

# **JAVA OOPS CONCEPT**

# Procedural Programming:

- Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- There is **no access specifier** in procedural programming i.e Visibility mode.
- **Adding new data and function is not easy.**
- Procedural programming does not have any proper way for hiding data so it is **less secure**.
- Examples: C, FORTRAN, Pascal, Basic etc.

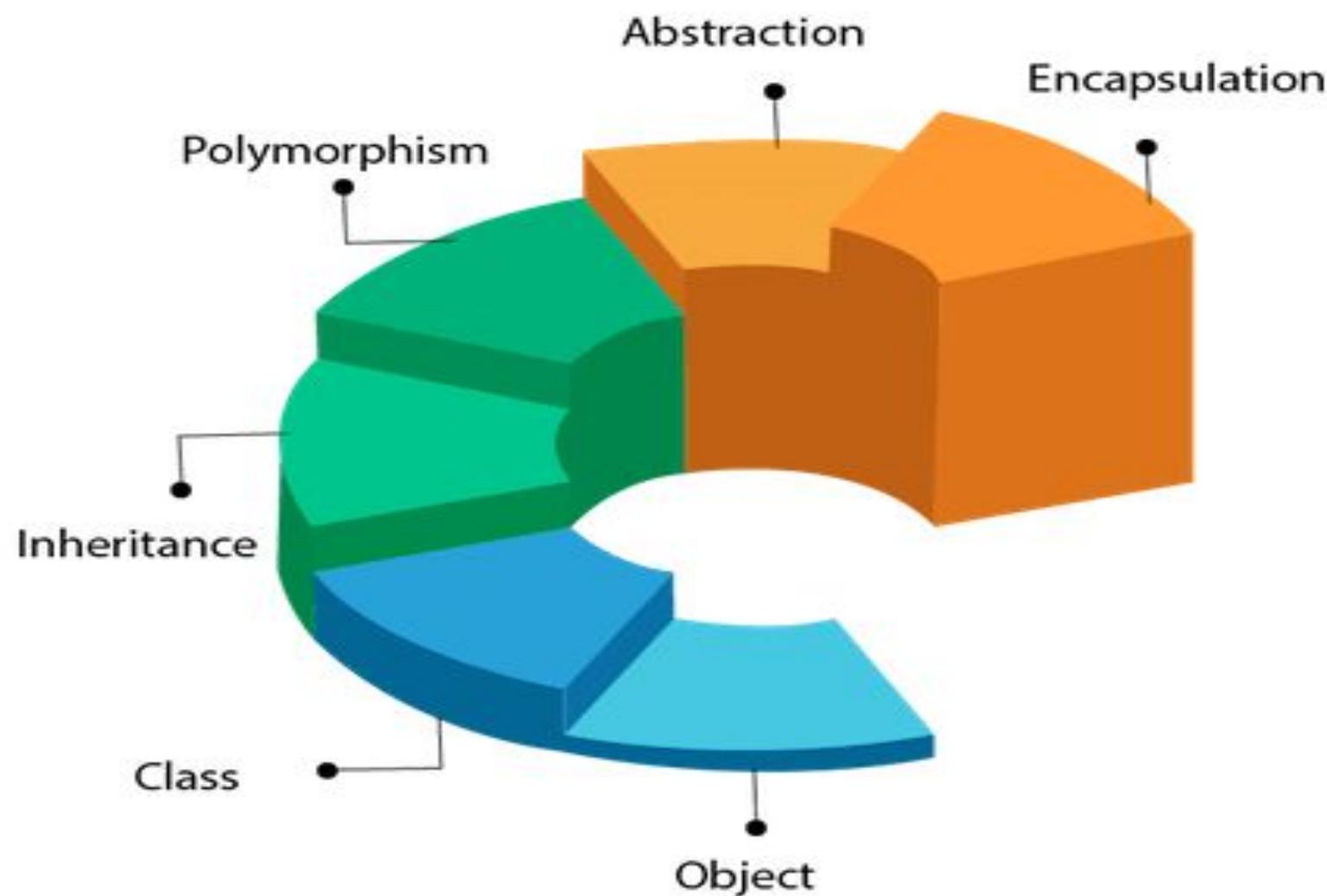
# Object Oriented Programming:

- Object oriented programming can be defined as a programming model which is based upon the **concept of objects**. Objects contain data in the form of **attributes(variables)** and code in the form of **methods(functions)**.
- Object oriented programming **have access specifiers** like private, public, protected etc.
- **Adding new data and function is easy.**
- Object oriented programming provides data hiding so it is **more secure**.
- Examples: C++, Java, Python, C# etc.

