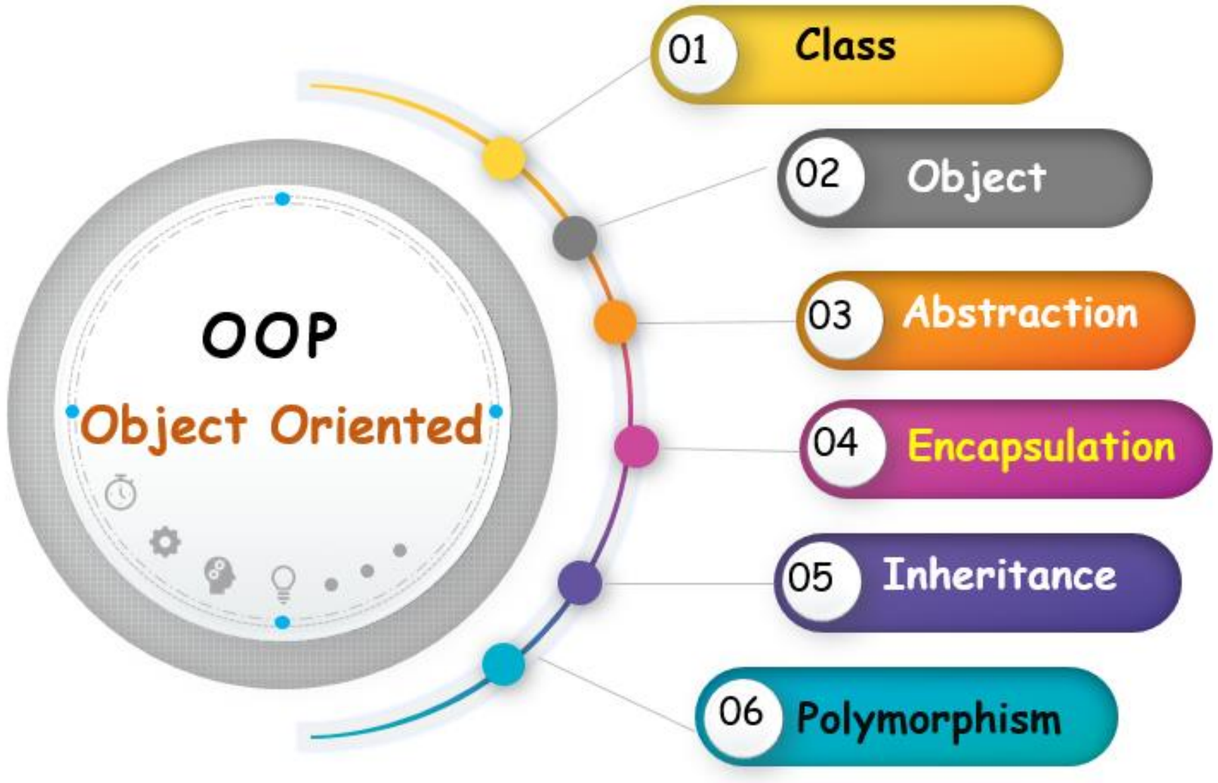
Module 1: Introduction and Encapsulation

Prof. Pramod Bide

Introduction to OOP

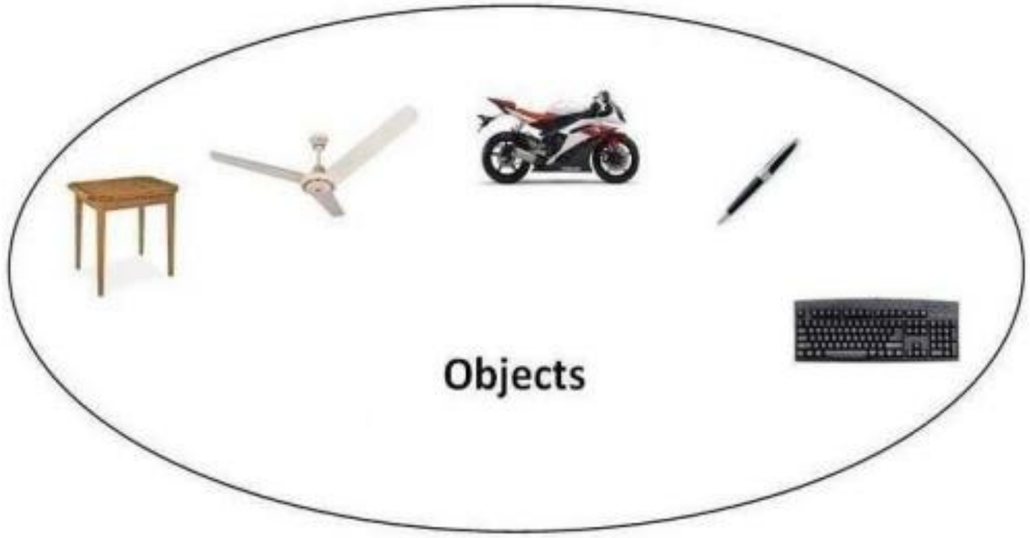
Object Oriented Programming (OOPs) is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.

