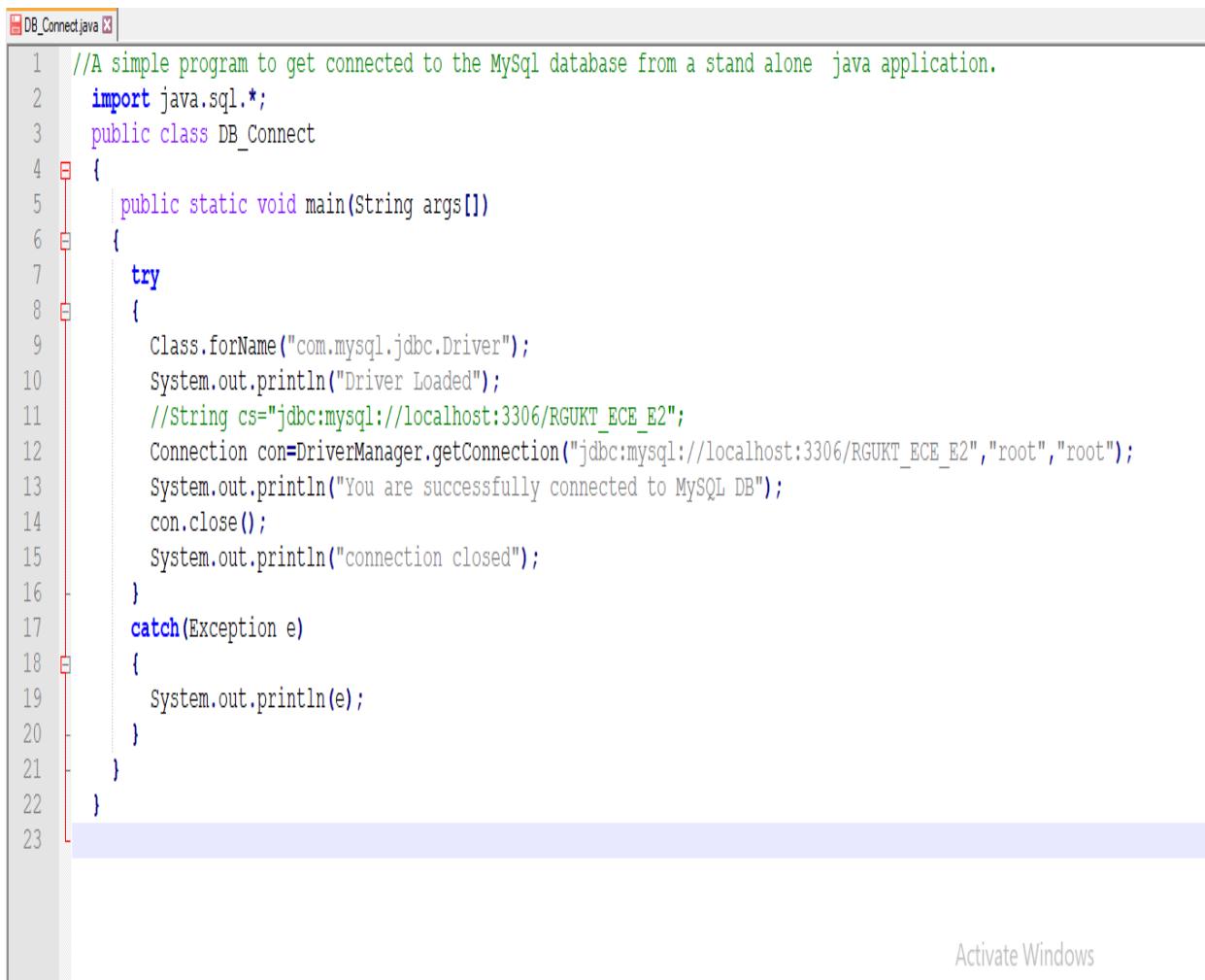
```
rs.getInt(); or rs.getString(); or rs.getString(2);depends on data first ,last
```

### **Step7: Close the Connection**

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

```
con.close();
```

## Example 1: To connect a Java Program to MySql Database



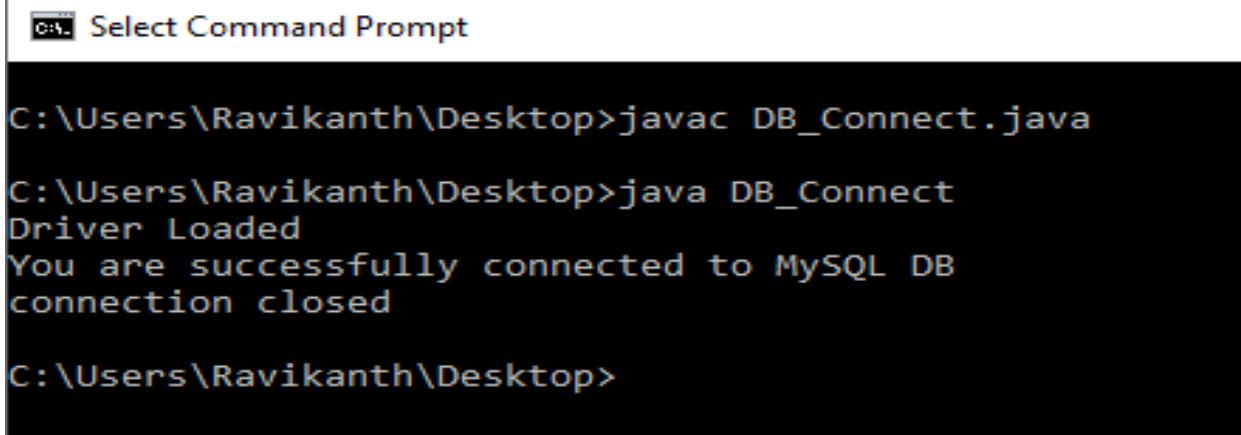
```

DB_Connect.java
1 //A simple program to get connected to the MySQL database from a stand alone java application.
2 import java.sql.*;
3 public class DB_Connect
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            //String cs="jdbc:mysql://localhost:3306/RGUKT_ECE_E2";
12            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/RGUKT_ECE_E2","root","root");
13            System.out.println("You are successfully connected to MySQL DB");
14            con.close();
15            System.out.println("connection closed");
16        }
17        catch(Exception e)
18        {
19            System.out.println(e);
20        }
21    }
22 }

```

Activate Windows  
Go to Settings to activate Windows

Lets us Execute



```

Select Command Prompt

C:\Users\Ravikanth\Desktop>javac DB_Connect.java

C:\Users\Ravikanth\Desktop>java DB_Connect
Driver Loaded
You are successfully connected to MySQL DB
connection closed

C:\Users\Ravikanth\Desktop>

```

## Example 2: Write a Java Program to fetch the records from the database

'C:\Users\Ravikanth\Desktop\DB\_Connect1.java - Notepad+

```

1 //A simple program to get connected to the Mysql database from a stand alone java application using Select Query.
2 import java.sql.*;
3 public class DB_Connect1
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            //String cs="jdbc:mysql://localhost:3306/RGUKT_ECE_E2";
12            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/rgukt_ece_e2","root","root");
13            System.out.println("connected");
14            Statement st=con.createStatement();
15            ResultSet rs=st.executeQuery("Select * from ece_oops");
16            while(rs.next())
17            {
18                System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3)+" "+rs.getString(4)+" "+rs.getInt(5)+"");
19            }
20            con.close();
21            System.out.println("connection closed");
22        }
23        catch(Exception e)
24        {
25            System.out.println(e);
26        }
27    }
28 }

```

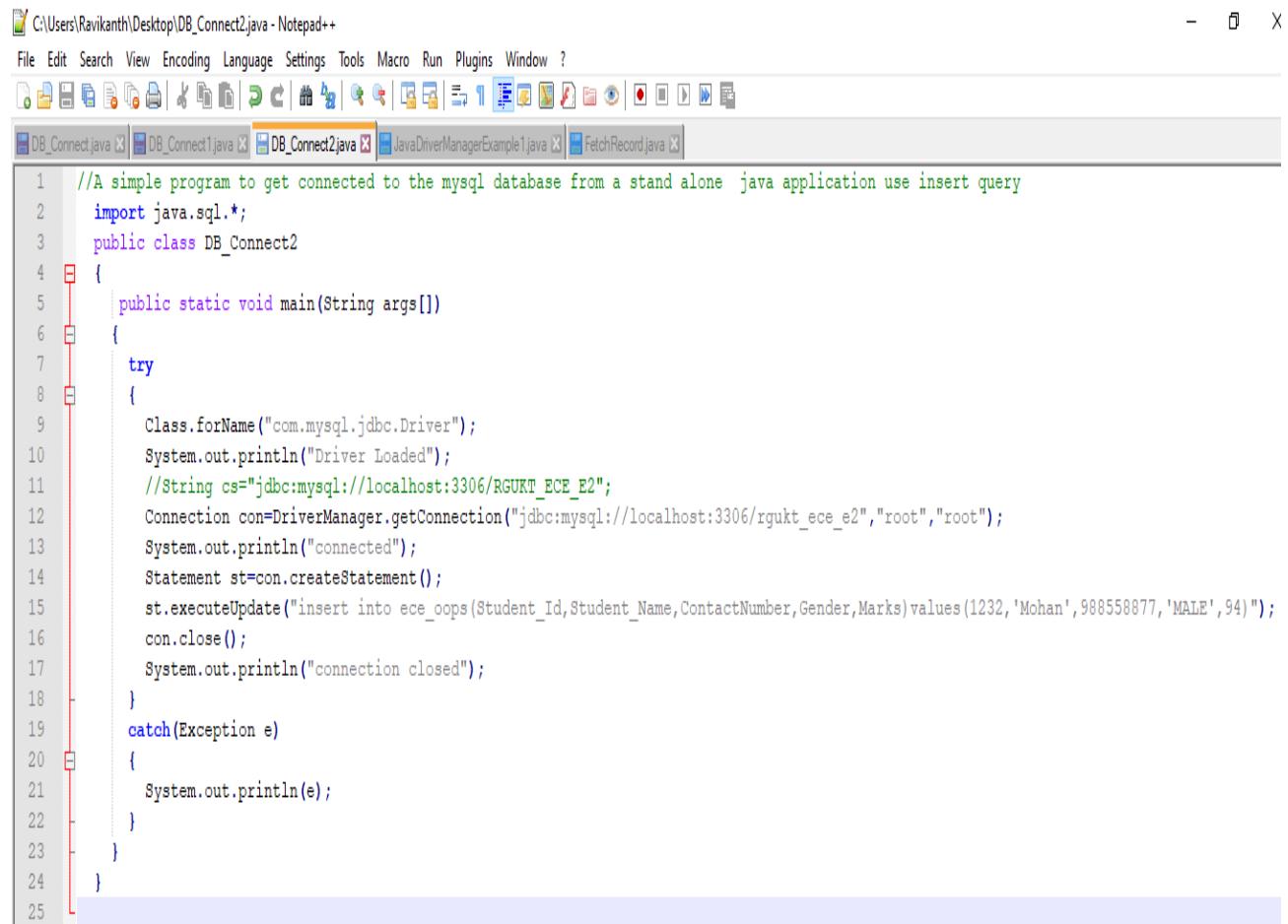
Command Prompt

```

C:\Users\Ravikanth\Desktop>javac DB_Connect1.java
C:\Users\Ravikanth\Desktop>java DB_Connect1
Driver Loaded
connected
1230 RAVIKANTH 924744876 MALE 96
connection closed

C:\Users\Ravikanth\Desktop>

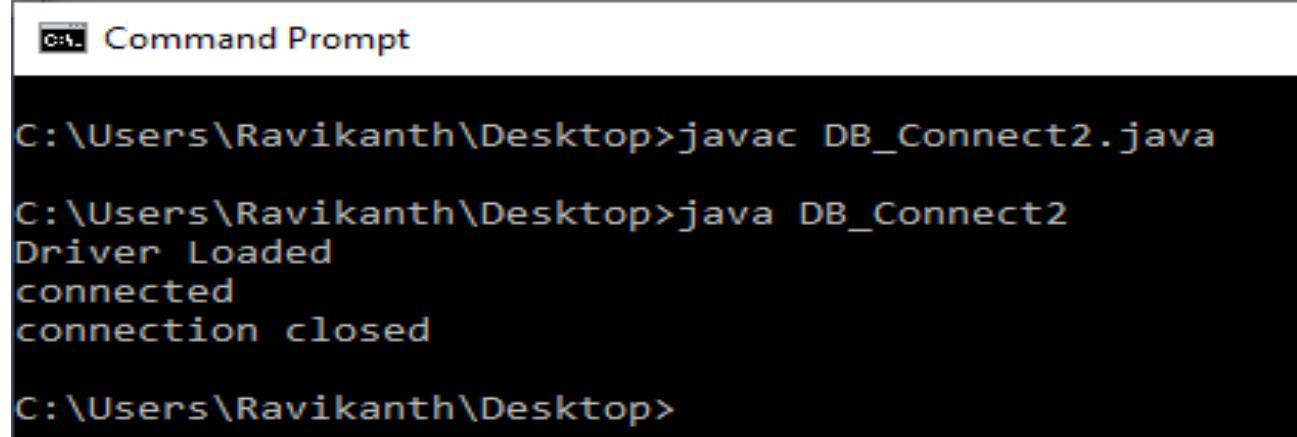
```

**Example 3: Write a Java Program to insert the records into the database**


```

1 //A simple program to get connected to the mysql database from a stand alone java application use insert query
2 import java.sql.*;
3 public class DB_Connect2
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            //String cs="jdbc:mysql://localhost:3306/RGUKT_ECE_E2";
12            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/rgukt_ece_e2","root","root");
13            System.out.println("connected");
14            Statement st=con.createStatement();
15            st.executeUpdate("insert into ece_oops(Student_Id,Student_Name,ContactNumber,Gender,Marks)values(1232,'Mohan',988558877,'MALE',94)");
16            con.close();
17            System.out.println("connection closed");
18        }
19        catch(Exception e)
20        {
21            System.out.println(e);
22        }
23    }
24 }
25

```



```

C:\Users\Ravikanth\Desktop>javac DB_Connect2.java
C:\Users\Ravikanth\Desktop>java DB_Connect2
Driver Loaded
connected
connection closed
C:\Users\Ravikanth\Desktop>

```

```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe

mysql> use rgukt_ece_e2;
Database changed
mysql> select * from ece_oops
-> ;
+-----+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+-----+
| 1230 | RAVIKANTH | 924744876 | MALE | 96 |
+-----+-----+-----+-----+
1 row in set (0.08 sec)

mysql> select * from ece_oops;
+-----+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+-----+
| 1230 | RAVIKANTH | 924744876 | MALE | 96 |
| 1232 | Mohan | 988558877 | MALE | 94 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

**Example 4: Write a JDBC Program for Driver and DriverManager Class**

C:\Users\Ravikanth\Desktop\JavaDriverManagerExample1.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

JavaDriverManagerExample1.java FetchRecord.java

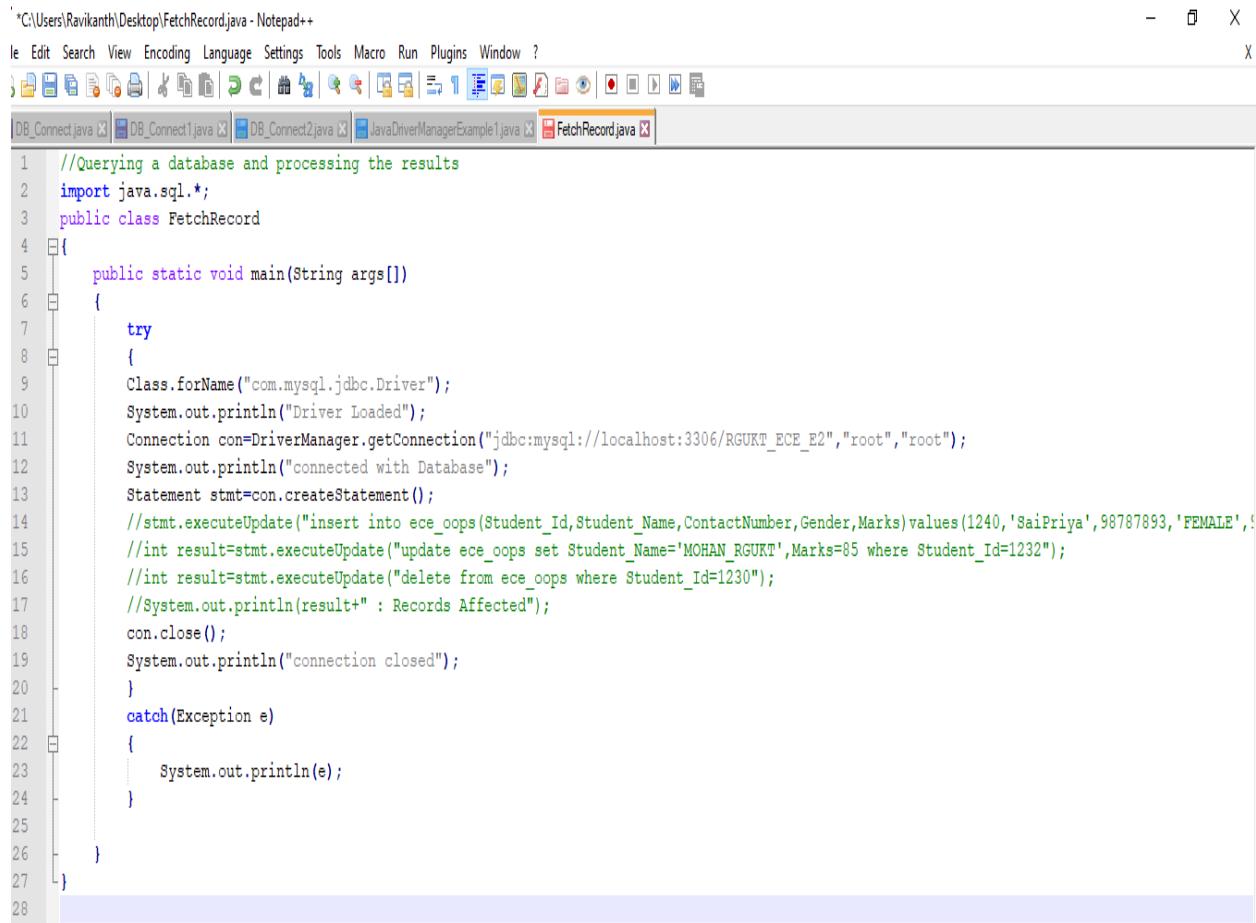
```
1 import java.sql.Driver;
2 import java.sql.DriverManager;
3 import java.sql.SQLException;
4
5 public class JavaDriverManagerExample1
6 {
7     public static void main(String args[]) throws SQLException
8     {
9         Driver d=new com.mysql.jdbc.Driver();
10        DriverManager.registerDriver(d);
11        System.out.println("Driver successfully registered !");
12    }
13 }
14
```

Command Prompt

```
C:\Users\Ravikanth\Desktop>javac JavaDriverManagerExample1.java
C:\Users\Ravikanth\Desktop>java JavaDriverManagerExample1
Driver successfully registered !
```

**Example 5: Write a JDBC Program for querying a database and processing the results.**

\*C:\Users\Ravikanth\Desktop\FetchRecord.java - Notepad++



```

1 //Querying a database and processing the results
2 import java.sql.*;
3 public class FetchRecord
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/RGUKT_ECE_E2","root","root");
12            System.out.println("connected with Database");
13            Statement stmt=con.createStatement();
14            //stmt.executeUpdate("insert into ece_oops(Student_Id,Student_Name,ContactNumber,Gender,Marks)values(1240,'SaiPriya',98787893,'FEMALE',85)");
15            //int result=stmt.executeUpdate("update ece_oops set Student_Name='MOHAN_RGUKT',Marks=85 where Student_Id=1232");
16            //int result=stmt.executeUpdate("delete from ece_oops where Student_Id=1230");
17            //System.out.println(result+" : Records Affected");
18            con.close();
19            System.out.println("connection closed");
20        }
21        catch(Exception e)
22        {
23            System.out.println(e);
24        }
25    }
26 }
27
28

```

```

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| employees |
| mysql |
| rgukt_cse |
| rgukt_ece_e2 |
| rgukt_ece_oops |
| rgukt_employees |
| test |
+-----+
8 rows in set (0.00 sec)

mysql> use rgukt_ece_e2;
Database changed
mysql>

```

C:\Users\Ravikanth\Desktop\FetchRecord.java - Notepad++

```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
DB_Connect.java DB_Connect1.java DB_Connect2.java JavaDriverManagerExample1.java FetchRecord.java

1 //Querying a database and processing the results
2 import java.sql.*;
3 public class FetchRecord
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/RGUKT_ECE_E2","root","root");
12            System.out.println("connected with Database");
13            Statement stmt=con.createStatement();
14            stmt.executeUpdate("insert into ece_oops(Student_Id,Student_Name,ContactNumber,Gender,Marks)values(1240,'SaiPriya',98787893,'FEMALE',95)");
15            //int result=stmt.executeUpdate("update ece_oops set Student_Name='MOHAN_RGUKT',Marks=85 where Student_Id=1232");
16            //int result=stmt.executeUpdate("delete from ece_oops where Student_Id=1230");
17            //System.out.println(result+" : Records Affected");
18            con.close();
19            System.out.println("connection closed");
20        }
21        catch(Exception e)
22        {
23            System.out.println(e);
24        }
25    }
26}
27

```

### Command Prompt

```

C:\Users\Ravikanth\Desktop>javac FetchRecord.java
C:\Users\Ravikanth\Desktop>java FetchRecord
Driver Loaded
connected with Database
connection closed

C:\Users\Ravikanth\Desktop>

```

```

mysql> select * from ece_oops;
+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+
| 1232 | MOHAN_RGUKT | 988558877 | MALE | 85 |
| 1240 | SaiPriya | 98787893 | FEMALE | 95 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

```

/*C:\Users\Ravikanth\Desktop\FetchRecord.java - Notepad+*/
file Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
DB_Connect.java DB_Connect1.java DB_Connect2.java JavaDriverManagerExample1.java FetchRecord.java
1 //Querying a database and processing the results
2 import java.sql.*;
3 public class FetchRecord
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/RGUKT_ECE_E2","root","root");
12            System.out.println("connected with Database");
13            Statement stmt=con.createStatement();
14
15            int result=stmt.executeUpdate("update ece_oops set Student_Name='MOHAN_RGUKT',Marks=85 where Student_Id=1232");
16            //int result=stmt.executeUpdate("delete from ece_oops where Student_Id=1230");
17            //System.out.println(result+" : Records Affected");
18            con.close();
19            System.out.println("connection closed");
20        }
21        catch(Exception e)
22        {
23            System.out.println(e);
24        }
25    }
26}
27

```

Command Prompt

```

C:\Users\Ravikanth\Desktop>javac FetchRecord.java
C:\Users\Ravikanth\Desktop>java FetchRecord
Driver Loaded
connected with Database
connection closed

C:\Users\Ravikanth\Desktop>

```

```

mysql> select * from ece_oops;
+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+
| 1232 | Mohan | 988558877 | MALE | 94 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from ece_oops;
+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+
| 1232 | MOHAN_RGUKT | 988558877 | MALE | 85 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

\*C:\Users\Ravikanth\Desktop\FetchRecord.java - Notepad++

```

1 //Querying a database and processing the results
2 import java.sql.*;
3 public class FetchRecord
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             Class.forName("com.mysql.jdbc.Driver");
10            System.out.println("Driver Loaded");
11            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/RGUKT_ECE_E2","root","root");
12            System.out.println("connected with Database");
13            Statement stmt=con.createStatement();
14            int result=stmt.executeUpdate("delete from ece_oops where Student_Id=1230");
15            System.out.println(result+" : Records Affected");
16            con.close();
17            System.out.println("connection closed");
18        }
19        catch(Exception e)
20        {
21            System.out.println(e);
22        }
23    }
24 }
25

```

Command Prompt

```

C:\Users\Ravikanth\Desktop>javac FetchRecord.java
C:\Users\Ravikanth\Desktop>java FetchRecord
Driver Loaded
connected with Database
1 : Records Affected
connection closed

C:\Users\Ravikanth\Desktop>

```

```

mysql> select * from ece_oops;
+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+
| 1230 | RAVIKANTH | 924744876 | MALE | 96 |
| 1232 | Mohan | 988558877 | MALE | 94 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from ece_oops;
+-----+-----+-----+-----+
| Student_Id | Student_Name | ContactNumber | Gender | Marks |
+-----+-----+-----+-----+
| 1232 | Mohan | 988558877 | MALE | 94 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

**Assignment Questions:**

1. Write a program to insert data into Student Table.
2. Write a program to retrieve the data from the table Student.
3. Create a Form to insert and retrieve the data from Database as user prefer.
4. Write a program to store an Image and retrieve an image from Database
5. Write a program to Store and retrieve file content from the Data base.