# Why Java ?

- Portable
- Independent
- Secure

# OOPs (Object-Oriented Programming System)

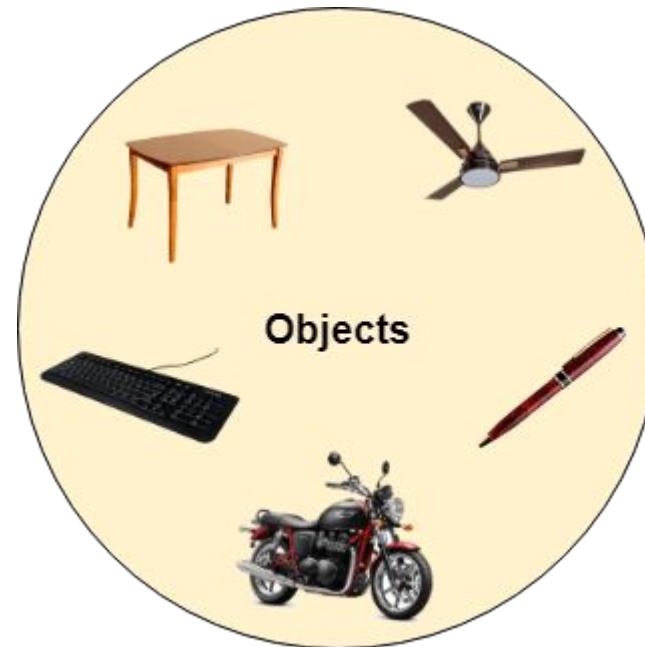


## 1. Object

Any entity that has state and behavior is known as an object.

An Object can be defined as an **instance(event or element) of a class**.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.



## 2. Class

- The class is a group of similar entities.
- A class can also be defined as a blueprint from which you can create an individual object.

## 3. Inheritance

- When one object acquires all the properties and behaviors of a parent object, it is known as inheritance.
- It provides code reusability.

## 4. Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

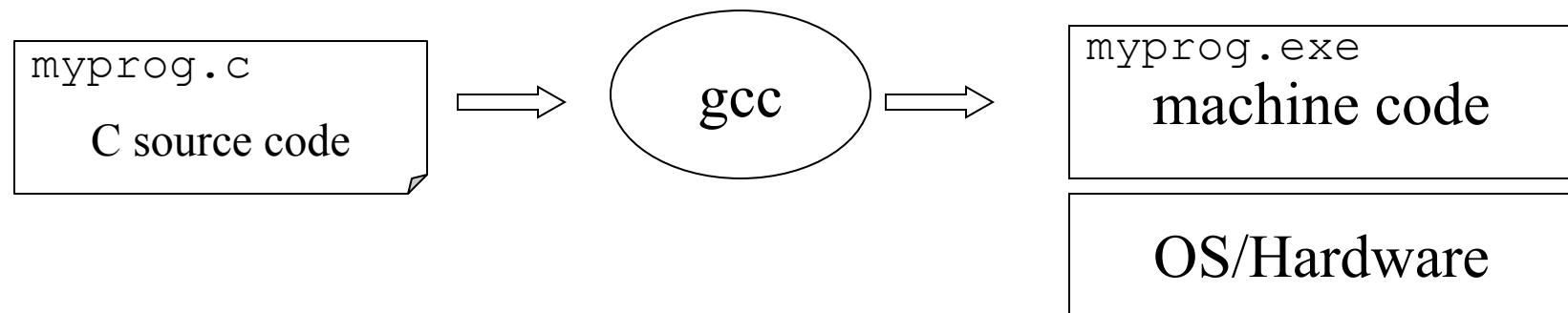
## 5. Abstraction

***Hiding internal details and showing functionality*** is known as abstraction. For example phone call, we don't know the internal processing.