The core concept of the object-oriented approach is to break complex problems into smaller objects.



Java Class and Objects

- Java is an object-oriented programming language.
- **Any entity that has state and behavior is known as an object.** For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.
- **Collection of objects is called class.** It is a logical entity.
- Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts like object, class, inheritance, polymorphism, encapsulation and abstraction.



How to create a class in Java

We can create a class in Java using the class keyword.

Syntax:

```
class ClassName {
  // fields
  // methods
}
```

- Fields are variables and are used to store data.
- Methods represent the state and behavior of the object and are used to perform some operations

How to create a class in Java

```
class Bicycle {

  // state or field
  private int gear = 5;

  // behavior or method
  public void braking() {
    System.out.println("Working of Braking");
  }
}
```

In the above example, we have created a class named Bicycle. It contains a field named gear and a method named braking(). Here, Bicycle is a prototype. Now, we can create any number of bicycles using the prototype. And, all the bicycles will share the fields and methods of the prototype.

How to create an object in Java

```
className object = new className();

// for Bicycle class
Bicycle sportsBicycle = new Bicycle();

Bicycle touringBicycle = new Bicycle();
```

- We have used the **new** keyword along with the constructor of the class to create an object. Constructors are similar to methods and have the same name as the class. For example, Bicycle() is the constructor of the Bicycle class.

- Here, sportsBicycle and touringBicycle are the names of objects. We can use them to access fields and methods of the class.

- We can create multiple objects of a single class in Java.

Procedural Programming

- Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out.
- During a program’s execution, any given procedure might be called at any point, including by other procedures or itself.
- Languages used in Procedural Programming: FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.

Procedural Programming

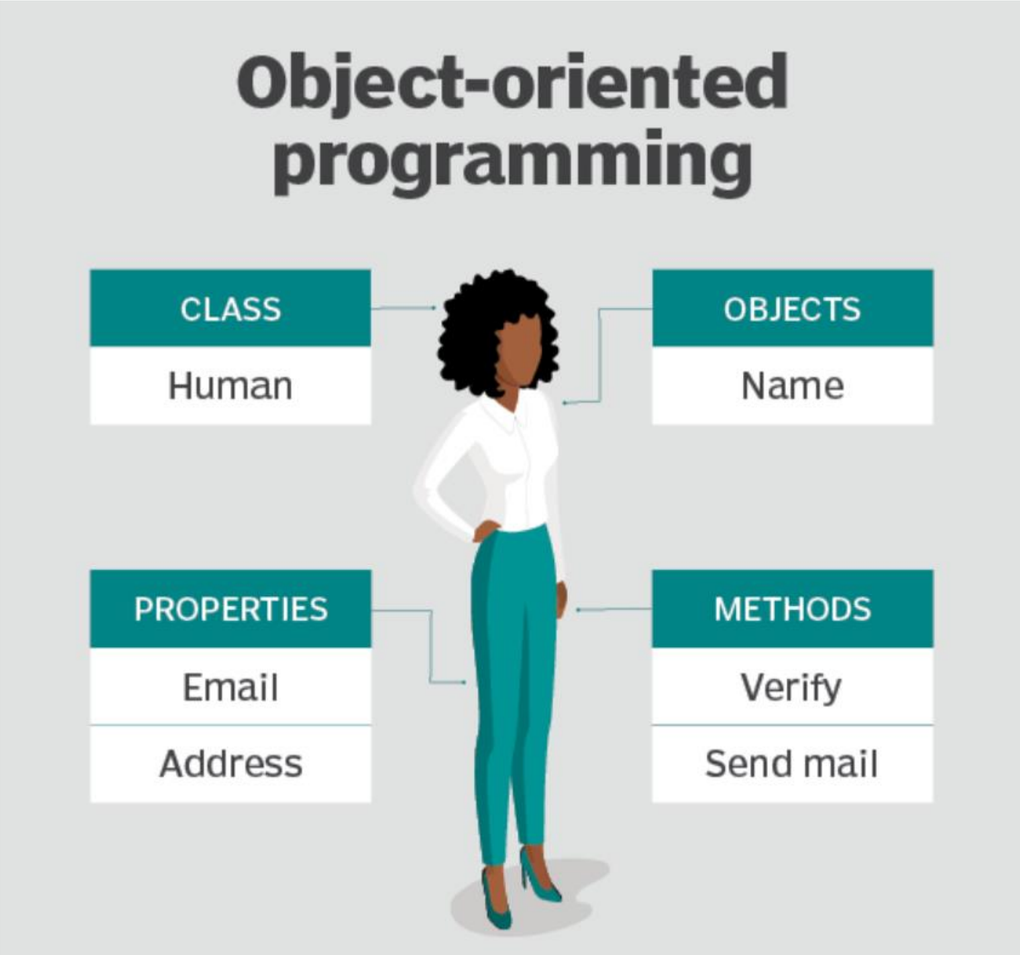
```
public double area(string shape) throws NoSuchShapeException
{
    if ( shape == "Square"){
        return getAreaOfSquare();
    }
    else if(shape == "Circle"){
        return getAreaOfCircle();
    }
    else if(shape == "Rectangle"){
        return getAreaOfRectangle();
    }
    throw new NoSuchShapeException();
}
```