\*\*\*\*\*

**UNIT-5:**

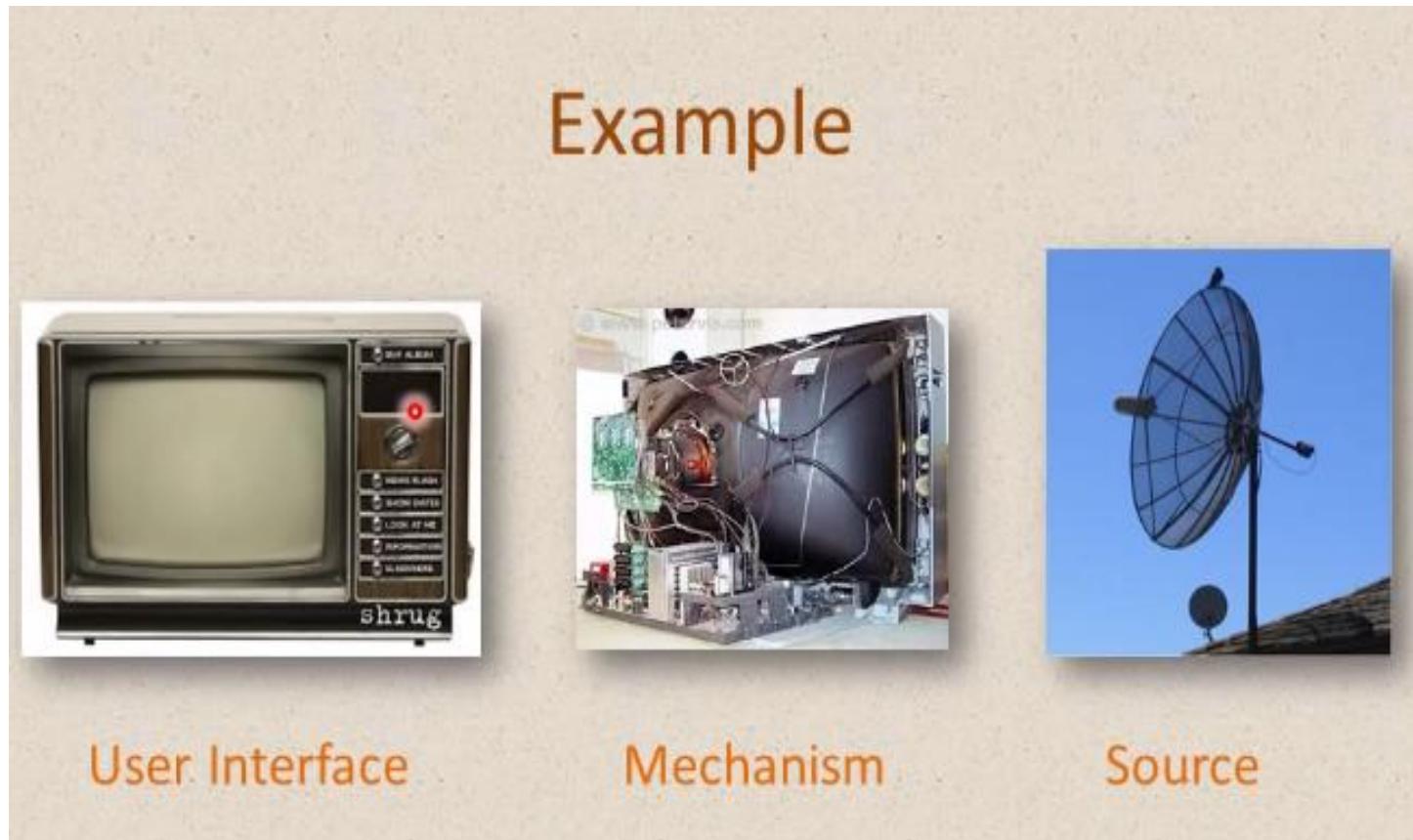
**MVC architecture**

Let us try to understand through a real time examples:

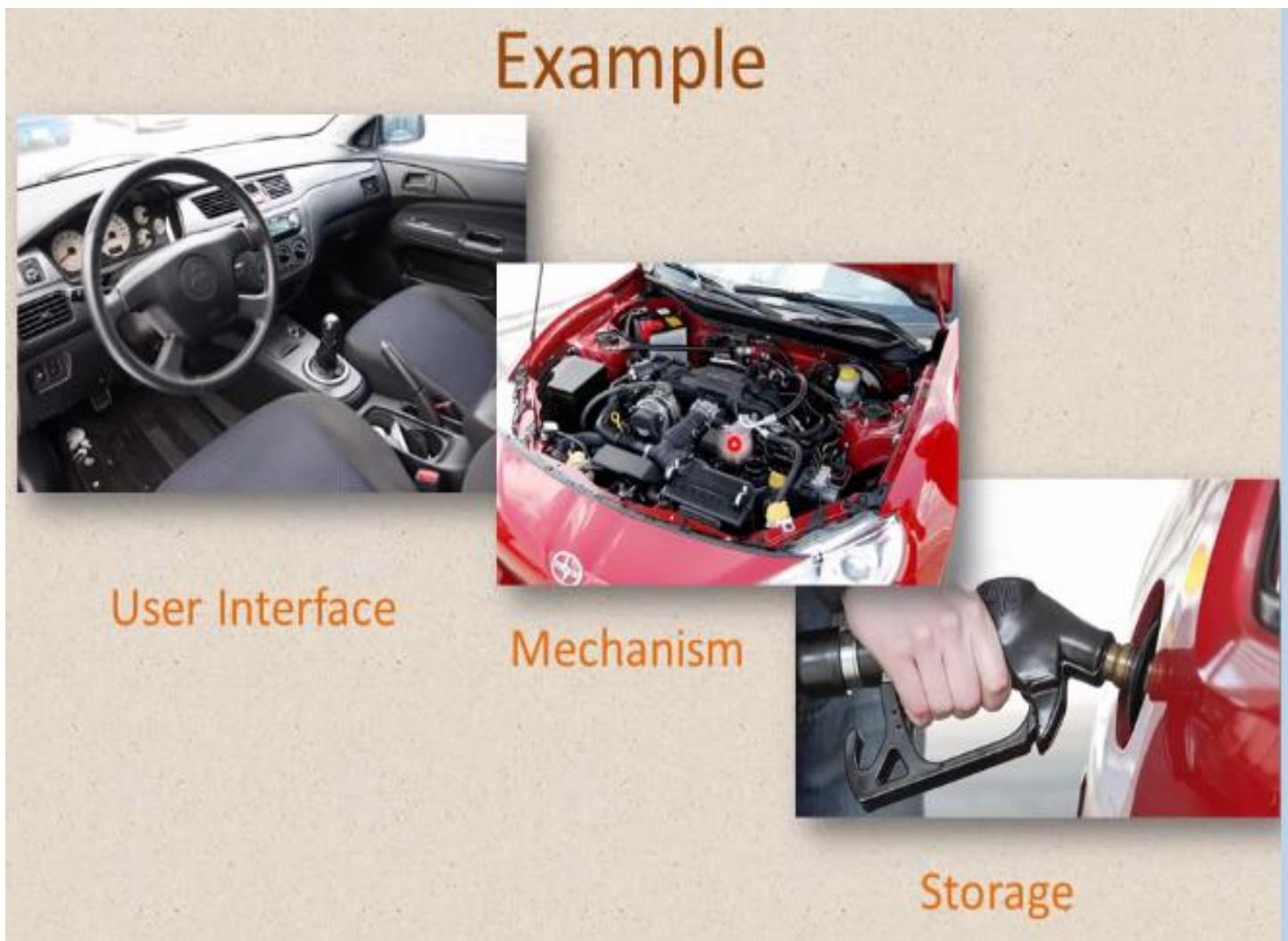
Ex: Any Engineering Products can be categorized as

- 1. User Interface**
- 2. Mechanism**
- 3. Input/Source/Storage**

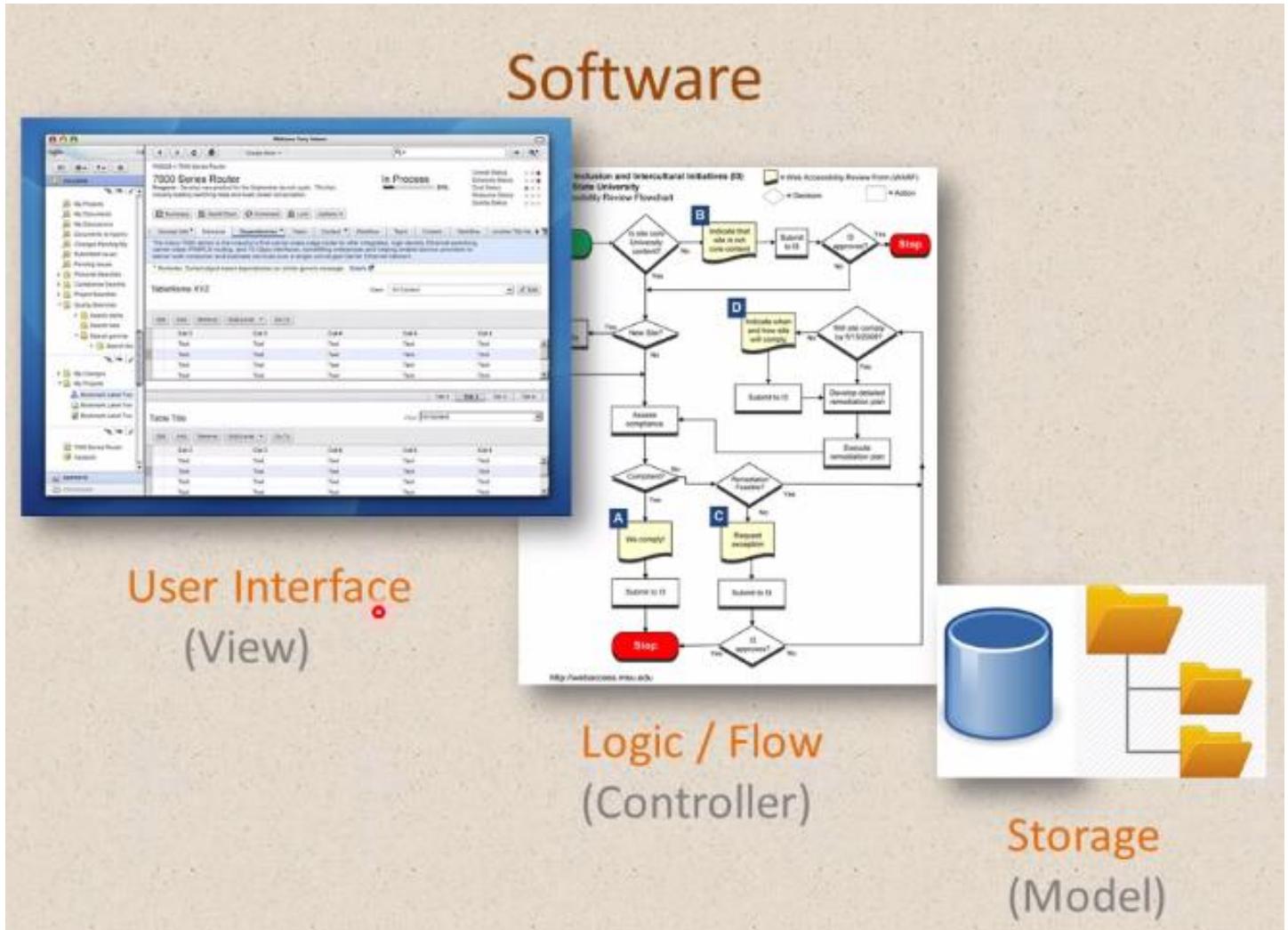
Consider an example of Electronic Engineering Product



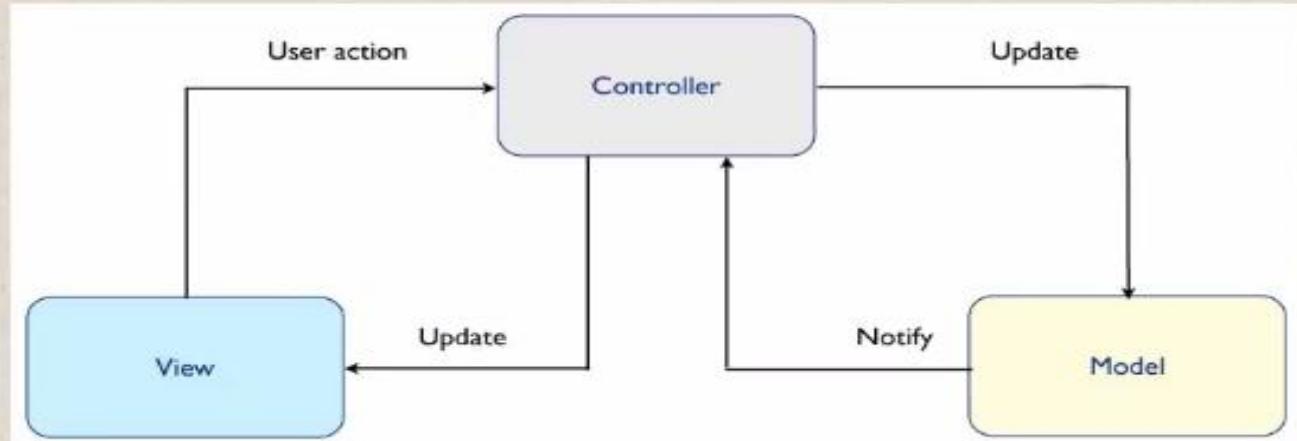
Consider an example of Automobile Engineering Product



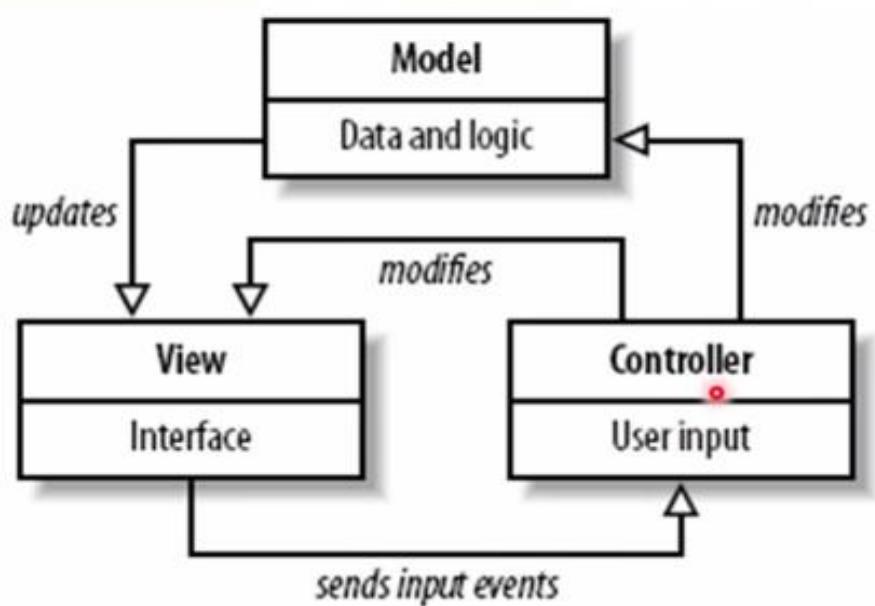
Now Let us consider from a Software System Point of View



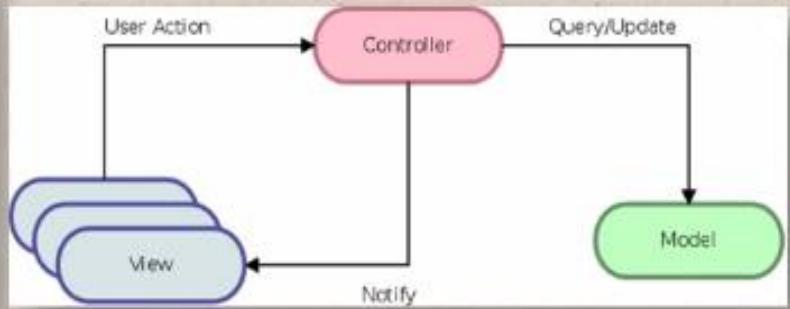
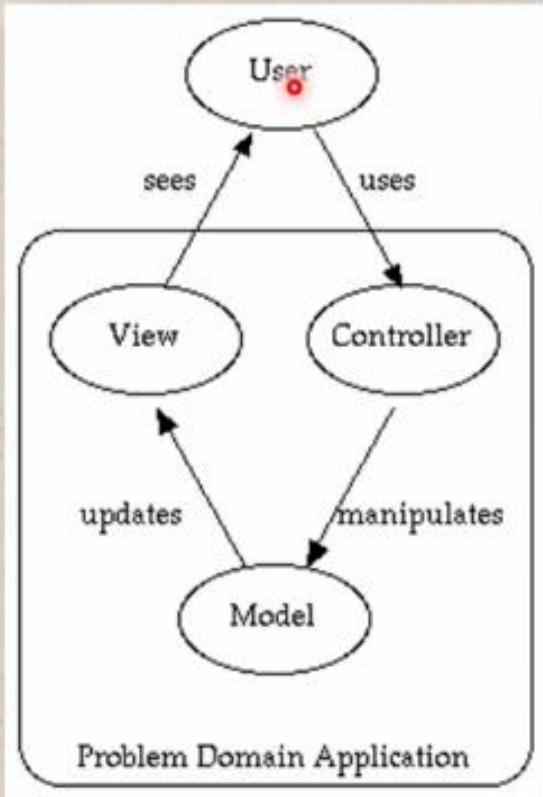
## Model View Controller



## Model View Controller



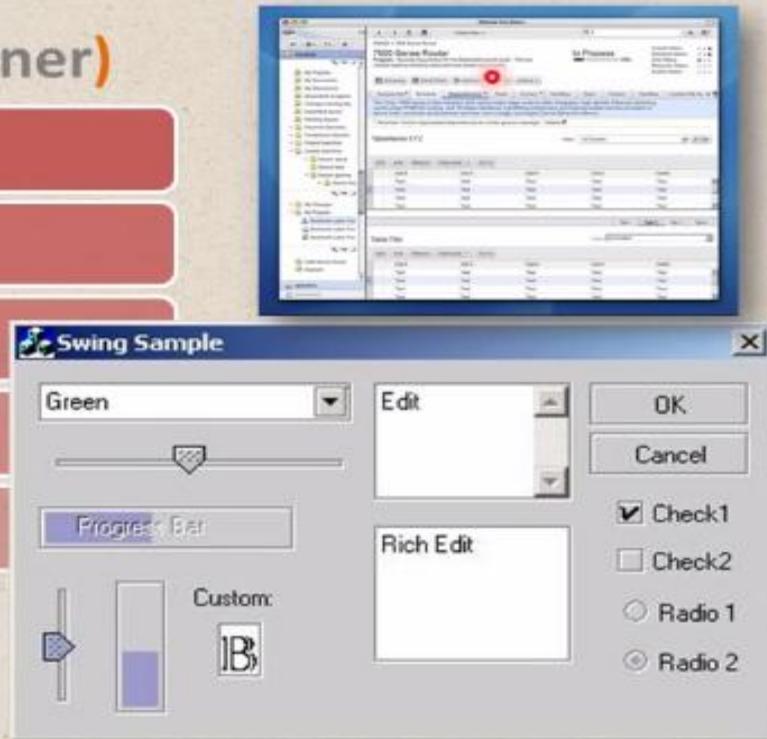
# Model View Controller



# Elements

## User Interface (Designer)

- Text
- Image
- Video
- Animation
- User Controls



# Elements

## Controller (Programmer)

- Control Statements ( if , Loop)
- Interfaces (functions)
- Components



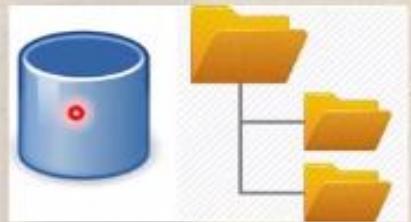
# Elements

## Model (Database Administrator – DBA)

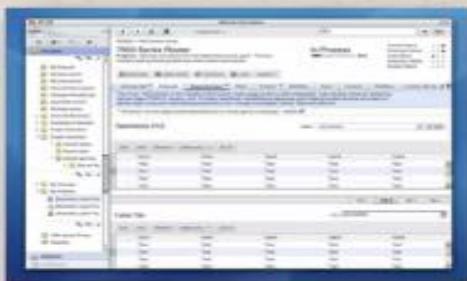
File System

Relational Database

Object-Oriented Database



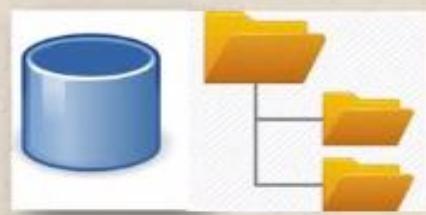
## MVC



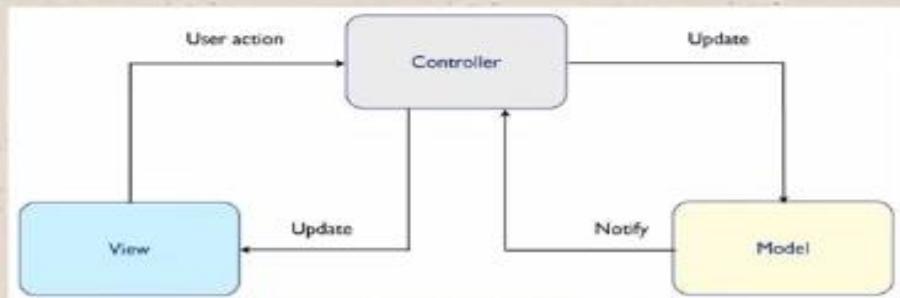
User Interface  
(View)  
Designer



Logic / Flow  
(Controller)  
Programmer



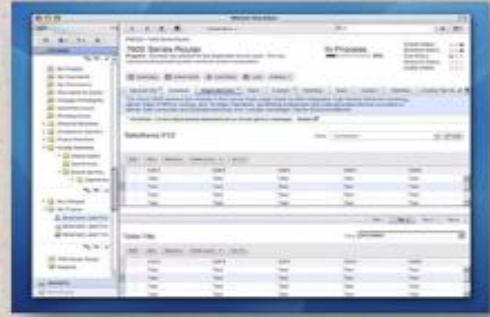
Storage  
(Model)  
DBA



# Technologies

## User Interface (Designers)

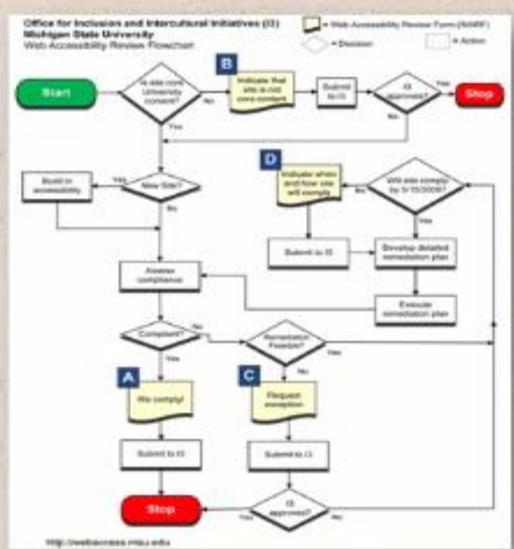
- AWT / Swing – Java
- JSP (Java Server Faces) – Java
- Silver Lite – .Net
- WPF (Windows Presentation Foundation) – .NET
- HTML + CSS
- Flash
- Visualforce (Salesforce)



# Technologies

## Controller (Programmer)

- Core Java , Servlets / JSP – Java
- VB.NET, C#.NET , ASP.NET – .NET
- PHP
- Ruby
- Python
- Apex



# Technologies

## Model (DBA)

Oracle

SQL Server

mysql

DB2

Hibernate Framework

Hadoop



**Note:** As per our syllabus we are going to learn

**AWT Components and Event Handlers:** Abstract window tool kit, Event Handlers, Event Listeners, AWT Controls and Event Handling: Labels, TextComponent, ActionEvent, Buttons, CheckBoxes, ItemEvent, Choice, Scrollbars, Layout Managers- Input Events, Menus, Programs

**GUI Programming with Java** - Introduction to Swing, limitations of AWT, Swing vs AWT, MVC architecture, Hierarchy for Swing components, Containers - JFrame, JApplet, JDialo, JPanel. Overview of some swing components JButton, JLabel, JTextField, JTextArea, simple swing applications.

**JDBC, ODBC Drivers:** JDBC ODBC Bridges, Seven Steps to JDBC, Importing java SQL Packages, Loading & Registering the drivers, Establishing connection. Creating & Executing the statement.