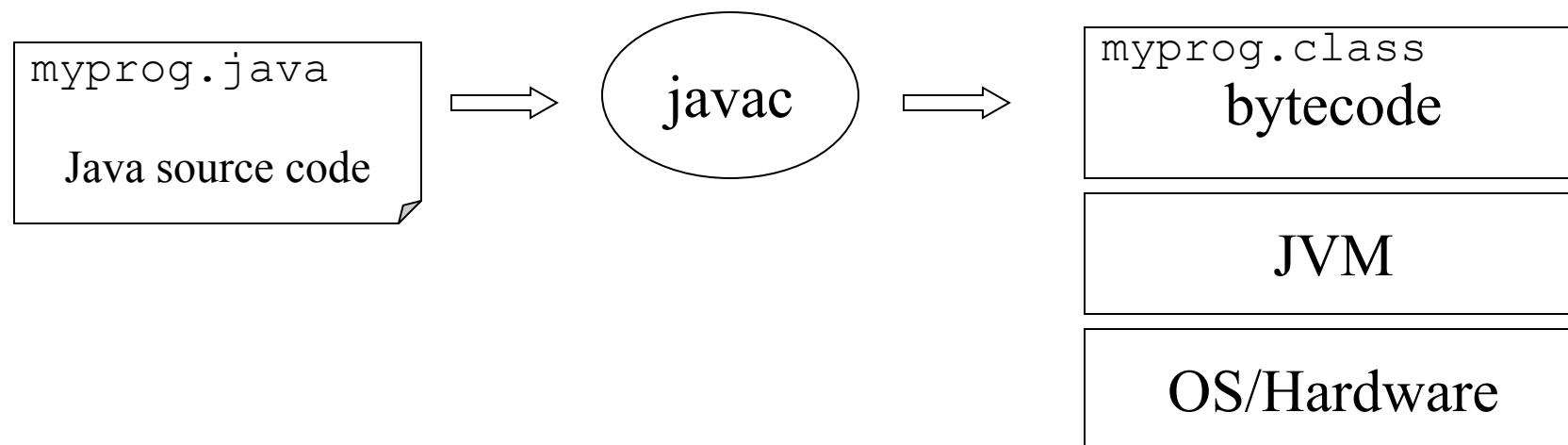
## 6. Encapsulation

*Binding (or wrapping or hiding) code and data together into a single unit are known as encapsulation.*

## Platform Dependent



## Platform Independent



# JVM

- JVM stands for

**Java Virtual Machine**

# Basics of Java:

## **1. Variables or Data Members of class:**

Used to store data or values.

## **2. Datatypes:**

Depending on data or values datatypes where given

E.g. For integer data or values datatype int is used

# Primitive types

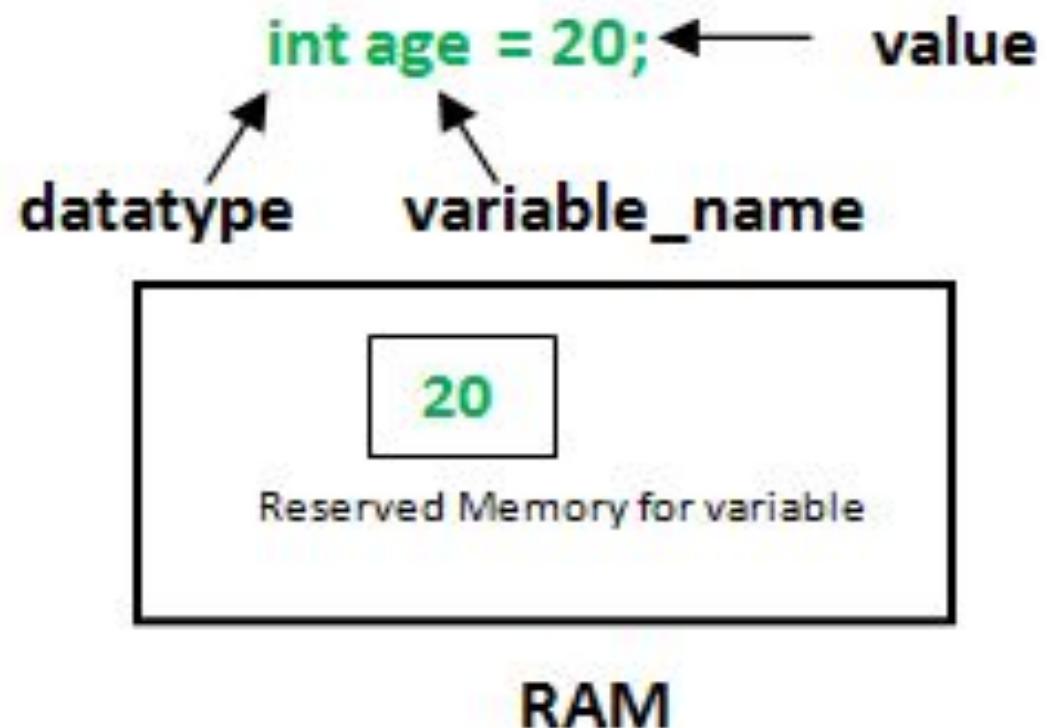
- int 4 bytes
- short 2 bytes
- long 8 bytes
- byte 1 byte
- float 4 bytes
- double 8 bytes
- char Unicode encoding (2 bytes)
- boolean {true,false}

*Behaviors is  
exactly as in  
C++*

**Note:**  
*Primitive type  
always begin  
with lower-case*

## How to declare variables?

We can declare variables in java as follows:



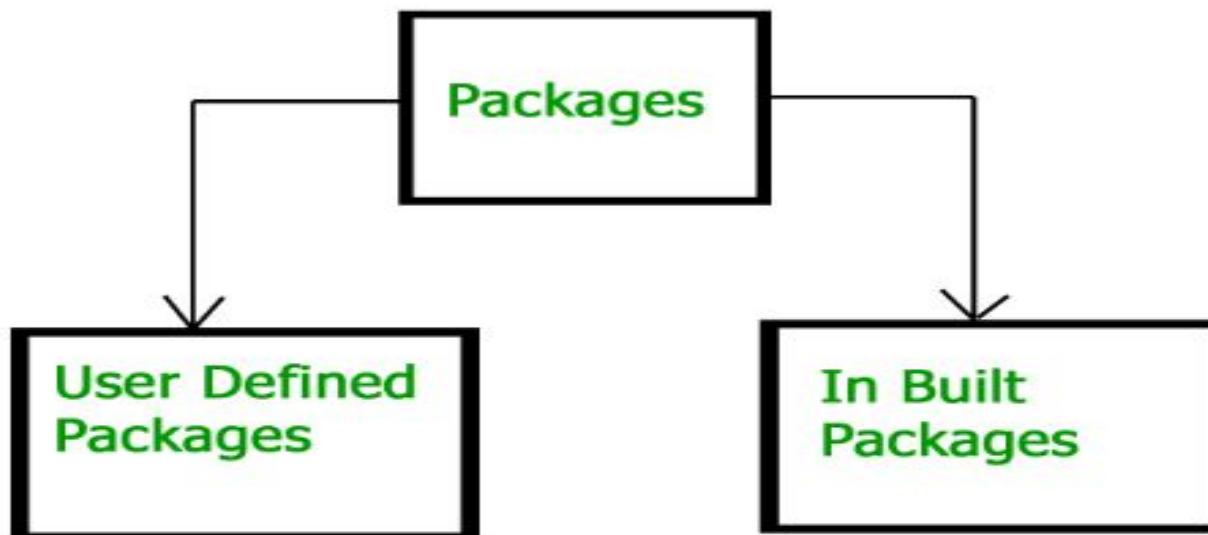
## Packages In Java

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

Example :

```
// import all the classes from util package  
import java.util.*;
```

Types of packages:



# In-Built Packages

1. **java.io:** Contains classes for supporting input / output operations.
2. **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations or Scanner class.

# Java Classes/Objects

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

## Create a Class or Define a Class

To create a class, use the **keyword class**:

### MyClass.java (Save file named as .java)

```
//Create a class called "MyClass"
```

**Syntax:**

```
public class MyClass  
{  
}
```

# Access Control

- ***public*** member (function/data)
  - Can be called/modified/ instantiated from outside.
- ***protected***
  - Can be called/modified from derived classes
- ***private***
  - Can be called/modified only from the current class
- ***default ( if no access modifier stated )***
  - Can be called/modified/instantiated from the same package.

# Hello World

Hello.java

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

C:\javac Hello.java

*(compilation creates Hello.class)*

C:\java Hello

*(Execution on the local JVM)*

# Compiling/running first java program

- Create source code file (call it for example MyFirstProgram.java).
- To compile:

prompt >> javac MyFirstProgram.java

- This produces byte code file named MyFirstProgram.class

- To run:

prompt >> java MyFirstProgram

# Java Methods

- A method is a block of code that performs a specific task.
- Suppose you need to create a program to create a circle and color it. You can create two methods to solve this problem:
  - a method to draw the circle
  - a method to color the circle
- In Java, there are two types of methods:
  - **User-defined Methods:** We can create our own method based on our requirements.
  - **Standard Library Methods:** These are built-in methods in Java that are available to use.

- Declaring a Java Method
- The syntax to declare a method is:

```
AccessSpecifier returnType methodName()  
{  
    // method body  
}
```

# Example 1: Java Methods

```
class Main {  
    // create a method  
    public int addNumbers(int a, int b) {  
        int sum = a + b;  
        // return value  
        return sum;  
    }  
    public static void main(String[] args) {  
        int num1 = 25;  
        int num2 = 15;  
        // create an object of Main  
        Main obj = new Main();  
        // calling method  
        int result = obj.addNumbers(num1, num2);  
        System.out.println("Sum is: " + result);  
    }  
}
```

# Programs

Write program to find area of a square by creating without function

Write program to find area of a square by creating function

# Java User Input(Scanner Class)

The Scanner class is used to get user input, and it is found in the `java.util` package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.

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

# Example

```
import java.util.Scanner; // Import the Scanner class

class Main {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in); // Create a Scanner object
        System.out.println("Enter username");

        String userName = myObj.nextLine(); // Read user input
        float f=myObj.nextFloat();
        System.out.println("Username is: " + userName); // Output user input
    }
}
```

THANK YOU