- In a procedural programming language, a program basically consists of a sequence of instructions each of which tells the computer to do something such as we are doing it, and functions are written for the accomplishment of a specific tasks,
- Here in this example we have created different methods for calculating areas of different geometries.

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 and code in the form of methods. In object-oriented programming, computer programs are designed using the concept of objects that interact with the real world.
- Object-oriented programming languages are various but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.
- Languages used in Object-Oriented Programming: Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, Scala.

Object-Oriented Programming



- In human analogy, human is the class.
- Object is the name of the human, which can be multiple inside human class.
- Properties can be email id, address, phone number which describe the object.
- Methods can be perform some operations like to send an email.

Object-Oriented Programming

```
public class Square implements Shape {
    private Point topLeft;
    private double side;

    public double area() {
        return side * side;
    }
}

public class Rectangle implements Shape {
    private Point topLeft;
    private double height;
    private double width;

    public double area() {
        return height * width;
    }
}
```

Continue->>

Object-Oriented Programming

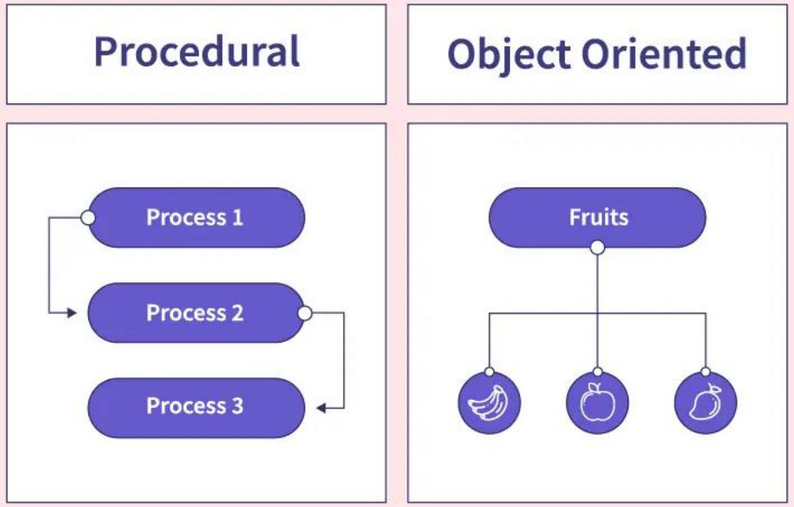
```
public class Circle implements Shape {
    private Point topLeft;
    private double radius;

    public double area() {
        return PI * radius * radius;
    }
}

public class Geometry {
    public final double PI = 3.14159265358973;
    public double area(Object shape) throws NoSuchShapeException {
        if (shape instanceof Square) {
            Square s = (Square) shape;
            return s.area();
        } else if (shape instanceof Circle) {
            Rectangle r = (Rectangle) shape;
            return r.area();
        } else if (shape instanceof Rectangle) {
            Circle c = (Circle) shape;
            return c.area();
        }
        throw new NoSuchShapeException();
    }
}
```

The concepts that we can take it out from here are –

- It must abstract the data concepts into modular units.
- It must have some polymorphic way to execute code.
- It must at least partially encapsulate that code and functionality.

Difference between POP & OOP

Procedural- The program is divided into functions.



Object-oriented-The program is divided into classes and objects.