### SYLLABUS:

#### UNIT-5:

**AWT Components and Event Handlers:** Abstract window tool kit, Event Handlers, Event Listeners, AWT Controls and Event Handling: Labels, TextComponent, ActionEvent, Buttons, CheckBoxes, ItemEvent, Choice, Scrollbars, Layout Managers- Input Events, Menus, Programs

#### Advantages of GUI over CUI

- Presents a user-friendly mechanism for interacting with an application.
- Built from GUI components.
- It need more resources
- Speed is less compare to the CUI
  
- In a Command Line Interface, the commands are entered from the keyboard.
- It is not user-friendly.
- Difficult to remember commands.
- It need less resources
- Speed is more compare to the GUI

#### Most modern programs use a GUI.

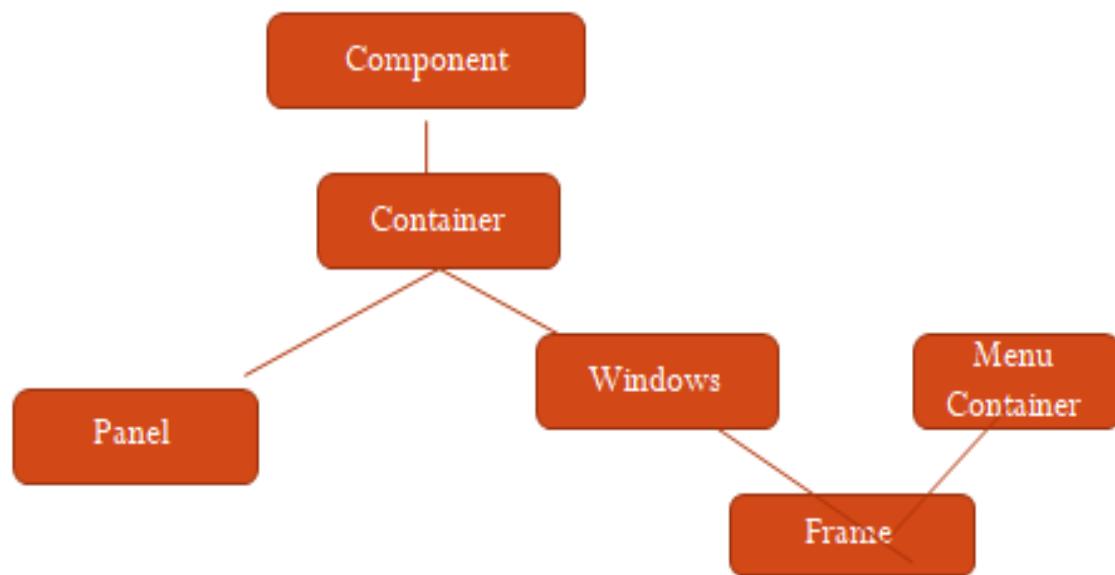
- Graphical: Not just text or characters but windows, menus, buttons, ..
- User: Person using the program
- Interface: Way to interact with the program

#### Graphical Elements include:

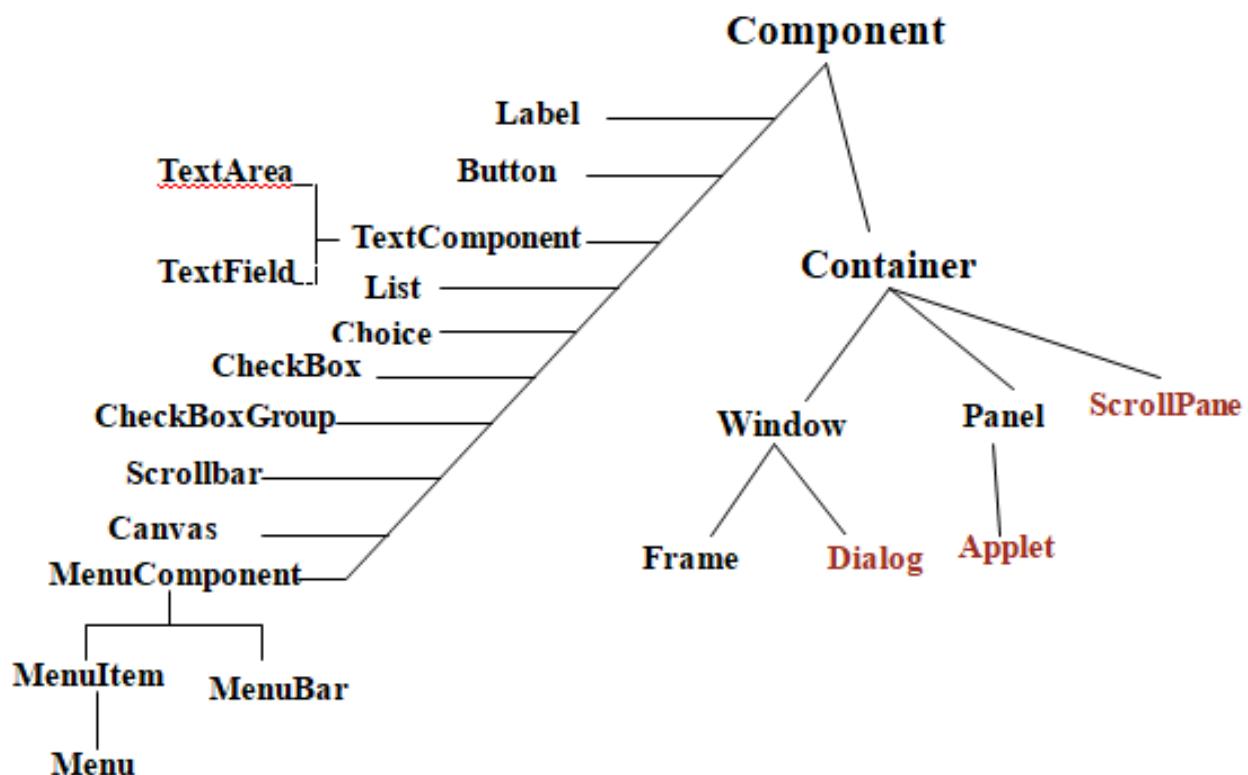
|        |                          |
|--------|--------------------------|
| Window | List                     |
| Choice | Label                    |
| Menu   | Scrollbar                |
| Button | TextComponent      etc., |

**Introduction:**

- **Java AWT** (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The `java.awt` package provides classes for AWT api such as `TextField`, `Label`, `TextArea`, `RadioButton`, `CheckBox`, `Choice`, `List` etc.
- It is logically organized in a top-down, hierarchical fashion, therefore it is easier to understand & use
- The AWT defines windows according to a class hierarchy that adds functionality & specificity with each other



## AWT class hierarchy



## Component

- “Component” class is at the top of the AWT hierarchy
- It is an abstract class that encapsulates all of the attributes of visual component
- All user interface elements that are displayed on the screen & that interact with the user are subclasses of “Component” class
- It defines a no of methods(public) that are responsible for managing events, like mouse, i/p, keyboard i/p, positioning & sizing the window
- An object of “Component” class is responsible for remembering the current foreground color, the current background color & the currently selected text font

## Container

- The “container” class is a subclass of the class “Component”
- It has additional methods that allow other “Component” objects to be nested within it
- Other container objects can also be stored inside a container.
- This makes for a multileveled containment system
- A container is responsible for laying out (positioning) any components that it contains, through the use of various layout managers

## Panel

- The “panel” class is a concrete subclass of the class “container”. It doesnot add any new methods, it simply implements “container”
- A panel may be thought of as a recursively nestable, concrete screen component
- The “panel” class is the superclass for the class “Applet”. When screen output is directed to an applet, it is drawn on the surface of a “Panel Object”
- In essence, a panel is a window that doesnot contain a title bar, menu bar, border

- Other components can be added to a panel object by its method “add()”, which is inherited from the class container
- Once the components have been added, we can position & resize them using the methods “setLocation()”, “setSize()”, “setBounds()” defined by components

## Frame

- The class “Frame” is a sub-class of the class “Window” & has a title bar, menu bar, borders & resizing corners [ set Title(String), get Title( ) , setMenuBar(MenuBar) ]

## Menu Container

- It acts as an interface to the class “Frame” which contains the attributes related to menu bar

## Window

- The class “Window” is a sub-class of “Container”. The class “window” creates a top-level window
- A top-level window is not contained within any other object, it sits directly on the desktop
- Generally we wont create objects for the “window” class directly. Instead, we use a sub-class of Window called Frame

## Canvas

- It is not a part of hierarchy. It is another type of window that we find valuable
- Canvas encapsulates a blank window upon which we can draw

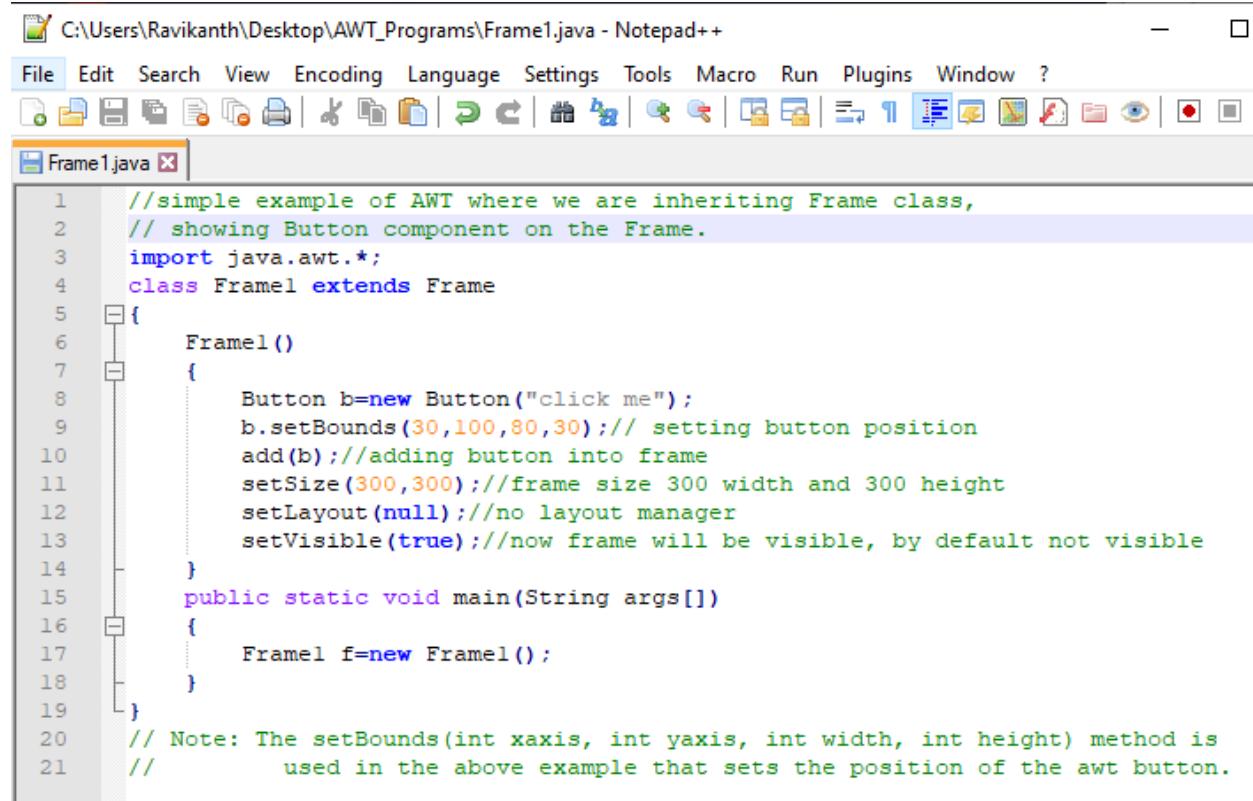
**Useful Methods of Component class**

| Method  | Description   |
|---|---|
| public void add(Component c)  | inserts a component on this component.  |
| public void setSize(int width,int height)                                     | sets the size (width and height) of the component.  |
| public void setLayout(LayoutManager m)  | defines the layout manager for the component.   |
| public void setVisible(boolean status)<br>setLocation(int,int), getLocation() | changes the visibility of the component, by default false.<br>To set and get component location |
| setForeground(Color), getForeground()   | To set and get foreground colors  |
| setBackground(Color), getBackground()   | To set and get background colors  |

**To create simple awt example, you need a frame.**

**There are two ways to create a frame in AWT.**

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

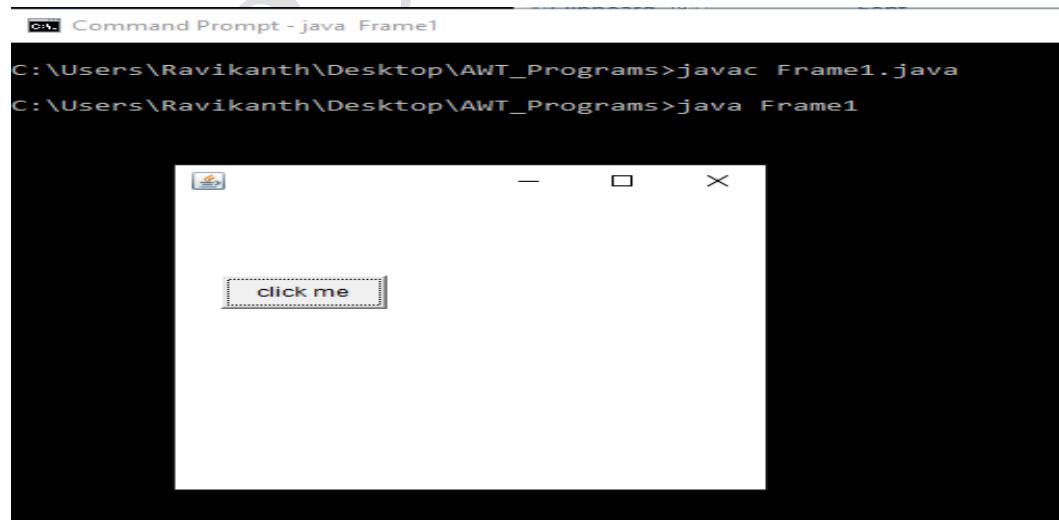
**Example 1:**


```

C:\Users\Ravikanth\Desktop\AWT_Programs\Frame1.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Frame1.java x
1 //simple example of AWT where we are inheriting Frame class,
2 // showing Button component on the Frame.
3 import java.awt.*;
4 class Frame1 extends Frame
5 {
6     Frame1()
7     {
8         Button b=new Button("click me");
9         b.setBounds(30,100,80,30);// setting button position
10        add(b);//adding button into frame
11        setSize(300,300);//frame size 300 width and 300 height
12        setLayout(null);//no layout manager
13        setVisible(true);//now frame will be visible, by default not visible
14    }
15    public static void main(String args[])
16    {
17        Frame1 f=new Frame1();
18    }
19 }
20 // Note: The setBounds(int xaxis, int yaxis, int width, int height) method is
21 //           used in the above example that sets the position of the awt button.

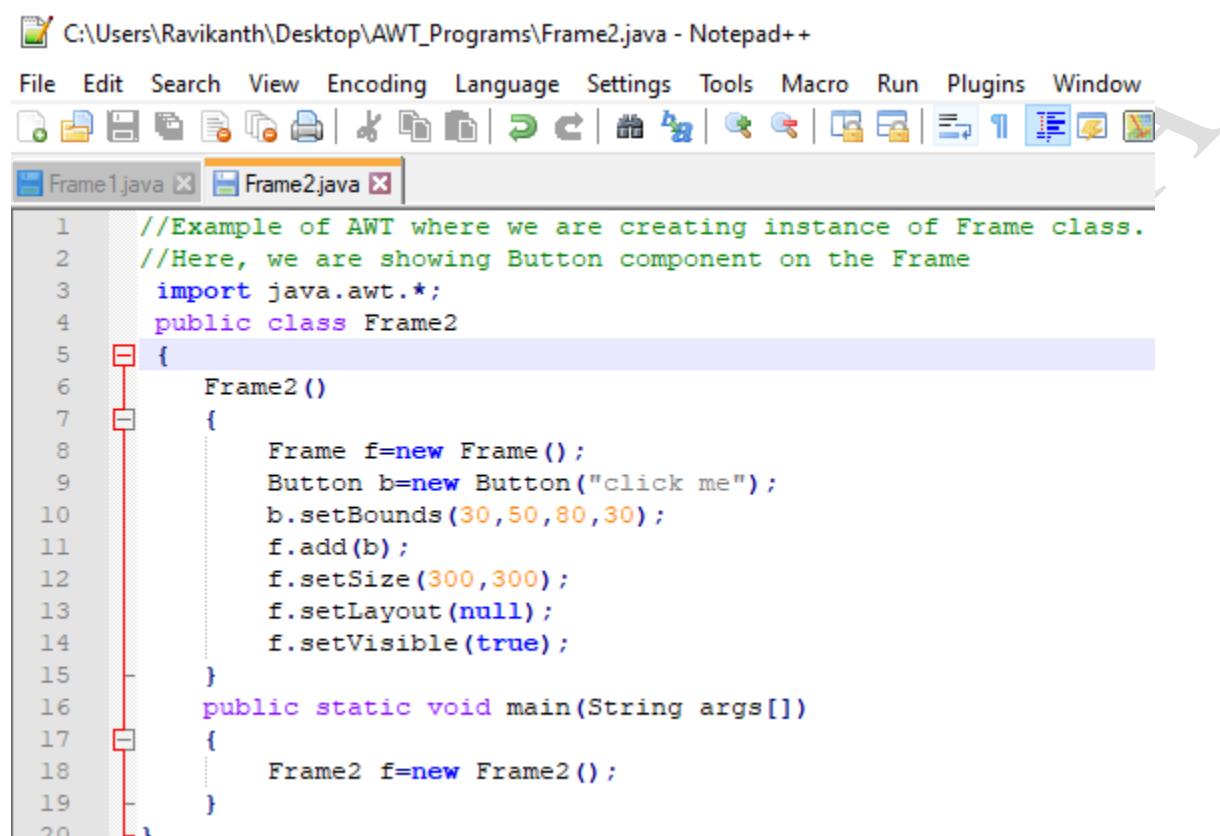
```

**Note:** The `setBounds(int xaxis, int yaxis, int width, int height)` method is used in the above example that sets the position of the awt button.

**Output:**

**Example 2:**

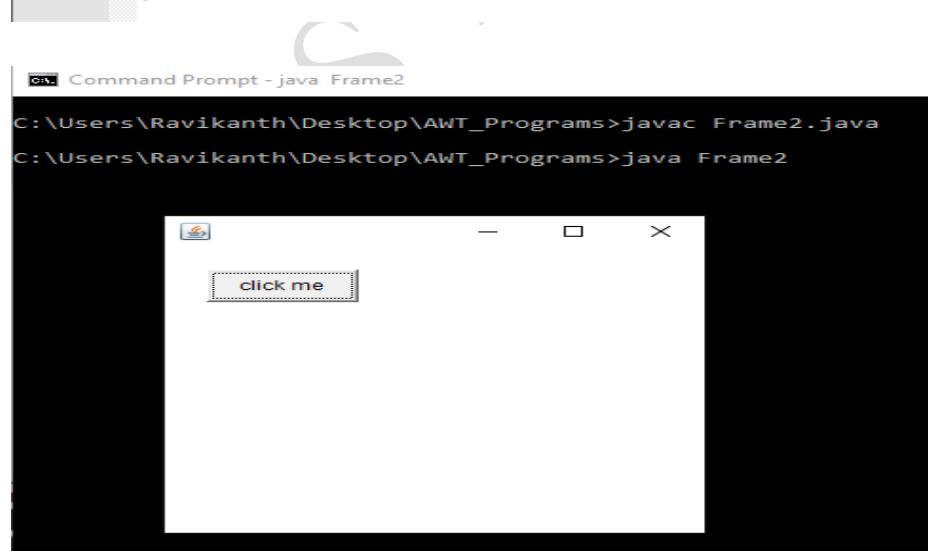
Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.



```

C:\Users\Ravikanth\Desktop\AWT_Programs\Frame2.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
Frame1.java Frame2.java
1 //Example of AWT where we are creating instance of Frame class.
2 //Here, we are showing Button component on the Frame
3 import java.awt.*;
4 public class Frame2
5 {
6     Frame2()
7     {
8         Frame f=new Frame();
9         Button b=new Button("click me");
10        b.setBounds(30,50,80,30);
11        f.add(b);
12        f.setSize(300,300);
13        f.setLayout(null);
14        f.setVisible(true);
15    }
16    public static void main(String args[])
17    {
18        Frame2 f=new Frame2();
19    }
20}

```



```

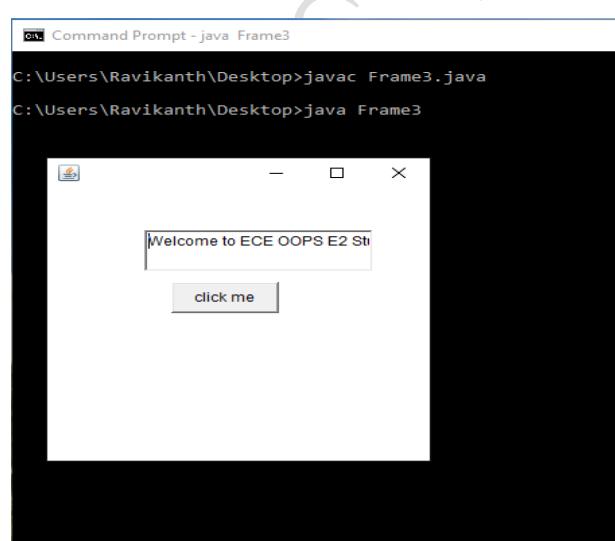
Command Prompt - java Frame2
C:\Users\Ravikanth\Desktop\AWT_Programs>javac Frame2.java
C:\Users\Ravikanth\Desktop\AWT_Programs>java Frame2

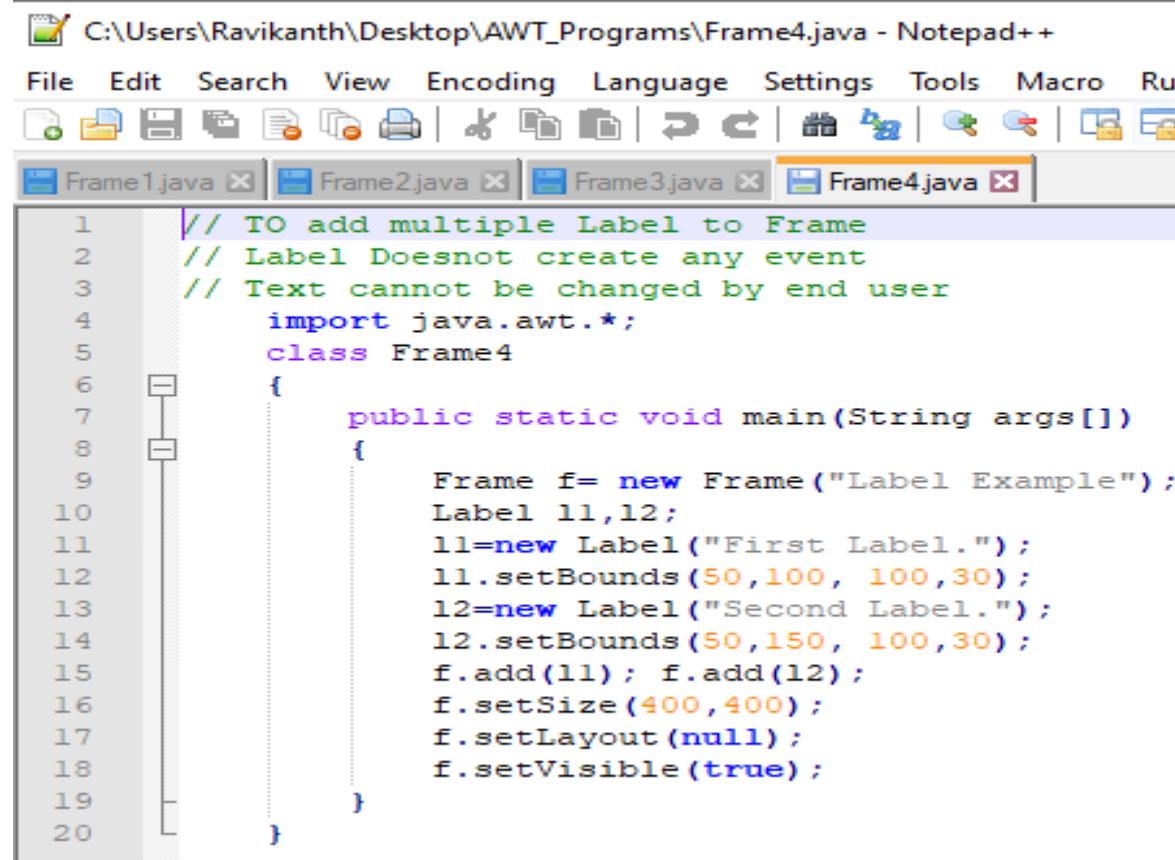
```

### Example 3:

```

1 //Java event handling by implementing ActionListener
2 // To create a TextField and Button
3 import java.awt.*;
4 import java.awt.event.*;
5 class Frame3 extends Frame implements ActionListener
6 {
7     TextField tf;
8     Frame3()
9     {
10         //create components
11         tf=new TextField();
12         tf.setBounds(80,70,170,40);
13         Button b=new Button("click me");
14         b.setBounds(100,120,80,30);
15         //register listener
16         b.addActionListener(this);//passing current instance
17         //add components and set size, layout and visibility
18         add(b);add(tf);
19         setSize(300,300);
20         setLayout(null);
21         setVisible(true);
22     }
23     public void actionPerformed(ActionEvent e)
24     {
25         tf.setText("Welcome to ECE OOPS E2 Students ");
26     }
27     public static void main(String args[])
28     {
29         new Frame3();
30     }
31 }
```



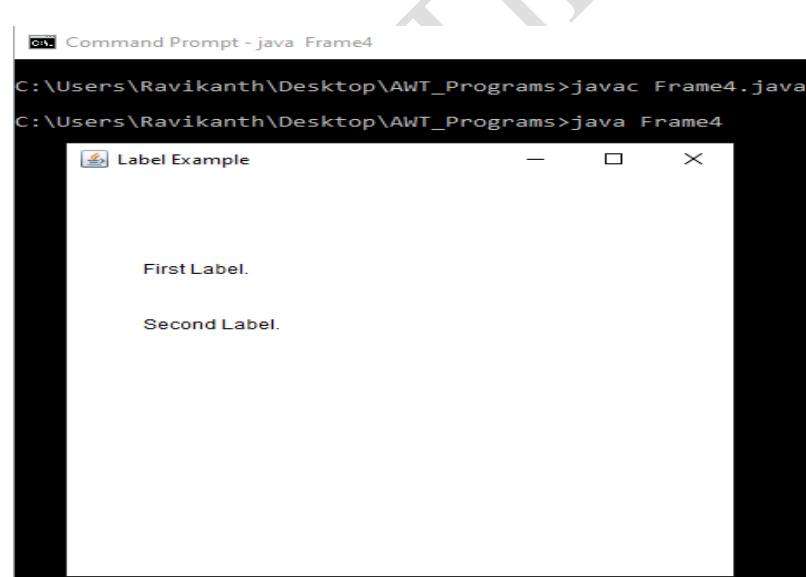
**Example 4:**

C:\Users\Ravikanth\Desktop\AWT\_Programs\Frame4.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run

Frame1.java Frame2.java Frame3.java Frame4.java

```
1 // TO add multiple Label to Frame
2 // Label Doesnot create any event
3 // Text cannot be changed by end user
4 import java.awt.*;
5 class Frame4
6 {
7     public static void main(String args[])
8     {
9         Frame f= new Frame("Label Example");
10        Label l1,l2;
11        l1=new Label("First Label.");
12        l1.setBounds(50,100, 100,30);
13        l2=new Label("Second Label.");
14        l2.setBounds(50,150, 100,30);
15        f.add(l1); f.add(l2);
16        f.setSize(400,400);
17        f.setLayout(null);
18        f.setVisible(true);
19    }
20 }
```



Command Prompt - java Frame4

C:\Users\Ravikanth\Desktop\AWT\_Programs>javac Frame4.java

C:\Users\Ravikanth\Desktop\AWT\_Programs>java Frame4

Label Example

First Label.