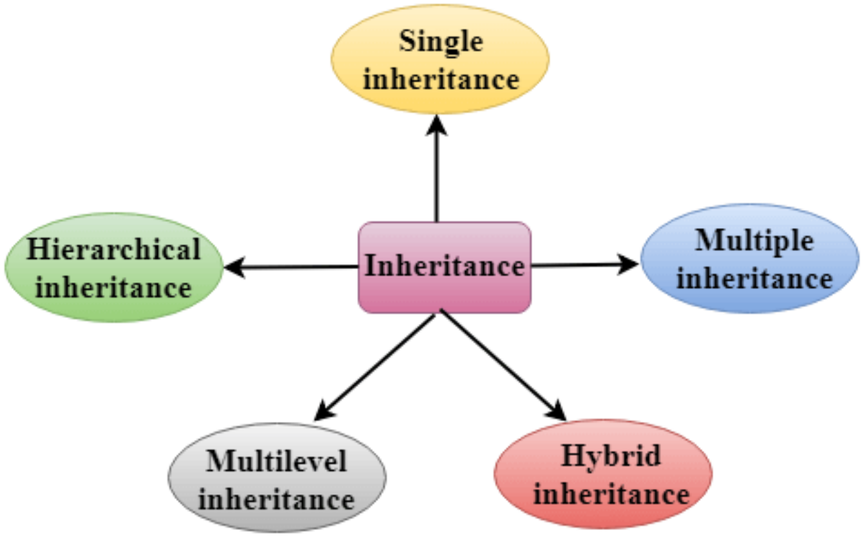
Difference between POP & OOP

Procedural Programming Language	Object Oriented Programming Language
1. Program is divided into functions.	1. Program is divide into classes and objects..
2. The emphasis is on doing things.	2. The emphasis on data.
3. Poor modeling to real world problems.	3. Strong modeling to real world problems.
4. It is not easy to maintain project if it is too complex.	4. It is easy to maintain project even if it is too complex.
5. Provides poor data security.	5. Provides strong data Security.
6. It is not extensible programming language.	6. It is highly extensible programming language.
7. Productivity is low.	7. Productivity is high.
8. Do not provide any support for new data types.	8. Provide support to new Data types.
9. Unit of programming is function.	9. Unit of programming is class.
10. Ex. Pascal , C , Basic , Fortran.	10. Ex. C++ , Java , Oracle.

Principles of OOP

1) Inheritance: When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability.

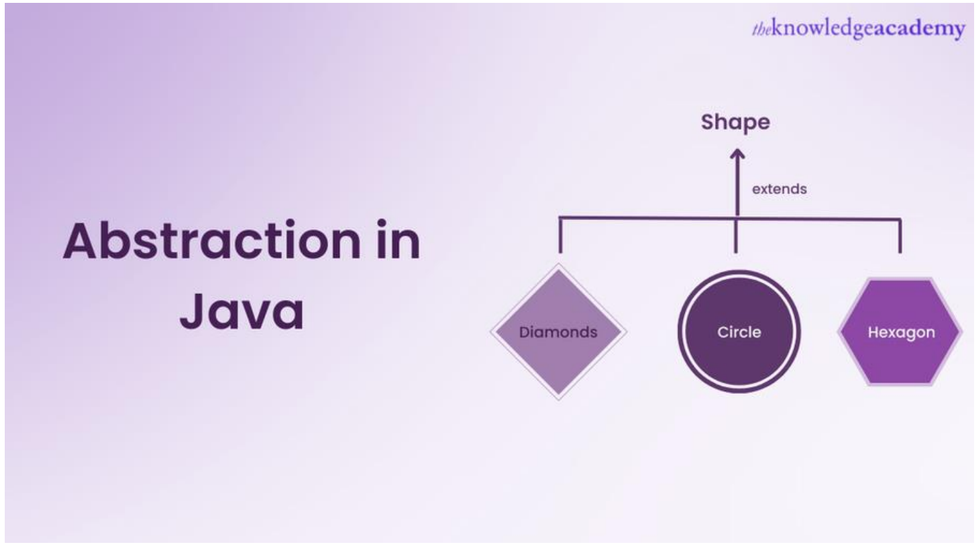
2) Polymorphism: When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc. In java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.



Principles of OOP

3) Abstraction: Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

4) Encapsulation: Binding (or wrapping) code and data together into a single unit is known as encapsulation. 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.



Benefits of OOPs

Benefits of OOP include:

- **Modularity.** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
- **Reusability.** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
- **Productivity.** Programmers can construct new programs quicker through the use of multiple libraries and reusable code.

Benefits of OOPs

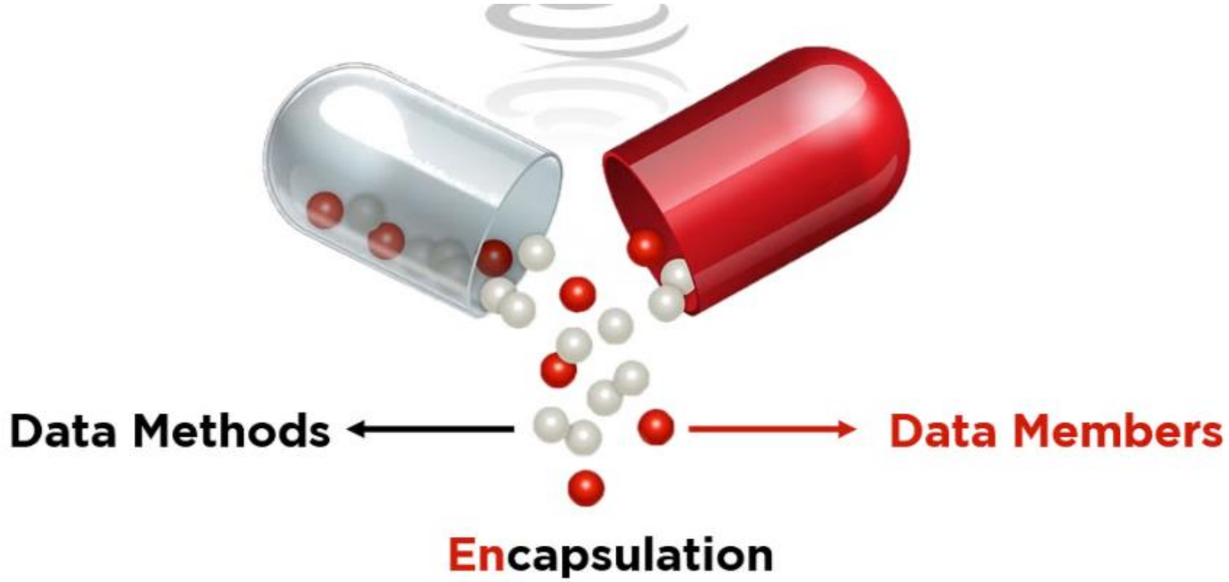
- **Easily upgradable and scalable.** Programmers can implement system functionalities independently.
- **Interface descriptions.** Descriptions of external systems are simple, due to message passing techniques that are used for objects communication.
- **Security.** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and internet protocols are protected.
- **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

Applications of OOP

- User interface design such as windows, menu.
- Real-Time Systems
- Simulation and Modeling
- Object-oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation systems etc.

Encapsulation

Encapsulation in Java refers to integrating data (variables) and code (methods) into a single unit. In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found.