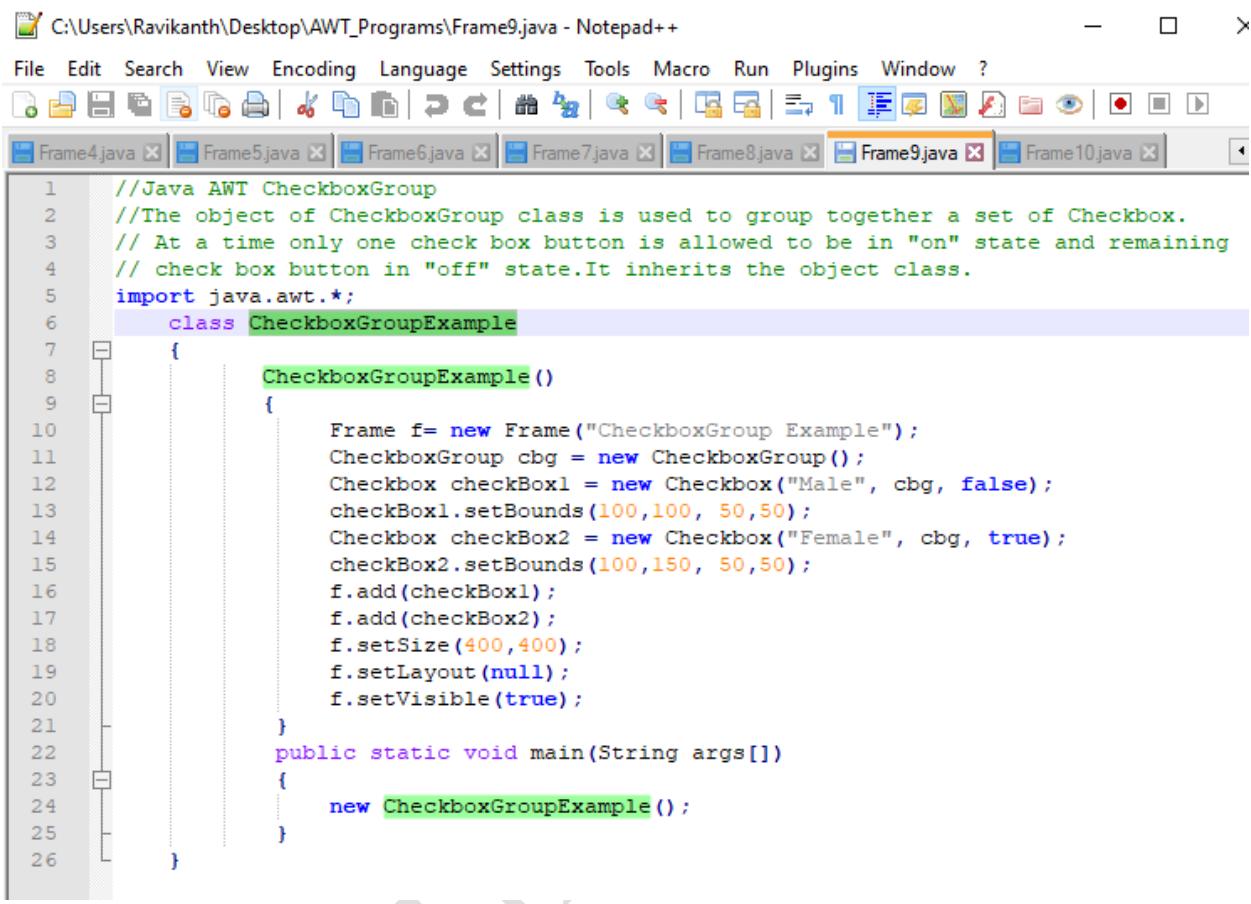
Second Label.

**Example 5:**

```
1 // To add two TextFields to Frame
2 // It allows user to edit single line of text
3 // Event is generated by user by typing in text field
4 import java.awt.*;
5 class Frame5
6 {
7     public static void main(String args[])
8     {
9         Frame f= new Frame("TextField Example");
10        TextField t1,t2;
11        t1=new TextField("Welcome to OOPS JAVA ECE E2");
12        t1.setBounds(50,100, 200,30);
13        t2=new TextField("Session over AWT GUI UNTI-5");
14        t2.setBounds(50,150, 200,30);
15        f.add(t1); f.add(t2);
16        f.setSize(400,400);
17        f.setLayout(null);
18        f.setVisible(true);
19    }
20 }
```

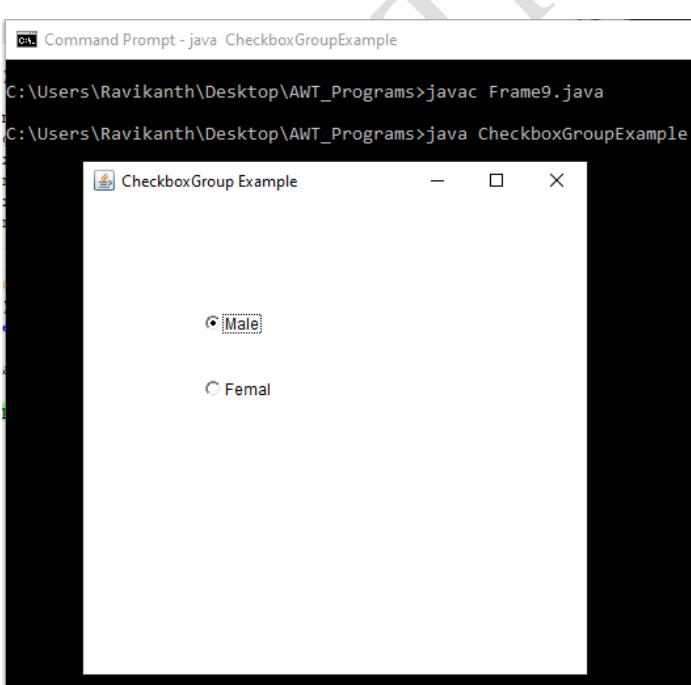


## Example 6:



```

1 //Java AWT CheckboxGroup
2 //The object of CheckboxGroup class is used to group together a set of Checkbox.
3 // At a time only one check box button is allowed to be in "on" state and remaining
4 // check box button in "off" state.It inherits the object class.
5 import java.awt.*;
6 class CheckboxGroupExample
7 {
8     CheckboxGroupExample()
9     {
10        Frame f= new Frame("CheckboxGroup Example");
11        CheckboxGroup cbg = new CheckboxGroup();
12        Checkbox checkBox1 = new Checkbox("Male", cbg, false);
13        checkBox1.setBounds(100,100, 50,50);
14        Checkbox checkBox2 = new Checkbox("Female", cbg, true);
15        checkBox2.setBounds(100,150, 50,50);
16        f.add(checkBox1);
17        f.add(checkBox2);
18        f.setSize(400,400);
19        f.setLayout(null);
20        f.setVisible(true);
21    }
22    public static void main(String args[])
23    {
24        new CheckboxGroupExample();
25    }
26 }
```



Command Prompt - java CheckboxGroupExample

```

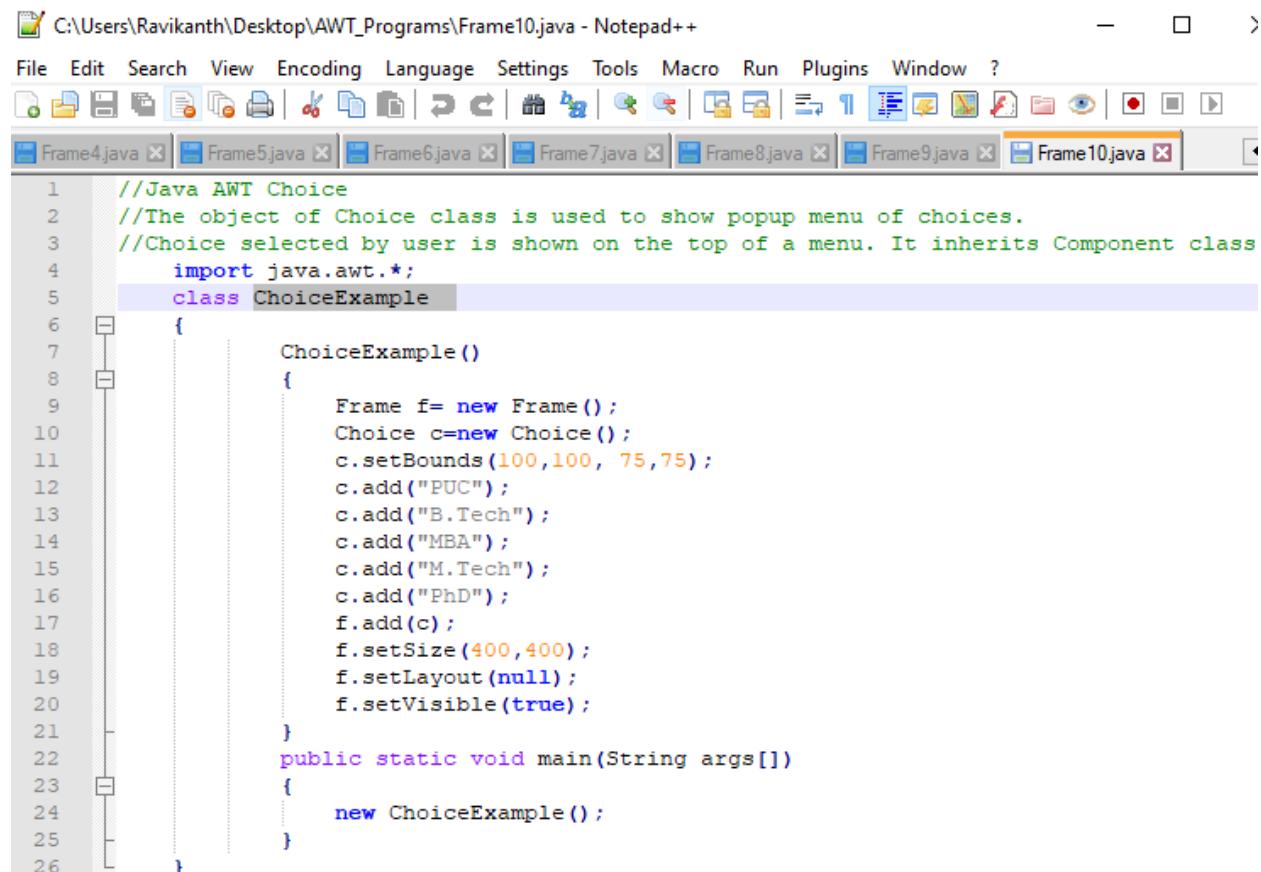
C:\Users\Ravikanth\Desktop\AWT_Programs>javac Frame9.java
C:\Users\Ravikanth\Desktop\AWT_Programs>java CheckboxGroupExample
```

Checkbox Group Example

Male

Female

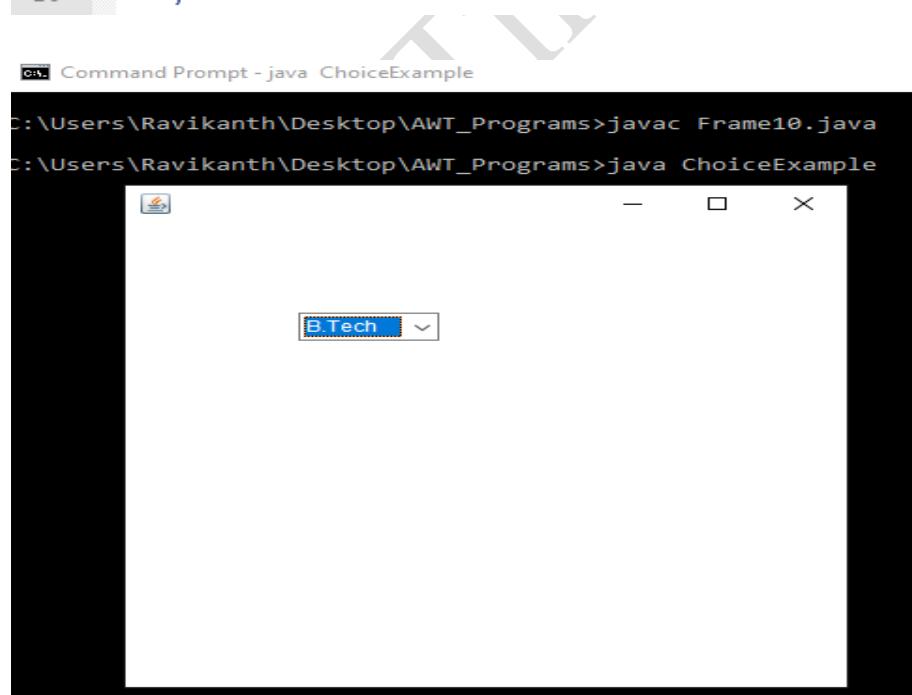
## Example 7:



```

C:\Users\Ravikanth\Desktop\AWT_Programs\Frame10.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Frame4.java Frame5.java Frame6.java Frame7.java Frame8.java Frame9.java Frame10.java
1 //Java AWT Choice
2 //The object of Choice class is used to show popup menu of choices.
3 //Choice selected by user is shown on the top of a menu. It inherits Component class
4 import java.awt.*;
5 class ChoiceExample
6 {
7     ChoiceExample()
8     {
9         Frame f= new Frame();
10        Choice c=new Choice();
11        c.setBounds(100,100, 75,75);
12        c.add("PUC");
13        c.add("B.Tech");
14        c.add("MBA");
15        c.add("M.Tech");
16        c.add("PhD");
17        f.add(c);
18        f.setSize(400,400);
19        f.setLayout(null);
20        f.setVisible(true);
21    }
22    public static void main(String args[])
23    {
24        new ChoiceExample();
25    }
26 }

```



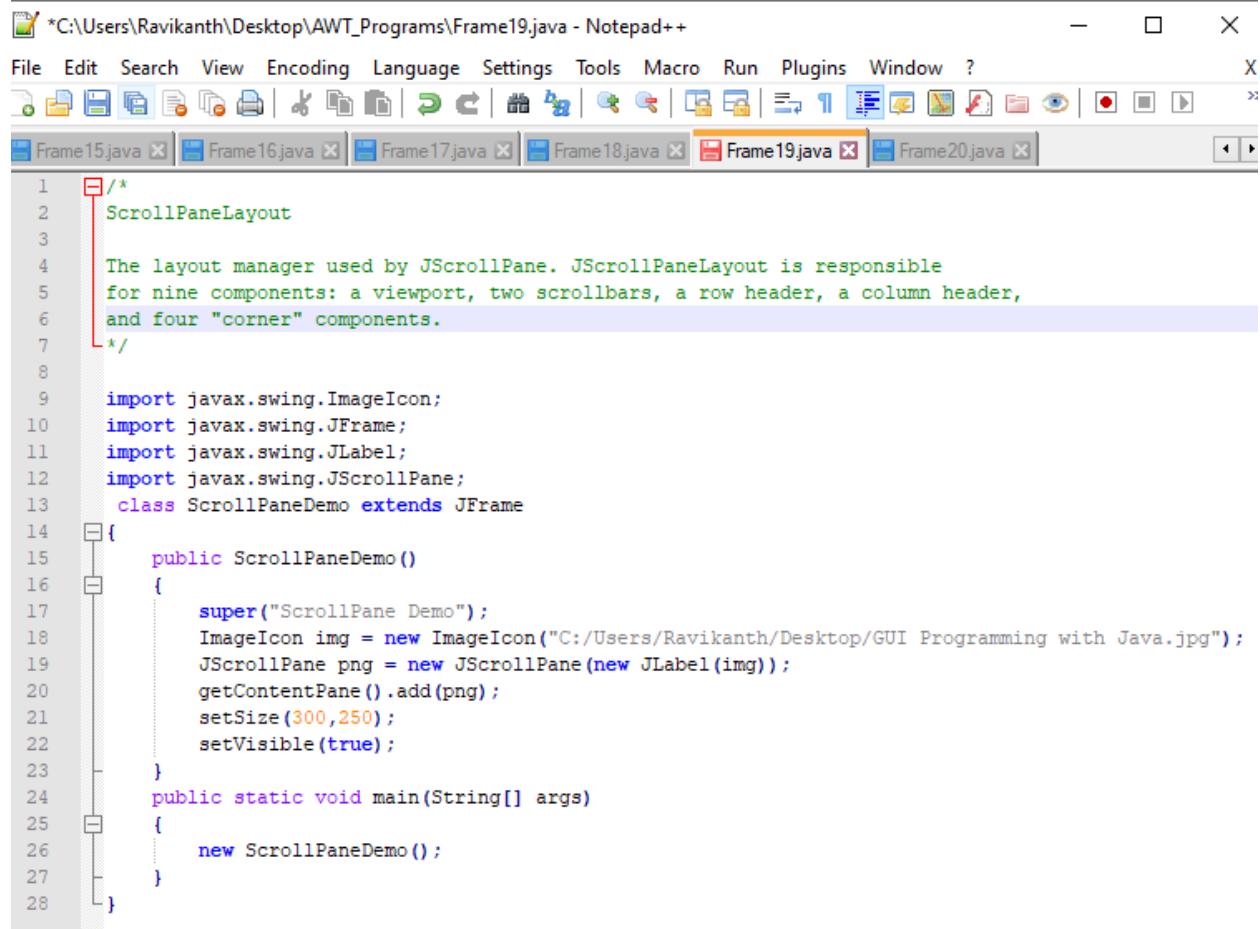
```

Command Prompt - java ChoiceExample
D:\Users\Ravikanth\Desktop\AWT_Programs>javac Frame10.java
D:\Users\Ravikanth\Desktop\AWT_Programs>java ChoiceExample

```

The screenshot shows a Windows Command Prompt window. The command `javac Frame10.java` is run, followed by `java ChoiceExample`. A Java application window titled "ChoiceExample" appears, displaying a dropdown menu with the items "B.Tech" and "M.TBA".

### Example 8:



```

1  /*
2   * ScrollPaneLayout
3   *
4   * The layout manager used by JScrollPane. JScrollPaneLayout is responsible
5   * for nine components: a viewport, two scrollbars, a row header, a column header,
6   * and four "corner" components.
7   */
8
9  import javax.swing.ImageIcon;
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import javax.swing.JScrollPane;
13 class ScrollPaneDemo extends JFrame
14 {
15     public ScrollPaneDemo()
16     {
17         super("ScrollPane Demo");
18         ImageIcon img = new ImageIcon("C:/Users/Ravikanth/Desktop/GUI Programming with Java.jpg");
19         JScrollPane png = new JScrollPane(new JLabel(img));
20         getContentPane().add(png);
21         setSize(300,250);
22         setVisible(true);
23     }
24     public static void main(String[] args)
25     {
26         new ScrollPaneDemo();
27     }
28 }

```

Command Prompt - java ScrollPaneDemo



```

C:\Users\Ravikanth\Desktop\AWT_Programs>javac Frame19.java
C:\Users\Ravikanth\Desktop\AWT_Programs>java ScrollPaneDemo

```

## Example 9:

```
//Java LocalDate Example
//Java LocalDate class is an immutable class that represents Date with a
//default format of yyyy-MM-dd.
// It inherits Object class and implements the ChronoLocalDate interface
import java.time.LocalDate;
class LocalDateExample
{
    public static void main(String[] args)
    {
        LocalDate date = LocalDate.now();
        LocalDate yesterday = date.minusDays(1);
        LocalDate tomorrow = yesterday.plusDays(2);
        System.out.println("Today date: "+date);
        System.out.println("Yesterday date: "+yesterday);
        System.out.println("Tommorow date: "+tomorrow);
    }
}
```

XX

```
Command Prompt
C:\Users\Ravikanth\Desktop>javac Frame20.java
C:\Users\Ravikanth\Desktop>java LocalDateExample
Today date: 2021-05-31
Yesterday date: 2021-05-30
Tommorow date: 2021-06-01
C:\Users\Ravikanth\Desktop>
```

## Example 10:

C:\Users\Ravikanth\Desktop\AWT\_Programs\Frame24.java - Notepad++

```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Frame20.java Frame21.java Frame22.java Frame23.java Frame24.java Frame25.java

1 import java.awt.*;
2 import java.applet.*;
3 import java.awt.event.*;
4 class student extends Frame implements ActionListener
5 {String msg;
6  Button bl=new Button("SUBMIT");
7  Label l1l=new Label("Student details",Label.CENTER);
8  Label l1=new Label("Name:",Label.LEFT);
9  Label l2=new Label("Age:",Label.LEFT);
10 Label l3=new Label("Gender(M/F):",Label.LEFT);
11 Label l4=new Label("Address:",Label.LEFT);
12 Label l5=new Label("Course:",Label.LEFT);
13 Label l6=new Label("Semester:",Label.LEFT);
14 Label l7=new Label("",Label.RIGHT);
15 TextField t1=new TextField();
16 Choice c1=new Choice();
17 CheckboxGroup cbg=new CheckboxGroup();
18 Checkbox ck1=new Checkbox("Male",false,cbg);
19 Checkbox ck2=new Checkbox("Female",false,cbg);
20 TextArea t2=new TextArea("",180,90,TextArea.SCROLLBARS_VERTICAL_ONLY);
21 Choice course=new Choice();
22 Choice sem=new Choice();
23 Choice age=new Choice();

```

C:\Users\Ravikanth\Desktop\AWT\_Programs\Frame24.java - Notepad++

```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Frame20.java Frame21.java Frame22.java Frame23.java Frame24.java Frame25.java

24 public student()
25 {
26     addWindowListener(new myWindowAdapter());
27     setBackground(Color.cyan);
28     setForeground(Color.black);
29     setLayout(null);
30     add(l1l);
31     add(l1);
32     add(l2);
33     add(l3);
34     add(l4);
35     add(l5);
36     add(l6);
37     add(l7);
38     add(t1);
39     add(t2);
40     add(ck1);
41     add(ck2);
42     add(course);
43     add(sem);
44     add(age);
45     add(bl);
46     bl.addActionListener(this);
47     add(bl);
48     course.add("RGUKT-PUC");
49     course.add("RGUKT-B.Tech");
50     course.add("RGUKT-M.Tech");
51     course.add("RGUKT-MBA");
52     course.add("RGUKT-PhD");

```

C:\Users\Ravikanth\Desktop\AWT\_Programs\Frame24.java - Notepad++

```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Frame20.java Frame21.java Frame22.java Frame23.java Frame24.java Frame25.java Frame26

53     sem.add("1");
54     sem.add("2");
55     sem.add("3");
56     sem.add("4");
57     sem.add("5");
58     sem.add("6");
59     age.add("17");
60     age.add("18");
61     age.add("19");
62     age.add("20");
63     age.add("21");
64     l1.setBounds(25,65,90,20);
65     l2.setBounds(25,90,90,20);
66     l3.setBounds(25,120,90,20);
67     l4.setBounds(25,185,90,20);
68     l5.setBounds(25,260,90,20);
69     l6.setBounds(25,290,90,20);
70     l7.setBounds(25,260,90,20);
71     l11.setBounds(10,40,280,20);
72     t1.setBounds(120,65,170,20);
73     t2.setBounds(120,185,170,60);
74     ck1.setBounds(120,120,50,20);
75     ck2.setBounds(170,120,60,20);
76     course.setBounds(120,260,100,20);
77     sem.setBounds(120,290,50,20);
78     age.setBounds(120,90,50,20);
79     b1.setBounds(120,350,50,30);
80 }
81 public void paint(Graphics g)
82 {g.drawString(msg,200,450);}
83 public void actionPerformed(ActionEvent ae)
84 {
85     if(ae.getActionCommand().equals("SUBMIT"))
86     {
87         msg="Student Details Saved!";
88         setForeground(Color.cyan);
89     }
90     stu.setSize(new Dimension(500,500));
91     stu.setTitle("RGUKT-Student Registration Page");
92     stu.setVisible(true);
93 }
94 }
95 class myWindowAdapter extends WindowAdapter
96 {
97     public void windowClosing(WindowEvent we)
98     {
99         System.exit(0);
100    }
101 }

```

RGUKT-Student Registration Page

Student details

Name: RAVIKANTH

Age: 21

Gender(M/F):  Male  Female

Address: RGUKT Basar Campus

Course: RGUKT-B.Tec

Semester: 1

**SUBMIT**

\*\*\*\*\*

## SYLLABUS:

### UNIT-5:

**GUI Programming with Java:** Introduction to Swing, limitations of AWT, Swing vs AWT, MVC architecture, Hierarchy for Swing components,

Containers - JFrame, JApplet, JDialog, JPanel.