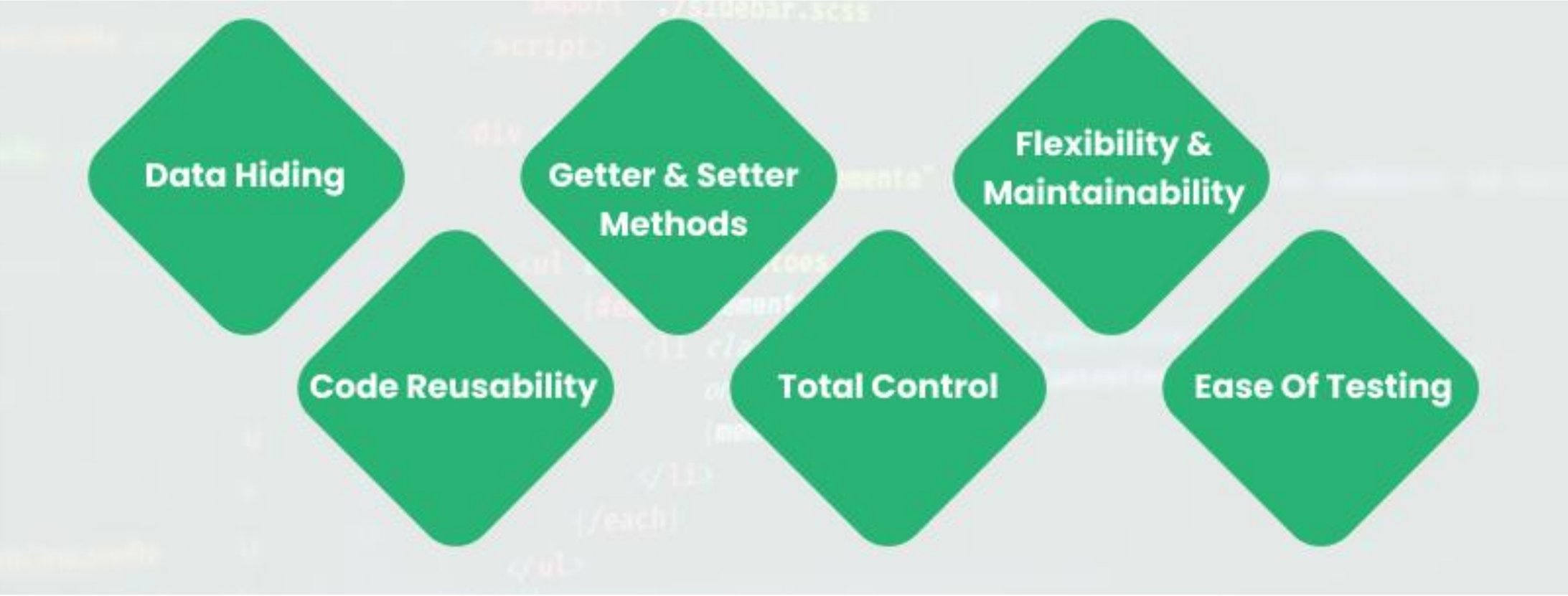


Need for Encapsulation

Need for Encapsulation

- **Better Control:** Encapsulation provides ultimate control over the data members and data methods inside the class.
- **Getter and Setter:** The standard IDEs provide in-built support for ‘Getter and Setter’ methods, which increases the programming pace.
- **Security:** Encapsulation prevents access to data members and data methods by any external classes. The encapsulation process improves the security of the encapsulated data.
- **Flexibility:** Changes made to one part of the code can be successfully implemented without affecting any other part of the code.

Benefits of Encapsulation



Benefits of Encapsulation

- A class can have complete control over its data members and data methods.
- The class will maintain its data members and methods as read-only.
- Data hiding prevents the user from the complex implementations in the code.
- The variables of the class can be read-only or write-only as per the programmer's requirement.
- Encapsulation in Java provides an option of code-reusability.
- Using encapsulation will help in making changes to an existing code quickly.
- Unit testing a code designed using encapsulation is elementary.
- Standard IDEs have the support of getters and setters; this makes coding even faster.

Encapsulation Example

- In the alongside example, we have created a class named Area. The main purpose of this class is to calculate the area.

- To calculate an area, we need two variables: length and breadth and a method: getArea(). Hence, we bundled these fields and methods inside a single class.
- Here, the fields and methods can be accessed from other classes as well. Hence, this is not data hiding.

- This is **only encapsulation**. We are just keeping similar codes together.

```
class Area {

  // fields to calculate area
  int length;
  int breadth;

  // constructor to initialize values
  Area(int length, int breadth) {
    this.length = length;
    this.breadth = breadth;
  }

  // method to calculate area
  public void getArea() {
    int area = length * breadth;
    System.out.println("Area: " + area);
  }
}

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

    // create object of Area
    // pass value of length and breadth
    Area rectangle = new Area(5, 6);
    rectangle.getArea();
  }
}
```

Output

Area: 30

Data Hiding in Java

- Data hiding is a procedure done to avoid access to the data members and data methods and their logical implementation.
- Data hiding can be done by using the access specifiers. We have four access specifiers, which are as follows.

Data Hiding in Java

- **Default**: Default is the first line of data hiding. If any class in Java is not mentioned with an access specifier, then the compiler will set ‘default’ as the access specifier. The access specifications of default are extremely similar to that of the public access specifier.
- **Public:** The public access specifier provides the access specifications to a class so that it can be accessed from anywhere within the program.
- **Private:** The private access specifier provides access to the data members, and the data methods limit to the class itself.
- **Protected:** The protected access specifier protects the class methods and members similar to the private access specifier. The main difference is that the access is limited to the entire package, unlike only a class with the private access specifier.

Data Hiding Example

- In the alongside example, we have a private field age. Since it is private, it cannot be accessed from outside the class. In order to access age, we have used public methods: getAge() and setAge(). These methods are called getter and setter methods.

- Making age private allowed us to restrict unauthorized access from outside the class. This is data hiding.

- *If we try to access the age field from the Main class, we will get an error.*
  // error: age has private access in Person
  p1.age = 24;

```java
class Person {

  // private field
  private int age;

  // getter method
  public int getAge() {
    return age;
  }

  // setter method
  public void setAge(int age) {
    this.age = age;
  }
}

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

    // create an object of Person
    Person p1 = new Person();

    // change age using setter
    p1.setAge(24);

    // access age using getter
    System.out.println("My age is " + p1.getAge());
  }
}
```

Output

My age is 24

Java Static Keyword

- Java's static keyword becomes important when we want to define a class member that will be used independently at class level.

- When a class member is declared static it can be accessed without creating any objects of its class.