#### Overview of some swing components

Jbutton, JLabel, JTextField, JTextArea, simple swingapplications.

#### GUI Programming with Java:

- ❖ Application program is of 2 types : CUI & GUI [ User Friendly]
- ❖ Java supports GUI application in two ways:
  1. Abstract window ToolKit
  2. Swings [ New Technology]
- ❖ To develop GUI application we require 3 essential things
  1. Container –GUI Controls like Title Bar, Tool Box, Check Box
  2. Components
  3. Event Handler
- ❖ Swings is used to develop rich user interface application/Component which includes [ 3D ( RGB &  $\alpha$  [ Sharpening, Contract, Brightness ] ) ]
  - ❖ AWT components are 2 Dimensional-RGB [ General components + 256 colors]
  - ❖ Swing has own GUI component support where as AWT has system platform GUI support

**Introduction to Swing:**

- Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JFrame, JButton, JLabel, JTextField, JTextArea, JRadioButton, JCheckBox, JMenu, JColorChooser, JComboBox, JPasswordField, JOptionPane etc.

**Classes of AWT and the Corresponding Swing**

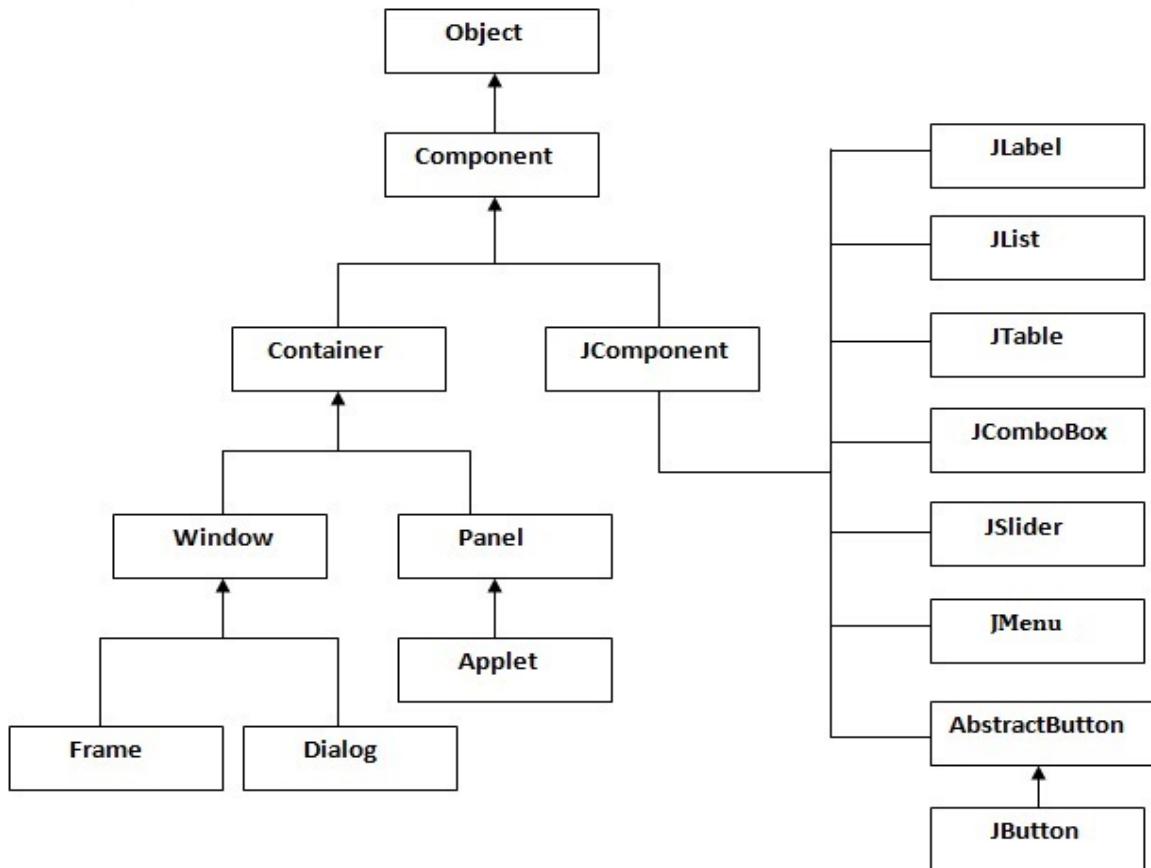
| <b>Classes of AWT</b> | <b>Corresponding Classes of Swing</b> |
|-----------------------|---------------------------------------|
| Frame                 | JFrame                                |
| Applet                | JApplet                               |
| Panel                 | JPanel                                |
| Label                 | JLabel                                |
| Button                | JButton                               |
| TextField             | JTextField                            |
| CheckBox              | JCheckBox                             |
| List                  | JList                                 |
| Menu                  | JMenu                                 |
| MenuBar               | JMenuBar                              |
| MenuItem              | JMenuItem                             |
| Choice                | JComboBox                             |
| TextArea              | JTextArea                             |
| .....                 | .....                                 |

**Difference between AWT and Swing:**

| <b>Java AWT</b>   | <b>Java Swing</b>  |
|---|--|
| AWT components are platform-dependent.                                      | Java swing components are platform-independent.  |
| AWT components are heavyweight.   | Swing components are lightweight.  |
| AWT doesn't support pluggable look and feel.                                | Swing supports pluggable look and feel.  |
| AWT provides less components than Swing.                                    | Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, etc. |
| AWT doesn't follows MVC   | Swing follows MVC.   |
| Import java.awt.*;  | Import javax.swing.*;  |
| Both are used to design GUI   | Both are used to design GUI  |
| Look and feel standard are bit low<br>256 colors                            | HIGH RGB +Alpha [ Contrast , Brightness, ]   |
| Default Layout manager for applet:<br>FlowLayout and Frame is Border Layout | The default Layout manager for content pane is Border Layout                                   |
| Doesnot support features like icons and tool-tips                           | Supports features like icon and tool-tips  |
| Not pure Java-based   | Pure Java-based  |

**Hierarchy of Java Swing classes**

The hierarchy of java swing API is given below.



### Features of Swing:

1. Swing components are light weight
2. It has lot of built-in controls- tabbed panes, sliders, toolbars, tables....etc
3. We can customize our GUI Appl.
4. Very much attractive – Pluggable look and feel features
5. Platform independent
6. All components are named after JApplet, JButton....
7. After java 5 lot of components got added up ( Component c)
8. All drawing is done using paintComponent rather than paint

### Steps to write a Swing GUI Application:

- **Step 1: Define Container class [By extending Frame class (inheritance)]**
- **Step 2: Define Control Components**
- **Step 3: Set Layout to the container**
- **Step 4: Add the components to the container**
- **Step 5: Register the components to the Listener Class**
  - A) Define Listener Implemented class
  - B) Register the source using addActionListener(Listener Object)
  - C) Override ListenerInterface abstract method for Business logic
- **Step 6: Define container Visual Properties**
- **Step 7: Define main class and create object**
- **Step 8: Compile and Execute**

**Javax.swing**

**Javax.swing.event**

**Java JButton**

- The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

**JButton class declaration**

Let's see the declaration for javax.swing.JButton class.

- public class JButton extends AbstractButton implements Accessible

**Commonly used Constructors:**

| Constructor       | Description   |
|-------------------|---|
| JButton()         | It creates a button with no text and icon.          |
| JButton(String s) | It creates a button with the specified text.        |
| JButton(Icon i)   | It creates a button with the specified icon object. |

**Commonly used Methods of Abstract Button class:**

| Methods                                  | Description   |
|--|---|
| void setText(String s)                   | It is used to set specified text on button            |
| String getText()                         | It is used to return the text of the button.          |
| void setEnabled(boolean b)               | It is used to enable or disable the button.           |
| void setIcon(Icon b)                     | It is used to set the specified Icon on the button.   |
| Icon getIcon()                           | It is used to get the Icon of the button.             |
| void setMnemonic(int a)                  | It is used to set the mnemonic on the button.         |
| void addActionListener(ActionListener a) | It is used to add the action listener to this object. |

**Java JButton Example**

```
import javax.swing.*;
public class ButtonExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
```

```
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

### Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Button Example");
        final JTextField tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        b.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                tf.setText("Welcome to Javatpoint.");
            }
        });
        f.add(b);f.add(tf);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

### Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample{
ButtonExample(){
JFrame f=new JFrame("Button Example");
JButton b=new JButton(new ImageIcon("D:\\icon.png"));
b.setBounds(100,100,100, 40);
f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args) {
    new ButtonExample();
}
}
```

### Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

#### JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. public class JLabel extends JComponent implements SwingConstants, Accessible

#### Commonly used Constructors:

| Constructor                                       | Description   |
|---|---|
| JLabel()  | Creates a JLabel instance with no image and with an empty string for the title.     |
| JLabel(String s)                                  | Creates a JLabel instance with the specified text.                                  |
| JLabel(Icon i)                                    | Creates a JLabel instance with the specified image.                                 |
| JLabel(String s, Icon i, int horizontalAlignment) | Creates a JLabel instance with the specified text, image, and horizontal alignment. |

#### Commonly used Methods:

| Methods                                    | Description  |
|--|--|
| String getText()                           | It returns the text string that a label displays.                  |
| void setText(String text)                  | It defines the single line of text this component will display.    |
| void setHorizontalAlignment(int alignment) | It sets the alignment of the label's contents along the X axis.    |
| Icon getIcon()                             | It returns the graphic image that the label displays.              |
| int getHorizontalAlignment()               | It returns the alignment of the label's contents along the X axis. |

#### Java JLabel Example

```
import javax.swing.*;
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
```

```

JLabel l1,l2;
l1=new JLabel("First Label.");
l1.setBounds(50,50, 100,30);
l2=new JLabel("Second Label.");
l2.setBounds(50,100, 100,30);
f.add(l1); f.add(l2); f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
} }

```

### Java JLabel Example with ActionListener

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
    JTextField tf; JLabel l; JButton b;
    LabelExample(){
        tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        l=new JLabel();
        l.setBounds(50,100, 250,20);
        b=new JButton("Find IP");
        b.setBounds(50,150,95,30);
        b.addActionListener(this);
        add(b);add(tf);add(l);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        try{
            String host=tf.getText();
            String ip=java.net.InetAddress.getByName(host).getHostAddress();
            l.setText("IP of "+host+" is: "+ip);
        }catch(Exception ex){System.out.println(ex);}
    }
    public static void main(String[] args) {
        new LabelExample();
    } }

```

### Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

#### JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

1. public class JTextField extends JTextComponent implements SwingConstants

#### Commonly used Constructors:

| Constructor                          | Description  |
|--------------------------------------|--|
| JTextField()                         | Creates a new TextField  |
| JTextField(String text)              | Creates a new TextField initialized with the specified text.             |
| JTextField(String text, int columns) | Creates a new TextField initialized with the specified text and columns. |
| JTextField(int columns)              | Creates a new empty TextField with the specified number of columns.      |

#### Commonly used Methods:

| Methods                                     | Description   |
|---|---|
| void addActionListener(ActionListener l)    | It is used to add the specified action listener to receive action events from this textfield.                       |
| Action getAction()                          | It returns the currently set Action for this ActionEvent source, or null if no Action is set.                       |
| void setFont(Font f)                        | It is used to set the current font.   |
| void removeActionListener(ActionListener l) | It is used to remove the specified action listener so that it no longer receives action events from this textfield. |

#### Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
```

```
JFrame f= new JFrame("TextField Example");
JTextField t1,t2;
t1=new JTextField("Welcome to Javatpoint.");
t1.setBounds(50,100, 200,30);
t2=new JTextField("AWT Tutorial");
t2.setBounds(50,150, 200,30);
f.add(t1); f.add(t2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

### Java JTextField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener{
    JTextField tf1,tf2,tf3;
    JButton b1,b2;
    TextFieldExample(){
        JFrame f= new JFrame();
        tf1=new JTextField();
        tf1.setBounds(50,50,150,20);
        tf2=new JTextField();
        tf2.setBounds(50,100,150,20);
        tf3=new JTextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new JButton("+");
        b1.setBounds(50,200,50,50);
        b2=new JButton("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

```

public void actionPerformed(ActionEvent e) {
    String s1=tf1.getText();
    String s2=tf2.getText();
    int a=Integer.parseInt(s1);
    int b=Integer.parseInt(s2);
    int c=0;
    if(e.getSource()==b1){
        c=a+b;
    }else if(e.getSource()==b2){
        c=a-b;
    }
    String result=String.valueOf(c);
    tf3.setText(result);
}
public static void main(String[] args) {
    new TextFieldExample();
}
}

```

### Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

#### JTextArea class declaration

Let's see the declaration for javax.swing.JTextArea class.

1. public class JTextArea extends JTextComponent

#### Commonly used Constructors:

| Constructor                              | Description  |
|--|--|
| JTextArea()                              | Creates a text area that displays no text initially.   |
| JTextArea(String s)                      | Creates a text area that displays specified text initially.  |
| JTextArea(int row, int column)           | Creates a text area with the specified number of rows and columns that displays no text initially. |
| JTextArea(String s, int row, int column) | Creates a text area with the specified number of rows and columns that displays specified text.    |

**Commonly used Methods:**

| Methods                             | Description  |
|-------------------------------------|--|
| void setRows(int rows)              | It is used to set specified number of rows.                        |
| void setColumns(int cols)           | It is used to set specified number of columns.                     |
| void setFont(Font f)                | It is used to set the specified font.                              |
| void insert(String s, int position) | It is used to insert the specified text on the specified position. |
| void append(String s)               | It is used to append the given text to the end of the document.    |

**Java JTextArea Example**

```
import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample(){
        JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}
```

**Java JTextArea Example with ActionListener**

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
JLabel l1,l2;
JTextArea area;
JButton b;
TextAreaExample() {
    JFrame f= new JFrame();
    l1=new JLabel();
    l1.setBounds(50,25,100,30);
```

```
l2=new JLabel();
l2.setBounds(160,25,100,30);
area=new JTextArea();
area.setBounds(20,75,250,200);
b=new JButton("Count Words");
b.setBounds(100,300,120,30);
b.addActionListener(this);
f.add(l1);f.add(l2);f.add(area);f.add(b);
f.setSize(450,450);
f.setLayout(null);
f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
    String text=area.getText();
    String words[]={};text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args) {
    new TextAreaExample();
}
}
```

\*\*\*\*\*

### Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

#### Nested Class

| Modifier and Type                    | Class                   | Description   |
|--------------------------------------|-------------------------|---|
| protected class<br>AccessibleContext | JFrame.AccessibleJFrame | This class implements accessibility support for the JFrame class. |

#### Fields

| Modifier and Type              | Field                   | Description  |
|--------------------------------|-------------------------|--|
| protected<br>AccessibleContext | accessibleContext       | The accessible context property.   |
| static int                     | EXIT_ON_CLOSE           | The exit application default window close operation.   |
| protected JRootPane            | rootPane                | The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane. |
| protected boolean              | rootPaneCheckingEnabled | If true then calls to add and setLayout will be forwarded to the contentPane.                                      |

#### Constructors

| Constructor                                       | Description  |
|---|--|
| JFrame()  | It constructs a new frame that is initially invisible.   |
| JFrame(GraphicsConfiguration gc)                  | It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.          |
| JFrame(String title)                              | It creates a new, initially invisible Frame with the specified title.                                    |
| JFrame(String title,<br>GraphicsConfiguration gc) | It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device. |

**Useful Methods**

| <b>Modifier and Type</b> | <b>Method</b>   | <b>Description</b>  |
|--------------------------|---|---|
| protected void           | addImpl(Component comp, Object constraints, int index)              | Adds the specified child Component.   |
| protected JRootPane      | createRootPane()  | Called by the constructor methods to create the default rootPane.   |
| protected void           | frameInit()   | Called by the constructors to init the JFrame properly.   |
| void                     | setContentPane(Container contentPane)                               | It sets the contentPane property  |
| static void              | setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated) | Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel. |
| void                     | setIconImage(Image image)   | It sets the image to be displayed as the icon for this window.  |
| void                     | setJMenuBar(JMenuBar menubar)                                       | It sets the menubar for this frame.   |
| void                     | setLayeredPane(JLayeredPane layeredPane)                            | It sets the layeredPane property.   |
| JRootPane                | getRootPane()   | It returns the rootPane object for this frame.  |
| TransferHandler          | getTransferHandler()  | It gets the transferHandler property.   |

**JFrame Example**

```

import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
    }
}

```

```

JLabel label = new JLabel("JFrame By Example");
JButton button = new JButton();
button.setText("Button");
panel.add(label);
panel.add(button);
frame.add(panel);
frame.setSize(200, 300);
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
}

```

### **JApplet class in Applet**

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing.  
The JApplet class extends the Applet class.

#### **Example of EventHandling in JApplet:**

```

import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener{
JButton b;
JTextField tf;
public void init(){
tf=new JTextField();
tf.setBounds(30,40,150,20);
b=new JButton("Click");
b.setBounds(80,150,70,40);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
}

```

In the above example, we have created all the controls in init() method because it is invoked only once.

### **myapplet.html**

1. <html>
2. <body>
3. <applet code="EventJApplet.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

### **Java JDialog**

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.

Unlike JFrame, it doesn't have maximize and minimize buttons.

### **JDialog class declaration**

Let's see the declaration for javax.swing.JDialog class.

1. public class JDialog extends Dialog implements WindowConstants, Accessible, RootPaneContainer

### **Commonly used Constructors:**

| <b>Constructor</b>                                | <b>Description</b>   |
|---|--|
| JDialog()   | It is used to create a modeless dialog without a title and without a specified Frame owner.  |
| JDialog(Frame owner)                              | It is used to create a modeless dialog with specified Frame as its owner and an empty title. |
| JDialog(Frame owner, String title, boolean modal) | It is used to create a dialog with the specified title, owner Frame and modality.            |

### **Java JDialog Example**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static JDialog d;
    DialogExample() {
```

```

JFrame f= new JFrame();
d = new JDialog(f , "Dialog Example", true);
d.setLayout( new FlowLayout() );
JButton b = new JButton ("OK");
b.addActionListener ( new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
        DialogExample.d.setVisible(false);
    }
});
d.add( new JLabel ("Click button to continue."));
d.add(b);   d.setSize(300,300);
d.setVisible(true);
}
public static void main(String args[])
{
    new DialogExample();
}
}

```

### **Java JPanel**

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

#### **JPanel class declaration**

1. public class JPanel extends JComponent implements Accessible

#### **Commonly used Constructors:**

| <b>Constructor</b>               | <b>Description</b>  |
|----------------------------------|---|
| JPanel()                         | It is used to create a new JPanel with a double buffer and a flow layout.               |
| JPanel(boolean isDoubleBuffered) | It is used to create a new JPanel with FlowLayout and the specified buffering strategy. |
| JPanel(LayoutManager layout)     | It is used to create a new JPanel with the specified layout manager.                    |

**Java JPanel Example**

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
    {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new PanelExample();
    }
}
```

**Execute and Run all the Programs and Check the Results**

---

# **UNIT-V OOPS**

## **GUI Programming with Java**

# Graphics

- **Graphics** object draws pixels on the screen that represent text and other graphical shapes such as lines, ovals, rectangles, polygons etc.
- The graphics class is an **abstract** class(i.e. Graphics objects can not be instantiated.)
- A graphics context is encapsulated by the Graphics class and is obtained in two ways:
  - It is passed to an applet when one of its various methods, such as `paint()` or `update()`, is called.
  - It is returned by the **getGraphics()** method of Component.

## **public void paint(Graphics g)**

- The **paint(Graphics g)** method is common to all components and containers.
- It is needed if you do any drawing or painting other than just using standard GUI components.
- Never call **paint(Graphics g)**, call **repaint()**

## **How does the paint(Graphics g) method get called?**

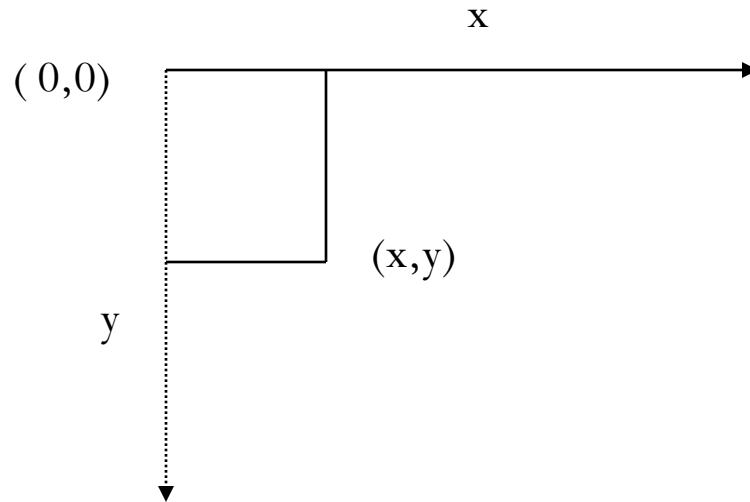
- It is called automatically by Java whenever the component or container is loaded, resized, minimized, maximized.
- You can cause the paint method to be called at any time by calling the component's **repaint()** method.

```
class myFrame extends Frame{  
    .....  
    .....  
    public void paint(Graphics g){  
        g.drawOval(x1,y1,width,height);  
        .....  
    }  
}
```

# The **repaint()** method will do two things:

1. It calls update(Graphics g), which writes over the old drawing in background color (thus erasing it).
2. It then calls paint(Graphics g) to do the drawing.

## Java coordinate system



# Graphics methods for drawing shapes

```
g.drawString (str, x, y);           //Puts string at x,y
```

```
g.drawLine( x1, y1, x2, y2 )      //Line from x1, y1 to x2, y2
```

```
g.drawRect( x1, y1, width, height) //Draws rectangle with upper left corner x1, y1
```

```
g.fillRect(x1, y1, width, height)   //Draws a solid rectangle.
```

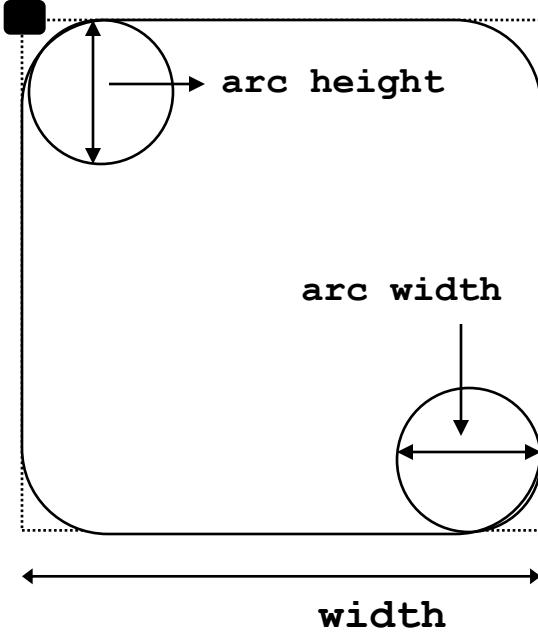
  

```
g.drawRoundRect( x, y, width, height,arcWidth, arcHeight )  
//Draws rectangle with rounded corners.
```

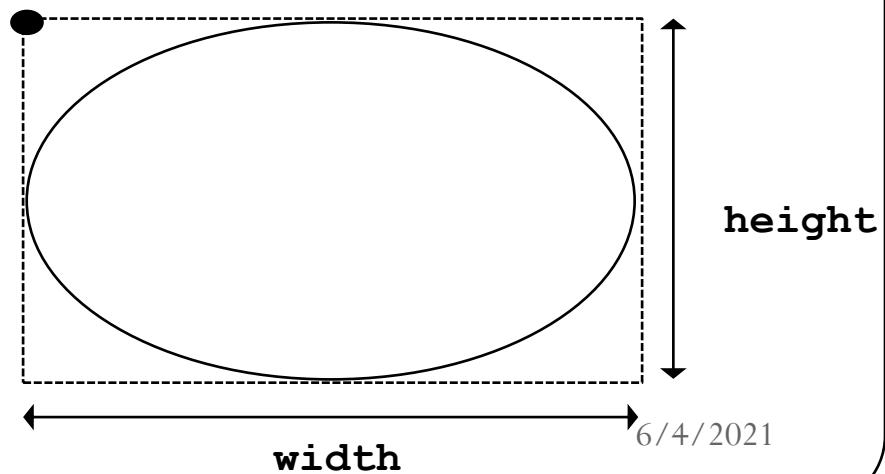
```
g.fillRoundRect( x, y, width, height,arcWidth, arcHeight )  
//Draws a solid rectangle with rounded corners.
```

**(x, y)**



**drawRoundRect**  
parameters

**(x, y)**



**drawOval** parameters

**g.drawOval(x1, y1, width, height)**

//Draws an oval with specified width and height. The bounding rectangle's top left corner is at the coordinate (x,y).The oval touches all four sides all four sides of the bounding rectangle.