- Both member methods and fields (variables) can be declared static. The most common example of a static member is Java's main() method.

- The main() method is declared as static because it must be called before any objects exist.

Java Static Data Members

- Data members or fields of a Java class declared static are called class members. There exists **exactly only one copy** of static fields or class members no matter how many objects of the class are finally created.

- A static field, also called a class variable comes into existence when the Java class is initialized. Data members declared as static are essentially global variables. When objects of its class are created they share the same copy of static field.

Java Static Data Members

```
class StaticField
{
    static int objectCount = 0;
    public StaticField()
    {
        objectCount++;
    }
    public String toString()
    {
        return new String ("There are " + objectCount + " objects of class " + this.getClass().getName());
    }
    public static String numOfObjects()
    {
        return new String ("There are " + objectCount + " objects of class " + StaticField.class.getName());
    }
}

class StaticFieldDemo
{
    public static void main(String[] args)
    {
        System.out.println(StaticField.numOfObjects());

        StaticField s1 = new StaticField();
        StaticField s2 = new StaticField();
        StaticField s3 = new StaticField();

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

Java Static Data Members

Output:

```
There are 0 objects of class StaticField
There are 3 objects of class StaticField
There are 3 objects of class StaticField
There are 3 objects of class StaticField
```

Java Static Methods

Java static methods are used to implement class behaviors that are not affected by the state of any instances. Java static methods are also implemented to access and mutate static members of the class in case they are private and cannot be accessed publicly. Java static methods have many restrictions over themselves:

- They can only call static methods within from them.
- Java static methods can only access static data members or fields.
- Java static methods cannot refer this and super

Java Static Methods

- A Java static method is always invoked without reference to a particular object. There will be a compile-time error if you try to invoke static method using the keyword this or the keyword super.

- A Java static method cannot access instance variables directly. Pay attention to this statement it looks tricky. See the following code and you would understand what the first statement of this paragraph exactly says.

Java Static Methods

```
class StaticMethod
{
    int instVar = 10;
    static int staticVar = 20;

    public static void staticMethod ()
    {
        //You can not access instVar directly.
        System.out.println(instVar); //error: Cannot make a static reference to the non-static field instVar

        System.out.println(staticVar);

        StaticMethod smObject = new StaticMethod();
        System.out.println(smObject.instVar);
    }
}


class StaticMethodDemo
{
    public static void main(String[] args)
    {
        StaticMethod.staticMethod();
    }
}
```

Java Static Methods

Output:

In the above program carefully along with in line comments and you will see that instance variable instVar cannot be accessed directly within from the method staticMethod(), and there will be a compile-time error.

This piece of code will not compile until either you remove the statement System.out.println(instVar); from staticMethod() or comment it.

It is very important to note that static members cannot be accessed by this and super keywords within from a Java static method, because keyword this represents the current instance and super represents the immediate parent class's instance in inheritance. Rule of thumb is this and super never come in static method. On the contrary, within from an instance method you can access static fields by using this and super.

Constructor

A constructor in Java is similar to a method that is invoked when an object of the class is created. Unlike Java Methods, a constructor has the same name as that of the class and does not have any return type. For example,

```
class Test {
  Test() {
    // constructor body
  }
}
```

Here, Test() is a constructor. It has the same name as that of the class and doesn't have a return type.

Constructor

```java
class Main {
  private String name;

  // constructor
  Main() {
    System.out.println("Constructor Called:");
    name = "PSOOP";
  }

  public static void main(String[] args) {

    // constructor is invoked while
    // creating an object of the Main class
    Main obj = new Main();
    System.out.println("The name is " + obj.name);
  }
}
```

input

```
Constructor Called:
The name is PSOOP


...Program finished with exit code 0
Press ENTER to exit console.
```

Types of Constructors

In Java, constructors can be divided into 3 types:

- No-Arg Constructor
- Parameterized Constructor
- Default Constructor

Java No-Arg Constructor

Similar to methods, a Java constructor may or may not have any parameters (arguments). If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

```
private Constructor() {
   // body of the constructor
}
```

Java Parameterized Constructor

```
class Main {

  String languages;

  // constructor accepting single value
  Main(String lang) {
    languages = lang;
    System.out.println(languages + " Programming Language");
  }

  public static void main(String[] args) {

    // call constructor by passing a single value
    Main obj1 = new Main("Java");
    Main obj2 = new Main("Python");
    Main obj3 = new Main("C");
  }
}
```

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

Output:

```
Java Programming Language
Python Programming Language
C Programming Language
```

Java Default Constructor

```
class Main {

  int a;
  boolean b;

  public static void main(String[] args) {

    // A default constructor is called
    Main obj = new Main();