**g.fillOval(x1, y1, width, height)**

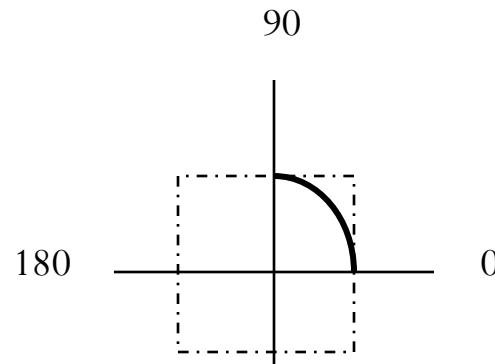
//Draws a filled oval.

**g.setColor(Color.RED)**

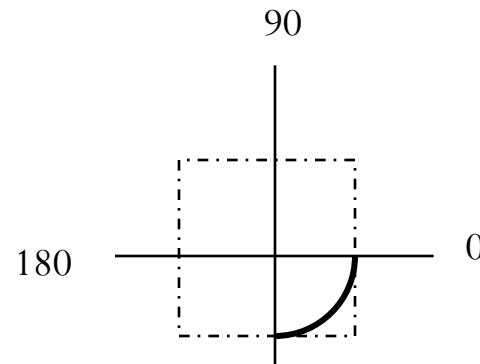
//Sets color, it is remain active until new color is set.

`g.drawArc(x1, y1, width, height, startAngle,arcAngle)`

//Draws an arc relative to the bounding rectangle's top-left coordinates with the specified width and height. The arc segment is drawn starting at startAngle and sweeps arcAngle degrees.



Positive angle



Negative angle

# Drawing Polygons and Polylines

- Polygon - multisided shape
- Polyline - series of connected points
- Methods of class Polygon

## **drawPolygon( xPoints[], yPoints[], points )**

//Draws a polygon, with x and y points specified in arrays. Last argument specifies number of points  
//Closed polygon, even if last point different from first.

## **drawPolyline ( xPoints[], yPoints[],points )**

//Draws a polyline

# Font

- Fonts are encapsulated by the **Font class**.
- To select a new font, you must first construct a **Font object** that describes that font.

**Font(String fontName, int fontStyle, int pointSize)**

//Here, fontName specifies the name of the desired font.

- All Java environments will support the following fonts:  
**Dialog, DialogInput, Sans Serif, Serif, Monospaced and Symbol.**
- The style of the font is specified by fontStyle. It may consist of one or more of these three constants:

**Font.PLAIN**

**Font.BOLD**

**Font.ITALIC**

**void setFont(Font fontObj)**

//To set the specified Font .

//It is defined in **Component** class.

//Here, fontObj is the object that contains the desired font.

**Font getFont()**

//To obtain information about the currently selected font which is defined in **Graphics** class

## Methods in Font Class:

**String getName()**

//To get the name of the Font

**String getFamily();**

//To get family name of the Font

**int getSize();**

//To get the size of the Font

**int getStyle();**

//To get the style of the Font

# Layout Managers

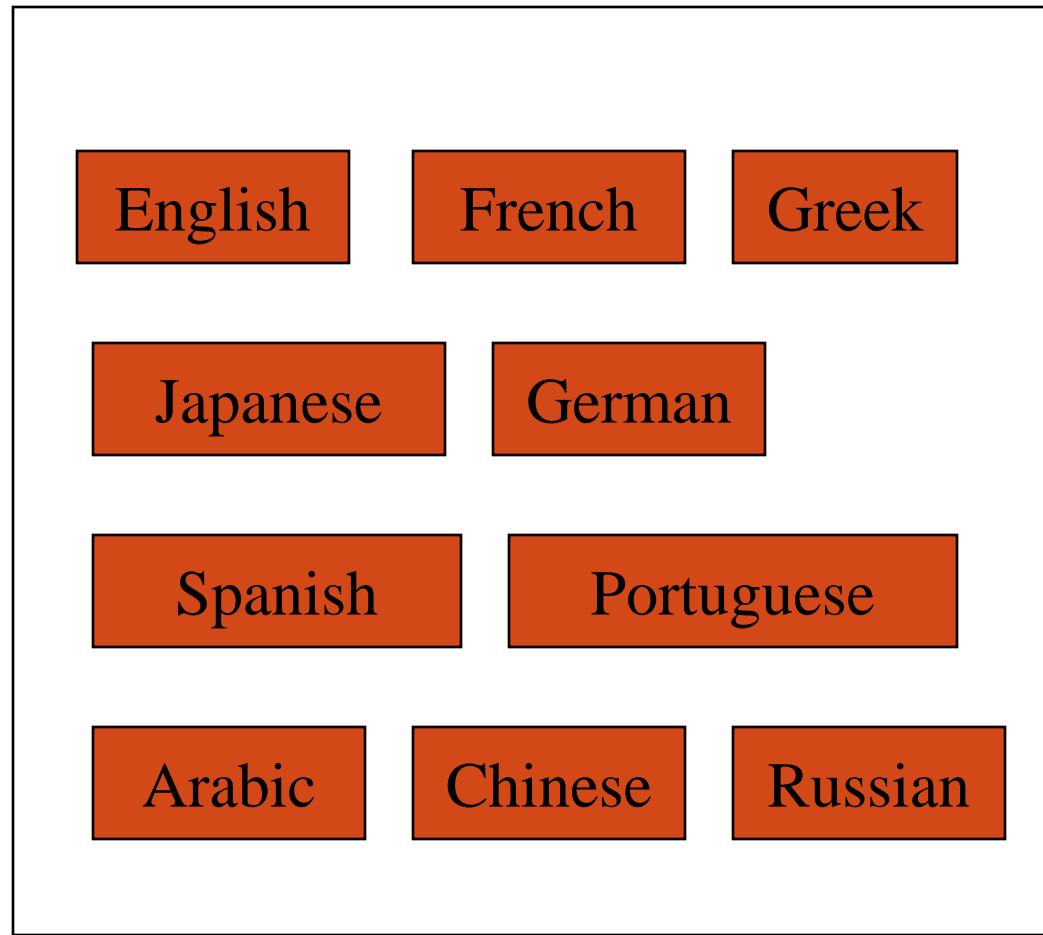
- Arranges and lays out the GUI components on a container.
  - Every container has a default Layout Manager:
    - Panel FlowLayout
    - Window (Frame) BorderLayout
  - Types:

|             |                |             |
|-------------|----------------|-------------|
| Flow Layout | Border Layout  | Grid Layout |
| Card Layout | Gridbag Layout |             |
  - To set layout manager:  
**myContainer.setLayout( new LayoutManger() );**

# Flow Layout

- The Flow Layout manager arranges the components left-to-right, top-to-bottom in the order they were inserted into the container.
- When the container is not wide enough to display all the components, the remaining components are placed in the next row, etc.
- By default each row is centered.
- The line alignment can be:
  - **FlowLayout.LEFT**
  - **FlowLayout.CENTER**
  - **FlowLayout.RIGHT**

# Flow Layout Example



# Flow Layout Constructors

## ➤ **FlowLayout()**

//A centered alignment and a default 5-unit horizontal and vertical gap.

## ➤ **FlowLayout(align)**

align – alignment used by the manager. A default 5-unit horizontal and vertical gap.

## ➤ **FlowLayout(align, hgap, vgap)**

//align – alignment used by the manager

//hgap – horizontal gaps between components

//vgap – vertical gaps between components

# Border Layout

- The Border Layout manager arranges components into five regions: **North**, **South**, **East**, **West**, and **Center**.
- Components in the North and South are set to their natural heights and horizontally stretched to fill the entire width of the container.
- Components in the East and West are set to their natural widths and stretched vertically to fill the entire width of the container.
- The Center component fills the space left in the center of the container.
- If one or more of the components, except the Center component, are missing then the rest of the existing components are stretched to fill the remaining space in the container.

➤ The positional constraints are:

**BorderLayout.NORTH**

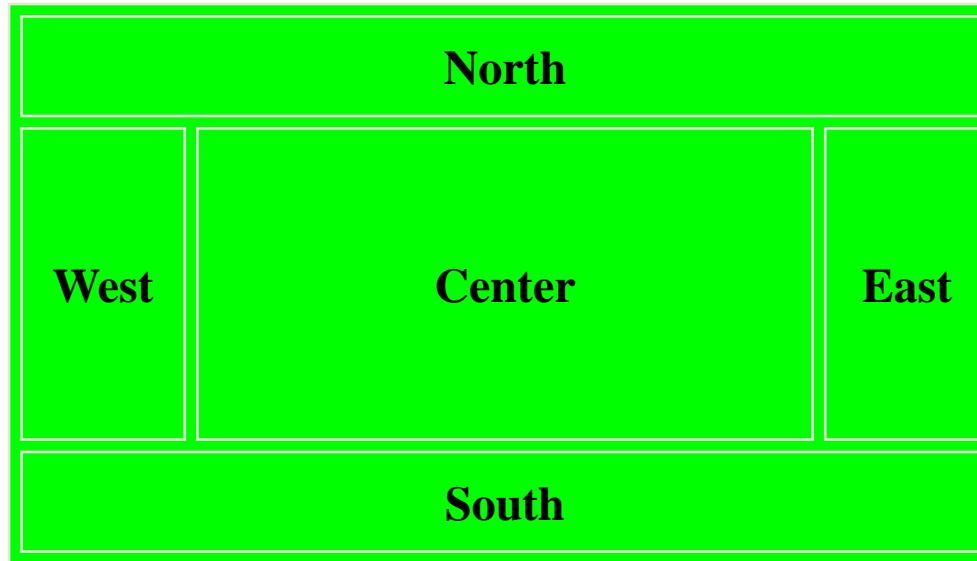
**BorderLayout.EAST**

**BorderLayout.CENTER**

**BorderLayout.SOUTH**

**BorderLayout.WEST**

# BorderLayout Manager



**To set BorderLayout:**

```
add( new Button("ok"), BorderLayout.NORTH);
```

# Border Layout Constructors

- **BorderLayout()**

//No vertical or horizontal gaps.

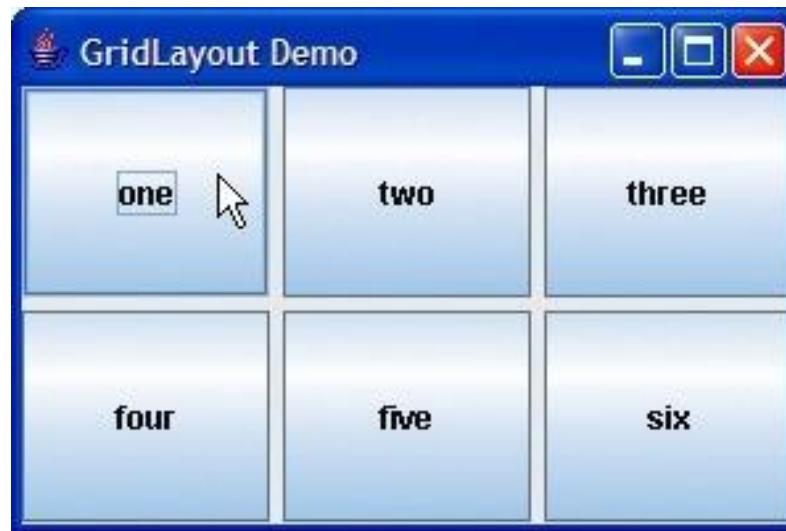
- **BorderLayout(hgap, vgap)**

//hgap – horizontal gaps between components

//vgap – vertical gaps between components

# Grid Layout

- Container is divided into a grid where components are placed in rows and columns.
  - Every component has the same width and height.
- Example



# Grid Layout Constructors

## ➤ **GridLayout()**

//A single row and no vertical or horizontal gaps.

## ➤ **GridLayout(r, c)**

//r – number of rows in the layout

//c – number of columns in the layout

//No vertical or horizontal gaps.

## ➤ **GridLayout(r, c, hgap, vgap)**

//r – number of rows in the layout

//c – number of columns in the layout

//hgap – horizontal gaps between components

//vgap – vertical gaps between components

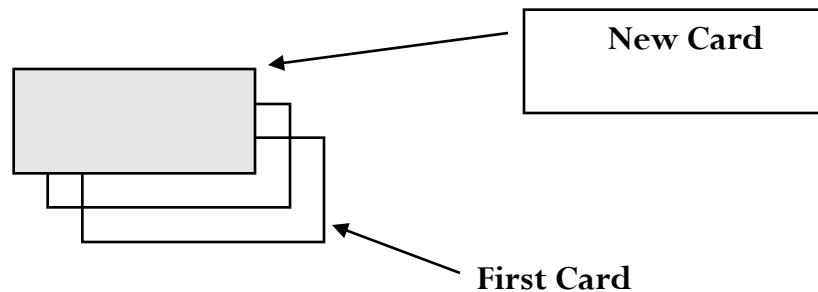
# CardLayout

- CardLayout places components (usually panels) on top of each other in a stack like a deck of cards.
- You can see only one card at a time.
- By default, the first card is visible.
- We can put a cards on top using another control by using the methods next(), previous(), first(), last(), and show().

**Constructor:** CardLayout()

**Methods:**

```
public void first(Container parent);  
public void next(Container parent);  
public void previous(Container parent);  
public void last(Container parent);  
public void show(Container parent, String name);
```



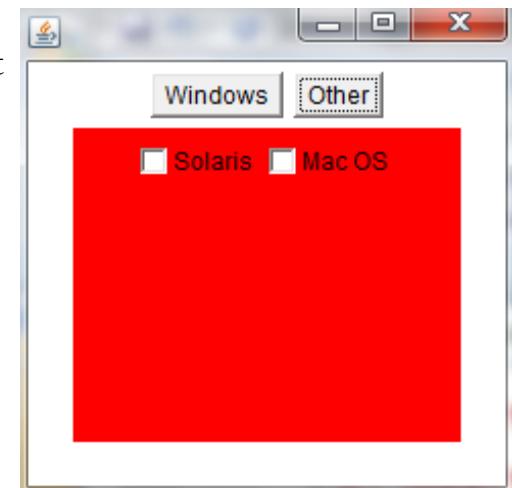
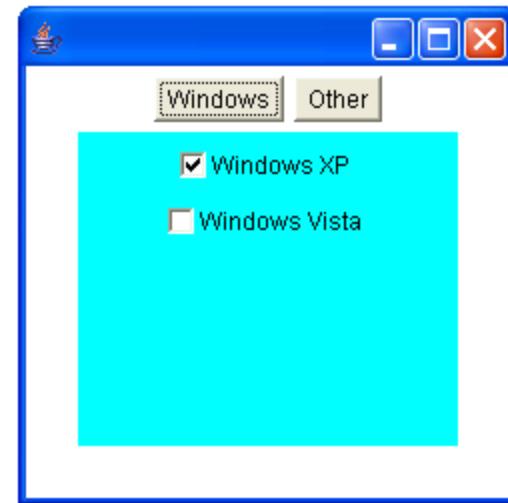
**add(component, name)**

Mr.K.RaviKanth, Asst.Prof.,IIIT-RGUKT,Basar

```

import java.awt.*; import java.awt.event.*;
public class CardLayoutDemo extends Frame implements ActionListener, MouseListener {
    Checkbox winXP, winVista, solaris, mac;
    Button Win, Other;
    Panel osCards;
    CardLayout cardLO;
    public CardLayoutDemo(){
        setLayout(new BorderLayout());
        cardLO = new CardLayout();
        Panel tabs=new Panel();
        Win = new Button("Windows");
        Other = new Button("Other");
        tabs.add(Win);
        tabs.add(Other);
        add(tabs,BorderLayout.NORTH);
        osCards = new Panel();
        osCards.setLayout(cardLO); // set panel layout to card layout
        Panel winPan = new Panel();
        winXP = new Checkbox("Windows XP", null, true);
        winVista = new Checkbox("Windows Vista");
        // add Windows check boxes to a panel
        winPan.setBackground(Color.CYAN);
        winPan.add(winXP);
        winPan.add(winVista);

```



```
// Add other OS check boxes to a panel
Panel otherPan = new Panel();
solaris = new Checkbox("Solaris");
mac = new Checkbox("Mac OS");
otherPan.setBackground(Color.RED);
otherPan.add(solaris);
otherPan.add(mac);

// add panels to card deck panel
osCards.add(winPan, "Windows");
osCards.add(otherPan, "Other");
// add cards to main applet panel
add(osCards,BorderLayout.CENTER);
// register to receive action events

Win.addActionListener(this);
Other.addActionListener(this);

// register mouse events
addMouseListener(this);
setSize(250,250);
setVisible(true);
}
```

```
public Insets getInsets(){
    return new Insets(30,30,30,30);
}
public void mousePressed(MouseEvent me) {
    cardLO.next(osCards);
}
public void mouseClicked(MouseEvent me) { }
public void mouseEntered(MouseEvent me) { }
public void mouseExited(MouseEvent me) { }
public void mouseReleased(MouseEvent me) { }
public void actionPerformed(ActionEvent ae) {
    if(ae.getSource() == Win) {
        cardLO.first(osCards); //cardLO.show(osCards, "Windows");
    }
    else {
        cardLO.last(osCards); //cardLO.show(osCards, "Other");
    }
}
public static void main(String arg[]){
    new CardLayoutDemo();
}
}
```

# GridBagLayout Layout Manager

- Flexible GridBagLayout
  - Components can vary in size
  - Components can occupy multiple rows and columns
  - Components can be added in any order
- There are two classes that are used:
  - GridBagLayout**: Provides the overall layout manager.
  - GridBagConstraints**: Defines the properties of each component in the grid such as placement, dimension and alignment

## Specifying a GridBagLayout

- To use a gridbag layout, you must declare the **GridBagLayout** and **GridBagConstaraints** object and attach the layout to your applet.

### Example:

```
GridBagLayout gridbag = new GridBagLayout();
```

```
GridBagConstraints constraints = new GridBagConstraints();
```

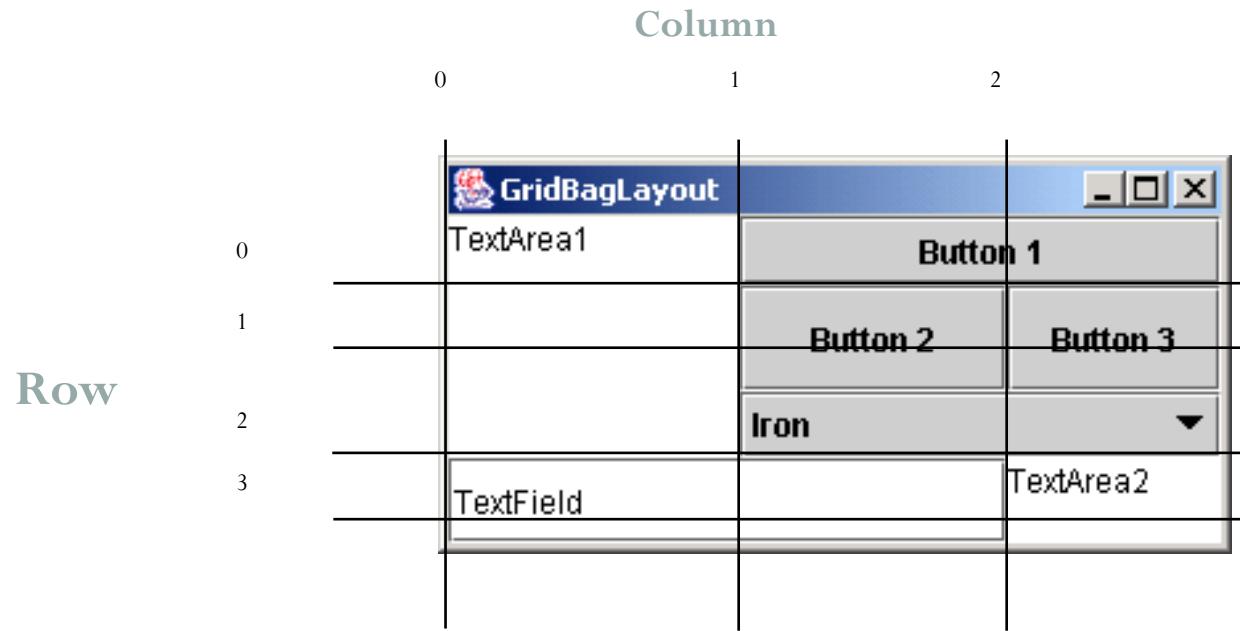
```
setLayout(gridbag);
```

# Setting Constraints for Components

- For each component that you add to the gridbag layout, you must first set an instance of the GridBagConstraints class.
- Let us look at an example:

```
constraints.gridx = 0 // put component in first cell.  
constraints.gridy = 0  
gridbag.setConstraints(button1, constraints);  
                                //set the constraints to button1.  
add(button1);  
                                //add to the Frame
```

# Precisely Placing Components



# GridBagConstraints Data Members

The following are the constraints to control the component:

**gridx, gridy, gridwidth, gridheight, weightx, weighty, ipadx, ipady , fill**

**constraints.gridx=0;** //To set the column (Default is RELATIVE).

**constraints.gridy=0;** //To set the row (Default is RELATIVE).

**constraints.gridwidth = 2;** //Joins two cells horizontally

**constraints.gridheight = 2;** //Joins two cells vertically

//Default value is 1.

```
constraints.weightx=2;          //Determines horizontal spacing between cells.  
constraints.weighty=3;          // Determines vertical spacing between cells.  
                                //Default value is 0.  
  
constraints.ipadx=2;  
                                //Specifies extra horizontal space that surrounds a component within a cell.  
constraints.ipady=0;  
                                //Specifies extra vertical space that surrounds a component within a cell.  
                                //Default is 0.  
  
fill      //Resizing rules for a component smaller than its display area;  
           //HORIZONTAL makes component as wide as the display area;  
           //VERTICAL as tall; BOTH expands it vertically and horizontally.  
           //NONE (default) is default.
```

Examples:

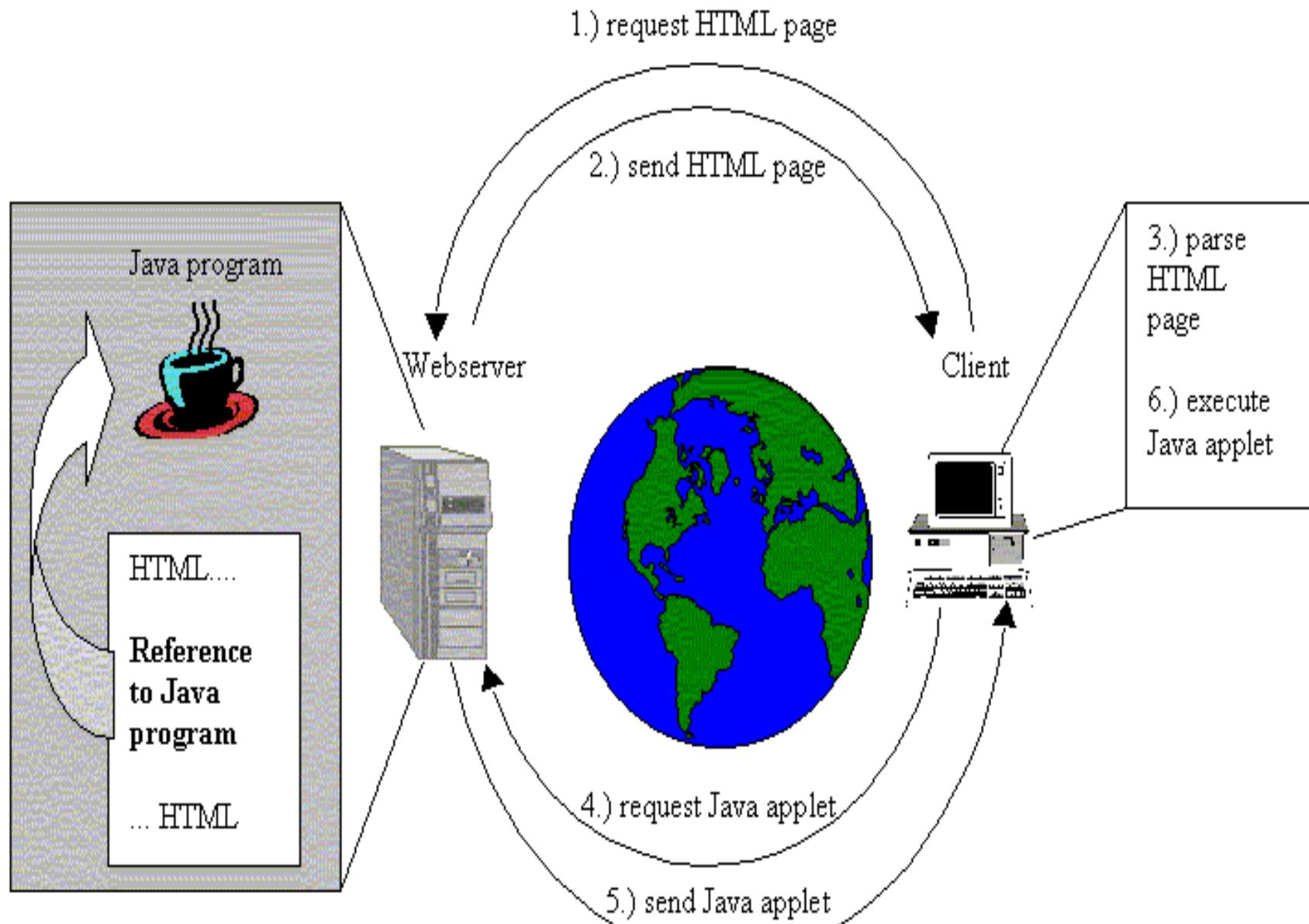
```
constraints.fill = GridBagConstraints.BOTH;          //Fill cell height & width  
constraints.fill = GridBagConstraints.HORIZONTAL;    //Fill cell width
```

## What is an applet?

**Applet:** A small Java program that can be inserted into a web page and run by loading that page in a browser.

An applet is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.

Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document.



# Applet classes in Java

java.lang.Object

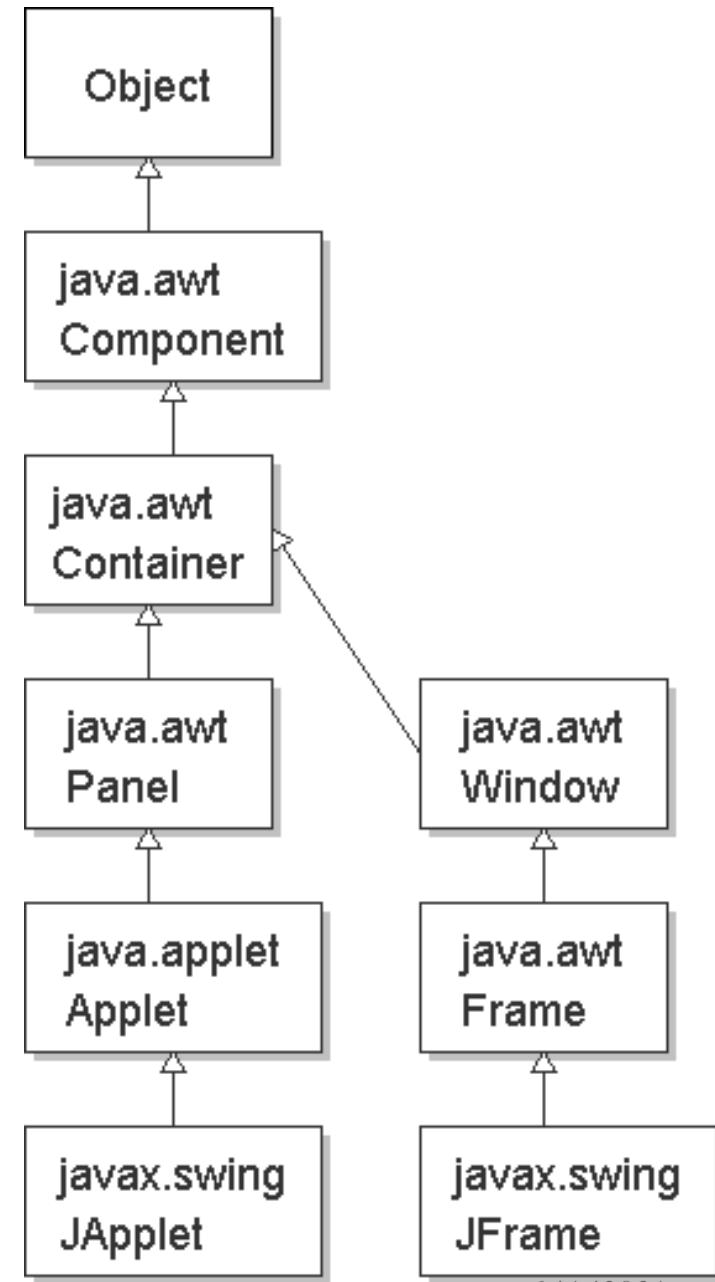
java.awt.Component

java.awt.Container

java.awt.Panel

**java.applet.Applet**

**javax.swing.JApplet**



## How Applets Differ from Applications

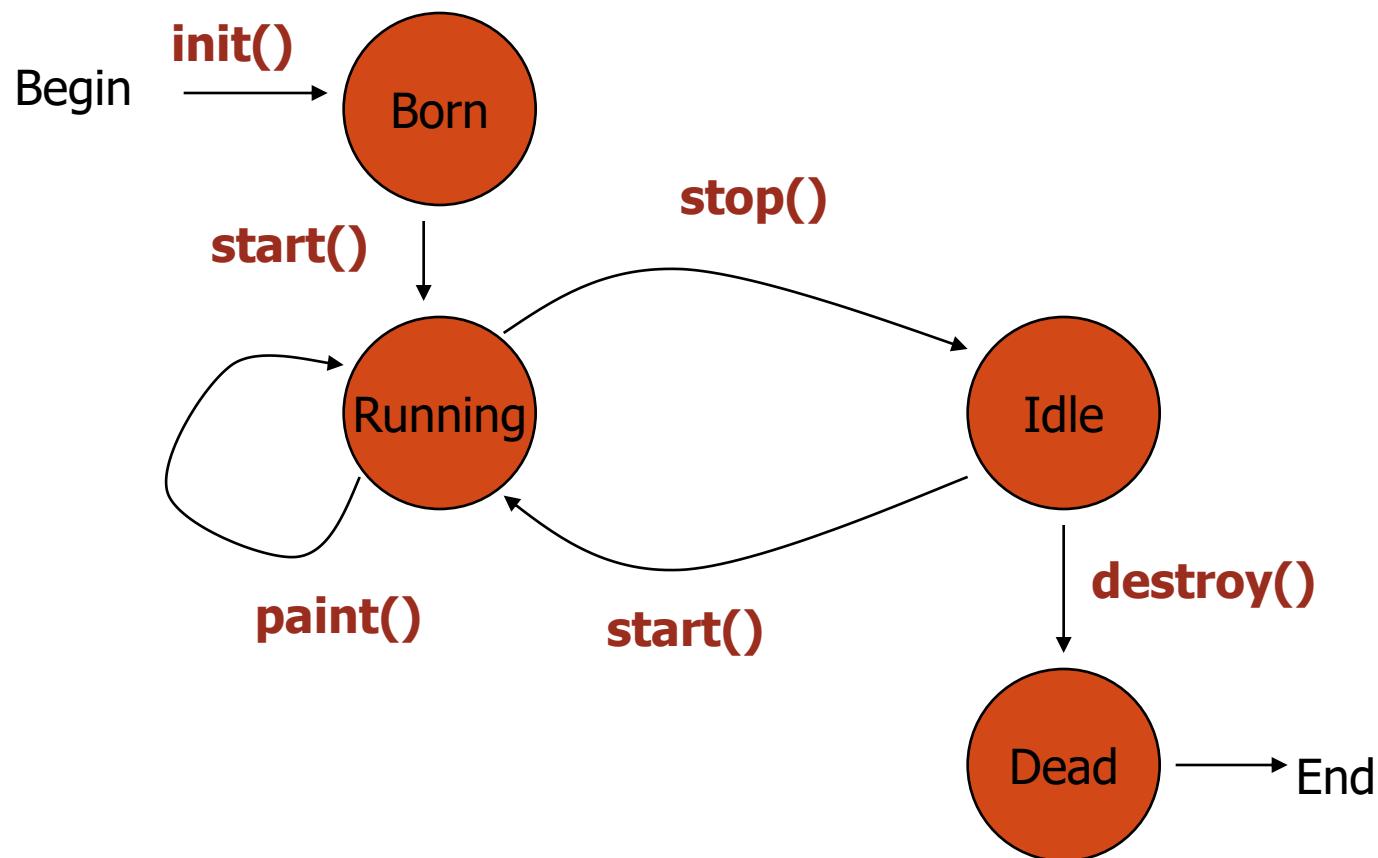
Although both the Applets and stand-alone applications are Java programs, there are certain restrictions imposed on Applets due to security concerns:

- Applets don't use the main() method, but when they are loaded, automatically call certain methods (init, start, paint, stop, destroy).
- They are embedded inside a web page and executed in browsers.
- Takes input through Graphical User Input ( GUI ).
- They cannot read from or write to the files on local computer.
- They cannot run any programs from the local computer.
- They are restricted from using libraries from other languages.

The above restrictions ensures that an Applet cannot do any damage to the local system.

1. Applets can be embedded in HTML pages and downloaded over the Internet whereas Applications have no special support in HTML for embedding or downloading.
2. Applets can only be executed inside a java compatible web browser or appletviewer whereas Applications are executed at command line by java.
3. After an applet arrives on the client, it has limited access to resources on local computer. Applications have no restriction to access resources.
4. Applets don't have the main() method as in applications. Instead they operate on an entirely different mechanism where they are initialized by init(), started by start(), stopped by stop() or destroyed by destroy().
5. A Java Applet is made up of at least one public class that has to be subclasses from java.applet.Applet. Whereas, A Java application is made up of a main() method declared as public static void that accepts a string array argument, along with any other classes that main() calls.

# Life cycle of an Applet



# Life cycle of an Applet

It is important to understand the **order** in which these methods are called.

- When an applet is started , the following sequence of method calls takes place:
  1. init( )
  2. start( )
  3. paint( )
  
- When an applet is terminated, the following sequence of method calls takes place:
  1. stop( )
  2. destroy( )

# Applet States

## Initialisation

- The **init( )** method is the first method to be called.
- This is where you should initialize variables.
- This method is called **only once** during the run time of your applet.

## Running – more than once

- The **start( )** method is called after **init( )**.
- It is also called to **restart** an applet after it has been **stopped**.
- It is called each time an applet's HTML document is **displayed on screen**.
- So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

## Display – more than once

- **paint()** happens immediately after the applet enters into the running state.  
It is responsible for displaying output.
- **paint( )** is also called when the applet begins execution.
- The **paint( )** method is called each time your applet's output must be redrawn.
- The **paint( )** method has one parameter of type **Graphics**.

## Idle

- The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page.

## Dead/Destroyed State – only once

- The **destroy( )** method is called when the environment determines that your applet needs to be **removed completely from memory**.
- At this point, you should free up any resources the applet may be using. The **stop()** method is always **called before destroy()**.

# Types of applets

There are two types of applets:

- First is based on the **Applet** class
  - These applets use the AWT to provide the GUI.
  - This style of applet has been available since java was created.
  - It is used for simple GUI's.
- The second is based on the Swing class **JApplet**.
  - These applets use the Swing classes to provide the GUI.
  - Swing offers a richer and easy to use interface than AWT .
  - Swing based applets are more popular.

# Structure of an applet

```
// An Applet AppletStructure  
import java.awt.*;  
import java.applet.*;  
  
/* <applet code="AppletStructure"  
width=300 height=100> </applet> */  
  
public class AppletStructure extends Applet {  
  
    // Called first.  
    public void init() {  
        // initialization  
    }  
  
    /* Called second, after init(). Also called  
    whenever the applet is restarted. */  
    public void start() {  
        // start or resume execution  
    }  
}
```

```
// Called when the applet is stopped.  
public void stop() {  
    // suspends execution  
}  
  
/* Called when applet is terminated.  
This is the last method executed. */  
public void destroy() {  
    // perform shutdown activities  
}  
  
// Called whenever an applet's output  
must be redisplayed.  
public void paint(Graphics g) {  
    // redisplay contents of window  
}  
}
```

# Building Applet Code: An Example

```
import java.awt.*;
import java.applet.Applet;
public class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString ("A Simple Applet",20, 20);
    }
}
```

- Begins with two import classes.
  - **java.awt.\*** -- required for GUI
  - **java.applet.\*** -- every applet you create must be a subclass of Applet, which is in java.applet package.
- The class should start with **public**, because is accessed from outside.

- Applets do not begin execution at **main()**.
- An applet begins its execution when the name of its class is passed to an **applet viewer** or to a **java compatible browser**.
- Compile the applet in the same way that we have been compiling programs.
- Running an applet involves a different process.

# Running an Applet

There are two ways in which you can run an applet:

- Executing the applet within a **Java-compatible browser**.
- Using a tool called , **appletviewer**. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

## Executing in a web browser.

- To execute an applet in a web browser, you need to write a short HTML file that contains a tag ( **Applet** ) that loads the applet.

### HTML file that contains a SimpleApplet

```
<APPLET code="SimpleApplet" width=400 height=300>
</APPLET>
```

- Save the file with **.html** extension (Example: Simple.html)
- After you create this file, open your browser and then load this file, which causes SimpleApplet to be executed.
- **width** and **height** specify the dimensions of the display used by the applet.

# Executing by using appletviewer

There are two ways

1. Use earlier html page, which contains applet tag, then execute by using following command.

**C:\>appletviewer SimpleApplet.html**

2. Include a comment at the beginning of your source code file that contains the applet tag, then start applet viewer with your java source code file. **C:\>appletviewer SimpleApplet.java**

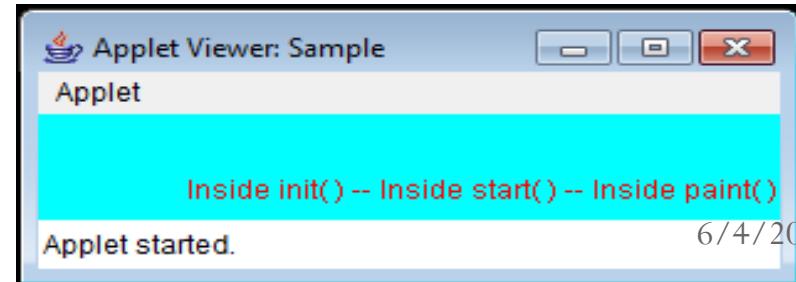
```
import java.awt.*;
import java.applet.Applet;
/* <applet code="SimpleApplet" width=200 height=60></applet> */
public class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString ("A Simple Applet",20, 20);
    }
}
```

- Four of these methods **init()**, **start()**, **stop()**, and **destroy()** are defined by **Applet**.
- Another, **paint()** is defined by the **AWT Component** class.
- Although the above program does not do anything, it can be compiled and run.

# Creating Applets

/\* A simple applet that sets the foreground and background colors and outputs a string. \*/

```
import java.awt.*;
import java.applet.*;
/* <applet code="Sample" width=300 height=50> </applet> */
public class Sample extends Applet{
    String msg;
    public void init() { // set the foreground and background colors.
        setBackground(Color.cyan);
        setForeground(Color.red);
        msg = "Inside init( ) --";
    }
    public void start() { // Initialize the string to be displayed.
        msg += " Inside start( ) --";
    }
    public void paint(Graphics g) { // Display msg in applet window.
        msg += " Inside paint( ).";
        g.drawString(msg, 60, 40);
    }
}
```



# Passing Parameters to Applet

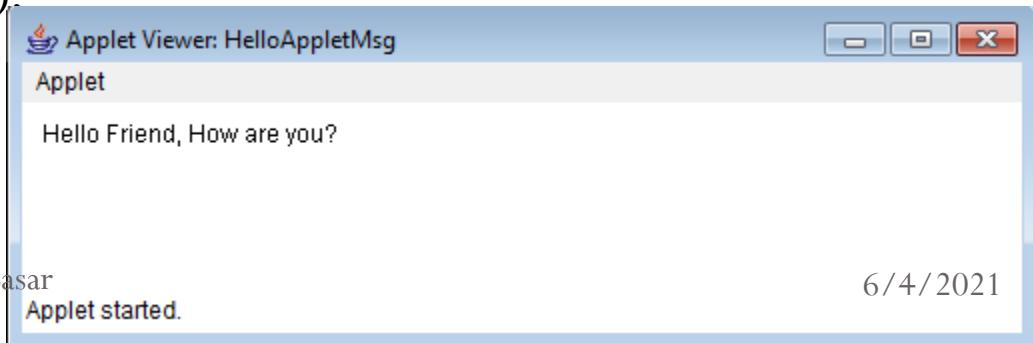
- The APPLET tag in HTML allows you to pass parameters to your applet.
- To retrieve a parameter, use the **getParameter( )** method.
- It returns the value of the specified parameter in the form of a **String** object.

# Applet Program Accepting Parameters

```
import java.applet.Applet;  
import java.awt.*;  
/* <APPLET CODE="HelloAppletMsg" width=500 height=400>  
   <PARAM NAME="Greetings" VALUE="Hello Friend, How are you?">  
</APPLET> */
```

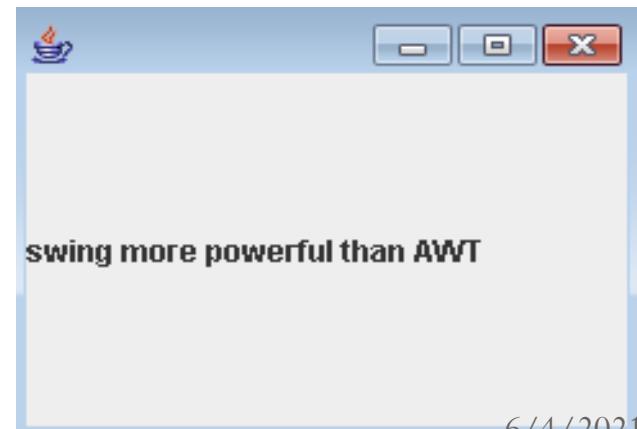
```
public class HelloAppletMsg extends Applet {  
    String msg;  
    public void init(){  
        msg = getParameter("Greetings");  
        if( msg == null)  
            msg = "Hello";  
    }  
    public void paint(Graphics g) {  
        g.drawString (msg,10, 100);  
    }  
}
```

This is name of parameter specified in PARAM tag;  
This method returns the value of parameter.



# Simple Swing Application

```
import java.awt.*;
import javax.swing.*;
public class myjframe extends JFrame{
    myjframe(){
        Container contentPane = getContentPane();
        JLabel jl=new JLabel("swing more powerful than AWT");
        contentPane.add(jl);
        setSize(250,250);
        setVisible(true);
    }
    public static void main(String arg[]){
        new myjframe();
    }
}
```



## **SYLLABUS:**

### **UNIT-5:**

**I/O Streams:** File, Streams, Advantages, The stream classes, Byte streams, Character streams.

#### **Introduction:**

#### **Input/Output Stream:**

Understand the basis of file handling

Understand how Input and Output operations can be done in Java

Understand how input is taken from the User

We will get introduced to **java.io** package

Two parts of computer: Giving input and Displaying Output

I/O classes are going to form core parts of any Programming lang.

Java 1.4 two predefined packages : **java.io** package [ input & output] – reading/writing content to console or a file

**Java.nio** package- advance operations – buffering, memory mapping, character encoding and decoding , pattern matching and locking a file ...

**Java.io** package-provide classes for reading and writing data ( Byte and character)

I/O is based on Streams

**Stream:** A stream is a continuous flow of Data – Byte stram Data and Charater stream data

**Byte Stream:** BS classes deals with reading and writing of bytes to files, socket...

**Character Stream:** CS classes deal with reading and writing characters to files or socket,etc...

**Java.io – java.io.InputStream** [ for reading bytes]

**Java.io.OutputStream**[ for writing bytes]

**Java.io.Reader**( for reading characters)

**Java.io.Writer**( for writing characters)

- **Java I/O** (Input and Output) is used *to process the input and produce the output.*
- Java uses the concept of a stream to make I/O operation fast.
- The **java.io package** contains all the classes required for input and output operations.
- We can perform **file handling in Java** by Java I/O API.
  
- A **Stream** is a **logical handle** to a input source from which you can **read** data or to a output source where you can **write** data

**Ex:**

- Operation system – Copying files from one location to another location
- Youtube Downlaoder- Writing video to Browser & then back reading video back to the end user through network stream
- **Byte Stream in java:** Classes in the Byte Stream allow you to read and write one Byte at a time. Where it could be a file or a network location
- **Character Stream in java:** Classes in the character stream like file reader, file writer they read one character at a time. In java characters are represented by uni-code which uses 2bytes
- **The Buffered Streams** are the wrapper classes that wrap the Byte streams and character streams and they provide the functionality of not going to the operating system every time. Which will improve the performance.

**Streams**

- Java programs perform I/O through streams
- A **stream** refers to reading or writing information
- **Input stream** refers to reading input: from a disk file, a keyboard, or a network socket
- **Output stream** refers to writing output: to the console, a disk file, or a network connection
- **Character stream** is used for handling input and output of characters
- **FileInputStream** class creates an InputStream that can be used to read bytes from a file
- **FileOutputStream** class creates an OutputStream that can be used to write bytes to a file
  
- **Java I/O** (Input and Output) is used to process the input and produce the output.
- Java uses the concept of stream to make I/O operation fast.

- The **java.io** package contains all the classes required for input and output operations.
- We can perform **file handling in java** by Java I/O API.
- **A stream** is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
- In java, 3 streams are created for us automatically
- 1) **System.out:** standard output stream
- 2) **System.in:** standard input stream
- 3) **System.err:** standard error stream

**code to print output and error message to the console.**

```
System.out.println("simple message");
```

```
System.err.println("error message");
```

**code to get input from console.**

```
int i=System.in.read();//returns ASCII code of 1st character
```

```
System.out.println((char)i);//will print the character
```

### **OutputStream vs InputStream**

The explanation of OutputStream and InputStream classes are given below:

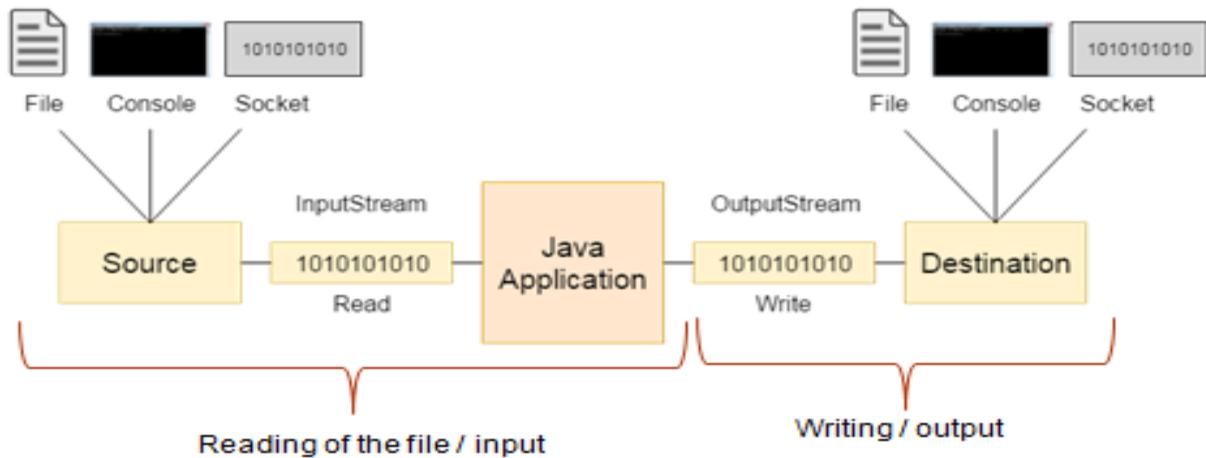
#### **OutputStream**

- Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.

#### **InputStream**

- Java application uses an input stream to read data from a source, it may be a file, an array, peripheral device or socket.

## Java OutputStream and InputStream



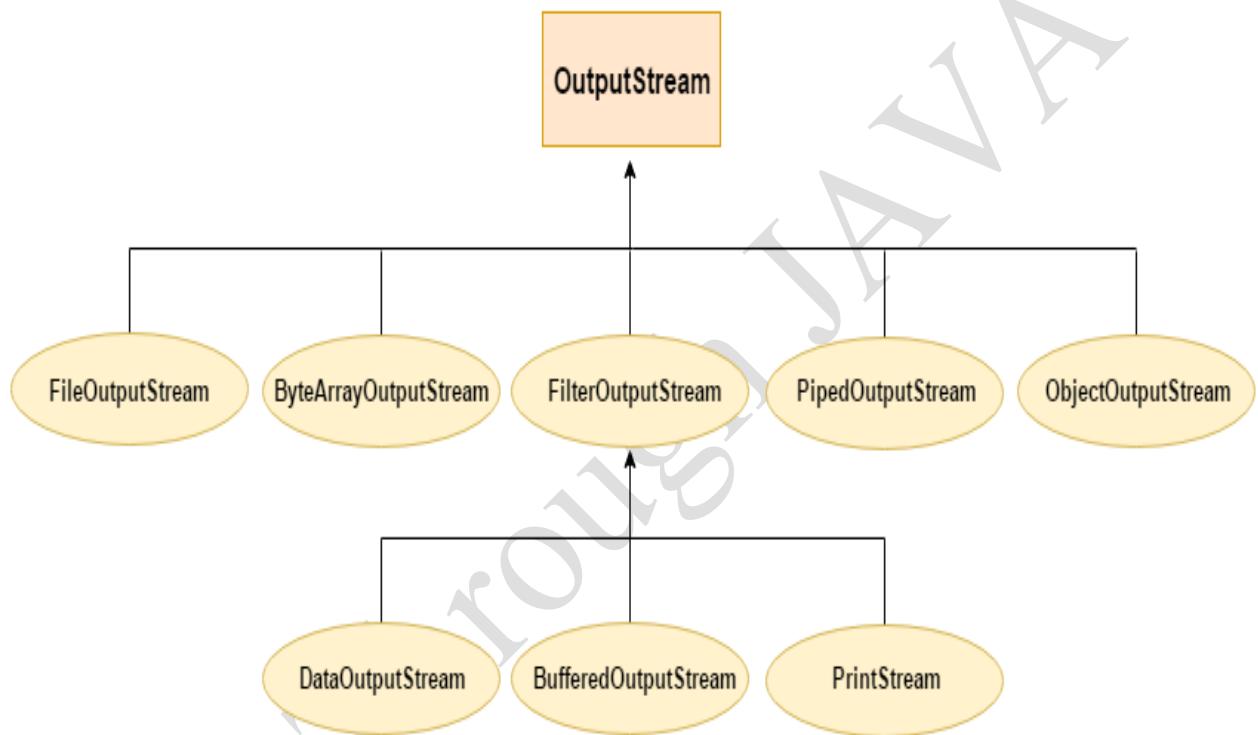
### OutputStream class

- OutputStream class is an abstract class. It is the super class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.
- **Useful methods of OutputStream**

| Method  | Description   |
|---|---|
| 1) public void write(int) throws IOException    | is used to write a byte to the current output stream.           |
| 2) public void write(byte[]) throws IOException | is used to write an array of byte to the current output stream. |
| 3) public void flush() throws IOException       | flushes the current output stream.                              |

4) public void close()throws IOException is used to close the current output stream.

### OutputStream Hierarchy



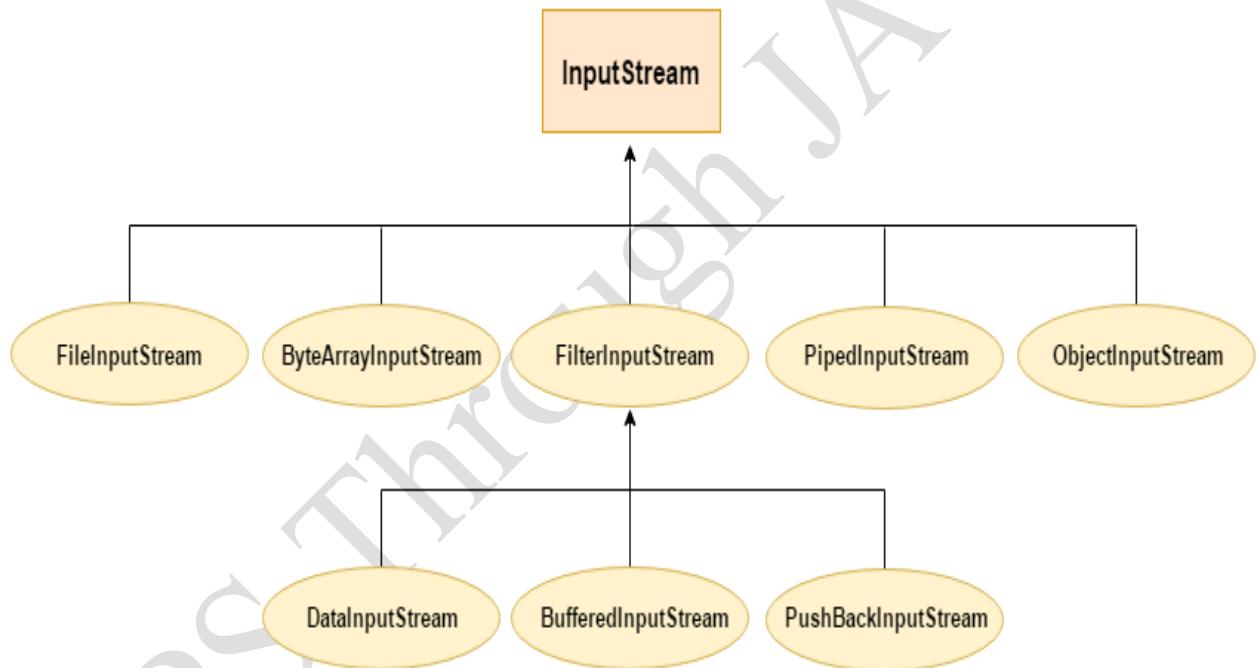
### InputStream class

- InputStream class is an abstract class. It is the super class of all classes representing an input stream of bytes.
- Useful methods of InputStream

| Method  | Description  |
|---|--|
| 1) public abstract int read()throws IOException | reads the next byte of data from the input stream. It returns -1 at the end of file. |

|   |  |
|---|--|
| 2) public int available()throws IOException | returns an estimate of the number of bytes that can be read from the current input stream. |
| 3) public void close()throws IOException    | is used to close the current input stream.   |

### InputStream Hierarchy



### **Java FileOutputStream Class**

- Java FileOutputStream is an output stream used for writing data to a file.
- If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

**FileOutputStream class declaration**

- Declaration for Java.io.FileOutputStream class:
- **public class FileOutputStream extends OutputStream**

***BufferedReader* class is used to read character stream**

| Stream Class     | Meaning   |
|------------------|---|
| DataInputStream  | An input stream that contains methods for reading the Java standard data types  |
| DataOutputStream | An output stream that contains methods for writing the Java standard data types |
| FileInputStream  | Input stream that reads from a file   |
| FileOutputStream | Output stream that writes to a file   |
| BufferedReader   | Buffered input character stream   |

**FileOutputStream class methods**

| Method                                       | Description   |
|--|---|
| protected void finalize()                    | It is sued to clean up the connection with the file output stream.                                  |
| void write(byte[] ary)                       | It is used to write ary.length bytes from the byte array to the file output stream.                 |
| void write<br>(byte[] ary, int off, int len) | It is used to write len bytes from the byte array starting at offset off to the file output stream. |
| void write(int b)                            | It is used to write the specified byte to the file output stream.                                   |
| FileChannel getChannel()                     | It is used to return the file channel object associated with the file output stream.                |
| FileDescriptor getFD()                       | It is used to return the file descriptor associated with the stream.                                |
| void close()                                 | It is used to closes the file output stream.  |

## Java FileInputStream Class

- Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class

### Java FileInputStream Class declaration

Declaration for java.io.FileInputStream class:

```
public class FileInputStream extends InputStream
```

### Java FileInputStream class methods

| Method                               | Description  |
|--------------------------------------|--|
| int available()                      | It is used to return the estimated number of bytes that can be read from the input stream. |
| int read()                           | It is used to read the byte of data from the input stream.                                 |
| int read(byte[] b)                   | It is used to read up to b.length bytes of data from the input stream.                     |
| int read(byte[] b, int off, int len) | It is used to read up to len bytes of data from the input stream.                          |

|                           |  |
|---------------------------|--|
| long skip(long x)         | It is used to skip over and discards x bytes of data from the input stream.                                  |
| FileChannel getChannel()  | It is used to return the unique FileChannel object associated with the file input stream.                    |
| FileDescriptor getFD()    | It is used to return the FileDescriptor object.  |
| Protected void finalize() | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close()              | It is used to closes the stream.   |

## Java BufferedInputStream Class

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.
- The important points about BufferedInputStream are:
- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.

### Java BufferedInputStream class declaration

```
public class BufferedInputStream extends FilterInputStream
```

### Java BufferedInputStream class constructors

| Constructor                                   | Description   |
|---|---|
| BufferedInputStream(InputStream IS)           | It creates the BufferedInputStream and saves its argument, the input stream IS, for later use.                              |
| BufferedInputStream(InputStream IS, int size) | It creates the BufferedInputStream with a specified buffer size and saves its argument, the input stream IS, for later use. |

### Java BufferedInputStream class methods

| Method                              | Description  |
|-------------------------------------|--|
| int available()                     | It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream. |
| int read()                          | It reads the next byte of data from the input stream.  |
| int read(byte[] b, int off, int ln) | It reads the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.                               |
| void close()                        | It closes the input stream and releases any of the system resources associated with the stream.  |
| void reset()                        | It repositions the stream at a position the mark method was last called on this input stream.  |
| void mark(int readlimit)            | It sees the general contract of the mark method for the input stream.  |
| long skip(long x)                   | It skips over and discards x bytes of data from the input stream.  |
| boolean markSupported()             | It tests for the input stream to support the mark and reset methods.   |

### Java BufferedOutputStream Class

- Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class.
- Syntax:**

```
OutputStream os= new BufferedOutputStream(new FileOutputStream("D:\\IO Package\\testout.txt"));
```

### Java BufferedOutputStream class declaration

```
public class BufferedOutputStream extends FilterOutputStream
```

### Java BufferedOutputStream class constructors

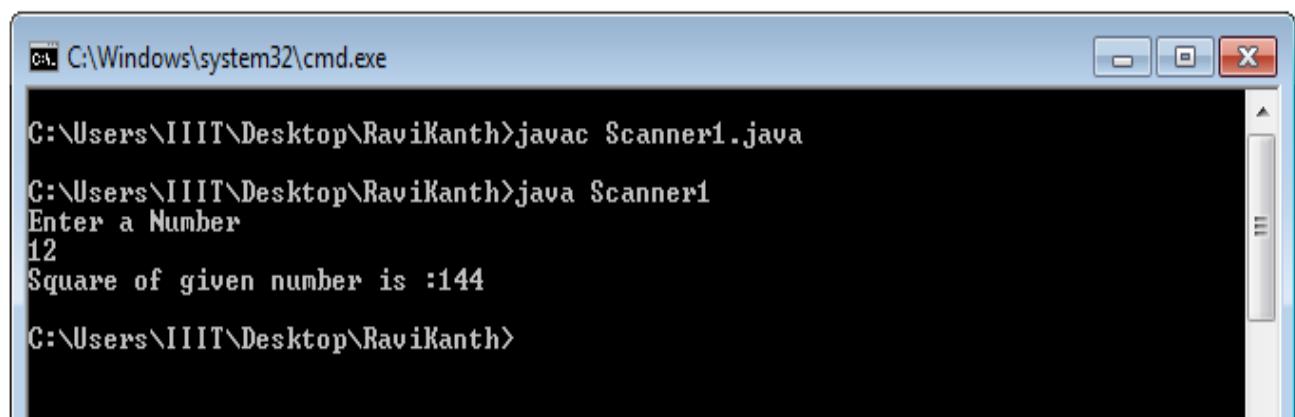
| Constructor                                     | Description   |
|---|---|
| BufferedOutputStream(OutputStream os)           | It creates the new buffered output stream which is used for writing the data to the specified output stream.                              |
| BufferedOutputStream(OutputStream os, int size) | It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size. |

**Java BufferedOutputStream class methods**

| Method                                 | Description   |
|--|---|
| void write(int b)                      | It writes the specified byte to the buffered output stream.   |
| void write(byte[] b, int off, int len) | It write the bytes from the specified byte-input stream into a specified byte <b>array</b> , starting with the given offset |
| void flush()                           | It flushes the buffered output stream.  |

**Example Programs:****Ex1:**

```
// Example using Scanner class and nextInt() method from util package
import java.util.Scanner;
class Scanner1
{
    public static void main(String args[])
    {
        int number;
        Scanner obj=new Scanner(System.in);
        System.out.println("Enter a Number");
        number=obj.nextInt();
        System.out.println("Square of given number is :" + (number*number));
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac Scanner1.java
C:\Users\IIIT\Desktop\RaviKanth>java Scanner1
Enter a Number
12
Square of given number is :144
C:\Users\IIIT\Desktop\RaviKanth>
```

**Ex2:**

```
//Example1(DataInputStream)
import java.io.*;
public class KeyboardReading
{
    public static void main(String args[]) throws IOException
    {
        DataInputStream dis = new DataInputStream(System.in);
        System.out.println("Enter your Name: ");
        String str1 = dis.readLine();

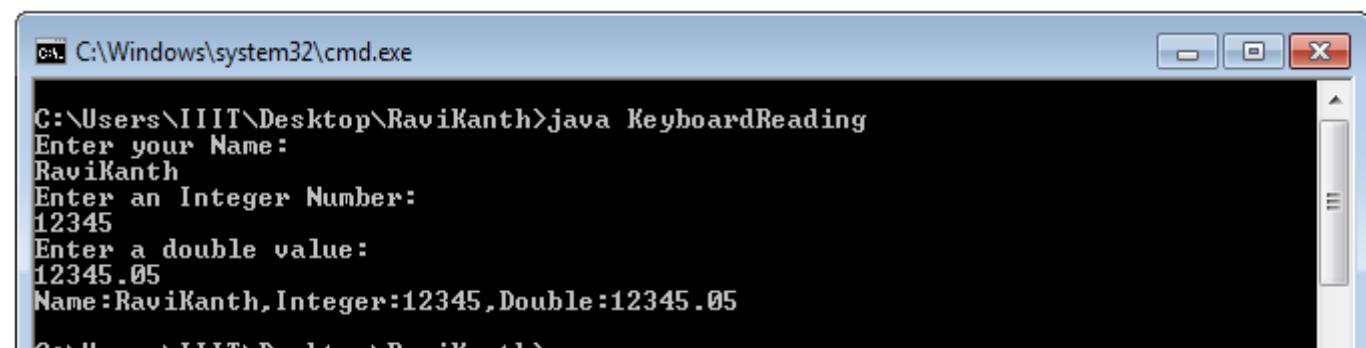
        System.out.println("Enter an Integer Number: ");
        String str2 = dis.readLine();

        int x = Integer.parseInt(str2);

        System.out.println("Enter a double value: ");
        String str3 = dis.readLine();
        double y = Double.parseDouble(str3);

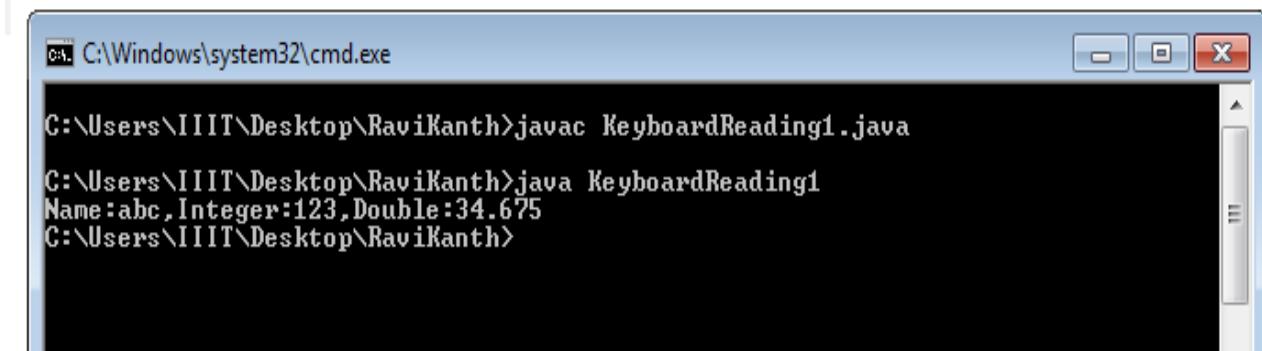
        System.out.println("Name:"+str1+", "+"Integer:"+x+", "+"Double:"+y);

        dis.close();
    }
}
```



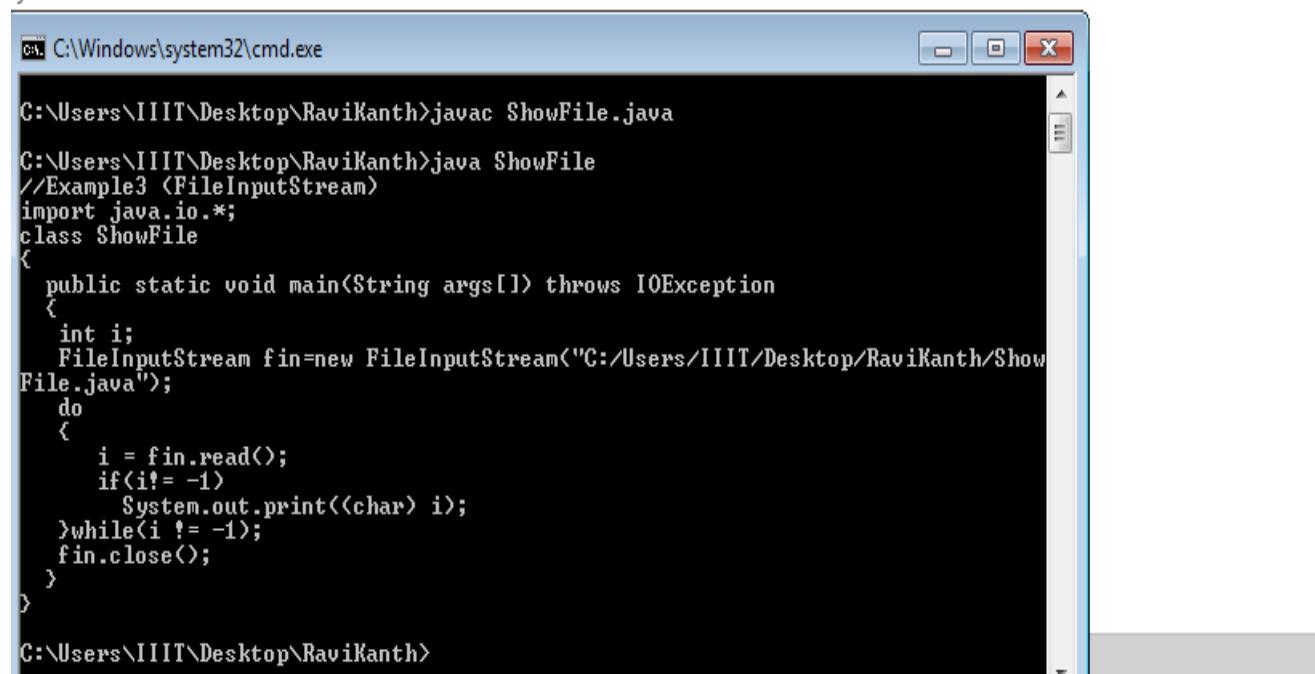
**Ex3:**

```
//Example2 (DataOutputStream)
import java.io.*;
public class KeyboardReading1
{
    public static void main(String args[]) throws IOException
    {
        DataOutputStream dis = new DataOutputStream(System.out);
        String str1="abc";
        int x=123;
        double y=34.675;
        dis.writeBytes("Name:"+str1+","+ "Integer:"+x+", "+"Double:"+y);
        dis.close();
    }
}
```



**Ex 4:**

```
//Example3 (FileInputStream)
import java.io.*;
class ShowFile
{
    public static void main(String args[]) throws IOException
    {
        int i;
        FileInputStream fin=new FileInputStream("C:/Users/IIIT/Desktop/RaviKanth>ShowFile.java");
        do
        {
            i = fin.read();
            if(i!= -1)
                System.out.print((char) i);
        }while(i != -1);
        fin.close();
    }
}
```

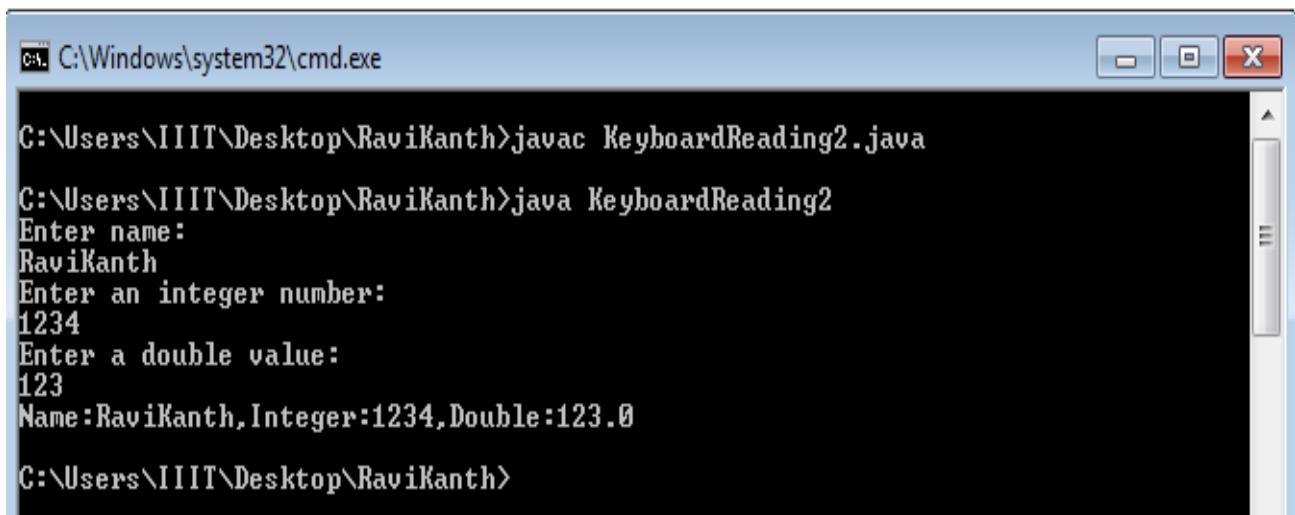


The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The command line shows the execution of a Java program. The user first runs 'javac ShowFile.java' to compile the Java file. Then, they run 'java ShowFile' to execute it. The output of the program is displayed in the console, showing the characters of the Java source code being printed one by one. The console window has a standard blue header bar and a white body.

```
C:\Users\IIIT\Desktop\RaviKanth>javac ShowFile.java
C:\Users\IIIT\Desktop\RaviKanth>java ShowFile
//Example3 <FileInputStream>
import java.io.*;
class ShowFile
{
    public static void main(String args[]) throws IOException
    {
        int i;
        FileInputStream fin=new FileInputStream("C:/Users/IIIT/Desktop/RaviKanth>ShowFile.java");
        do
        {
            i = fin.read();
            if(i!= -1)
                System.out.print((char) i);
        }while(i != -1);
        fin.close();
    }
}
C:\Users\IIIT\Desktop\RaviKanth>
```

**Ex 5:**

```
//Example4(BufferedReader)
import java.io.*;
public class KeyboardReading2
{
    public static void main(String args[]) throws IOException
    {
        BufferedReader dis = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter name: ");
        String str1 = dis.readLine();
        System.out.println("Enter an integer number: ");
        String str2 = dis.readLine();
        int x = Integer.parseInt(str2);
        System.out.println("Enter a double value: ");
        String str3 = dis.readLine();
        double y = Double.parseDouble(str3);
        System.out.println("Name:"+str1+", "+"Integer:"+x+", "+"Double:"+y);
        dis.close();
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac KeyboardReading2.java
C:\Users\IIIT\Desktop\RaviKanth>java KeyboardReading2
Enter name:
RaviKanth
Enter an integer number:
1234
Enter a double value:
123
Name:RaviKanth, Integer:1234, Double:123.0
C:\Users\IIIT\Desktop\RaviKanth>
```

**Ex6:**

```
import java.io.*;
class FileTest
{
    public static void main(String args[]) throws Exception
    {
        File f=new File("C:/Users/IIIT/Desktop/RaviKanth/Addition.java");
        System.out.println(f.getName());
        System.out.println(f.getPath());
        System.out.println(f.getParent());
        System.out.println(f.isDirectory());
        System.out.println(f.isFile());
        System.out.println(f.exists());
        System.out.println(f.length()+"bytes");
        System.out.println(f.canRead());
        System.out.println(f.canWrite());
        System.out.println(f.isHidden());
        System.out.println(f.lastModified());
        java.util.Date d=new java.util.Date(f.lastModified());
        System.out.println(d);
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\Users\IIIT\Desktop\RaviKanth>javac FileTest.java
C:\Users\IIIT\Desktop\RaviKanth>java FileTest
Addition.java
C:\Users\IIIT\Desktop\RaviKanth\Addition.java
C:\Users\IIIT\Desktop\RaviKanth
false
true
true
314bytes
true
true
false
1552755071947
Sat May 16 09:04:44 IST 2010
```

**Ex7:**

```
// list() methods of java.io.File
//list() methods can be used to retrieve the contents of directory
import java.io.*;
class FileTest1
{
    public static void main(String args[])
    {
        File f=new File("C:/Users/IIIT/Desktop ");
        if(f.isDirectory())
        {
            String str[]=f.list();
            for(String s:str)
            {
                System.out.println(s);
            }
        }
        else
            System.out.println("File object is not pointing to directory");
    }
}
```

C:\Windows\system32\cmd.exe

C:\Users\IIIT\Desktop\RaviKanth>javac FileTest1.java

C:\Users\IIIT\Desktop\RaviKanth>java FileTest1

CodeBlocks.lnk  
Database Software  
desktop.ini  
emu8086.lnk  
JDBC, ODBC Drivers.pdf  
LoginWindow1.class  
LoginWindow1.java  
MYSQL DB.docx  
mysql-connector.jar  
PPS Week- 8 Sorting.doc  
RaviKanth  
~~\$SET1.docx  
~~WRL0001.tmp

**Ex8:**

```
// BufferedInput Stream
import java.io.*;
public class BufferedInputStreamExample
{
    public static void main(String args[])
    {
        try
        {
            FileInputStream fin=new FileInputStream("D:\\JAVA.txt");
            BufferedInputStream bin=new BufferedInputStream(fin);
            int i;
            while((i=bin.read())!=-1)
            {
                System.out.print((char)i);
            }
            bin.close();
            fin.close();
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

**Ex9:**

```
//BufferedOutputStream
import java.io.*;
public class BufferedOutputStreamExample
{
    public static void main(String args[])throws Exception
    {
        FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
        BufferedOutputStream bout=new BufferedOutputStream(fout);
        String s="Welcome to javaTpoint.";
        byte b[]={s.getBytes()};
        bout.write(b);
        bout.flush();
        bout.close();
        fout.close();
        System.out.println("success");
    }
}
```

\*\*\*\*\*