    System.out.println("Default Value:");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
  }
}
```

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

Output:

```
Default Value:
a = 0
b = false
```

Constructor Important Points

- Constructors are invoked implicitly when you instantiate objects.
- The two rules for creating a constructor are:
  The name of the constructor should be the same as the class.
  A Java constructor must not have a return type.
- If a class doesn't have a constructor, the Java compiler automatically creates a default constructor during run-time. The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
- Constructor types:
  No-Arg Constructor - a constructor that does not accept any arguments
  Parameterized constructor - a constructor that accepts arguments
  Default Constructor - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- A constructor cannot be abstract or static or final.
- A constructor can be overloaded but can not be overridden.

Java Strings

In Java, a string is a sequence of characters. For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.
We use double quotes to represent a string in Java. For example,

```
// create a string
String type = "Java programming";
```

Here, we have created a string variable named type. The variable is initialized with the string Java Programming.

Java Strings

- Demonstrates various operations on a String.
- Finds the length, extracts a substring, concatenates strings, and converts to uppercase.

Output:

```
Length of the string: 12
Substring: Hello
Concatenated Message: Hello, Java! My name is John
Uppercase: HELLO, JAVA!
```

```java
public class StringExample {
    public static void main(String[] args) {
        // Creating a String
        String greeting = "Hello, Java!";

        // Finding the length of the String
        int length = greeting.length();
        System.out.println("Length of the string: " + length);

        // Extracting a substring
        String substring = greeting.substring(0, 5);
        System.out.println("Substring: " + substring);

        // Concatenating strings
        String name = "John";
        String message = greeting + " My name is " + name;
        System.out.println("Concatenated Message: " + message);

        // Converting to uppercase
        String uppercase = greeting.toUpperCase();
        System.out.println("Uppercase: " + uppercase);
    }
}
```

Java Arrays

An array is a collection of similar types of data.
For example, if we want to store the names of 100 people then we can create an array of the string type that can store 100 names.

```
String[] array = new String[100];
```

Here, the above array cannot store more than 100 names. The number of values in a Java array is always fixed.

Java Arrays [Declaration]

```
dataType[] arrayName;
```

- dataType - it can be primitive data types like int, char, double, byte, etc. or Java objects
- arrayName - it is an identifier

```
double[] data;
```

Java Arrays [Declaration]

```
// declare an array
double[] data;

// allocate memory
data = new double[10];
```

To define the number of elements that an array can hold, we have to allocate memory for the array in Java. Here, the array can store 10 elements. We can also say that the size or length of the array is 10. In Java, we can declare and allocate the memory of an array in one single statement. For example,

```
double[] data = new double[10];
```

Java Arrays [Inialization]

```
//declare and initialize and array
int[] age = {12, 4, 5, 2, 5};
```

Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).

Note:

- Array indices always start from 0. That is, the first element of an array is at index 0.
- If the size of an array is n, then the last element of the array will be at index n-1.

Java Arrays

- Illustrates the creation, initialization, and manipulation of an array of integers.
- Prints array elements and calculates the sum.

```java
public class Main {
    public static void main(String[] args) {
        // Initializing an array of integers directly
        int[] numbers = {1, 3, 5, 7, 9};

        // Accessing and printing array elements
        System.out.println("Array Elements:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Element at index " + i + ": " + numbers[i]);
        }

        // Calculating the sum of array elements
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        System.out.println("Sum of array elements: " + sum);
    }
}
```

Output->

```
input
Array Elements:
Element at index 0: 1
Element at index 1: 3
Element at index 2: 5
Element at index 3: 7
Element at index 4: 9
Sum of array elements: 25


...Program finished with exit code 0
Press ENTER to exit console.
```

Java Array Example

```java
import java.util.Scanner;

public class ArrayMaxFinder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        // Input array elements
        int[] numbers = new int[size];
        System.out.println("Enter the array elements:");
        for (int i = 0; i < size; i++) {
            System.out.print("Element at index " + i + ": ");
            numbers[i] = scanner.nextInt();
        }

        // Find the maximum element
        int max = findMax(numbers);

        // Output the result
        System.out.println("The maximum element in the array is: " + max);

        scanner.close();
    }
```

```java
    static int findMax(int[] arr) {
        if (arr.length == 0) {
            // Handle the case when the array is empty
            System.out.println("The array is empty.");
            return Integer.MIN_VALUE; // Minimum possible integer value
        }

        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        return max;
    }
}
```

- **The program takes the size of the array as input.**
- **It then takes user input for each element of the array.**
- **The findMax method is used to find the maximum element in the array.**

Java Array Example

Output: