Access Modifier: Defines the **access type** of the method i.e. from where it can be accessed in your application. In Java, there are 4 types of access specifiers:

- **public:** Accessible in all classes in your application.
- **protected:** The methods or data members declared as protected are accessible within the same package or subclasses in different packages.
- **private:** Accessible only within the class in which it is defined.
- **default (declared/defined without using any modifier):** Accessible within the same class and package within which its class is defined.

Pillars of OOPs

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
  - Compile-time polymorphism
  - Runtime polymorphism

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. Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times. This includes classes for objects occurring more than once in your code

An object is a basic unit of Object-Oriented Programming that represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. The objects are what perform your code, they are the part of your code visible to the viewer/user.

Pillar 1: Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or non-essential units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
The abstract method contains only method declaration but not implementation.

```java
abstract class GFG{
  //abstract methods declaration
  abstract void add();
  abstract void mul();
  abstract void div();
}
```

The class extending abstract can contain abstract methods.

Pillar 2: Encapsulation

It is defined as the wrapping up of data under a single unit. It is the mechanism that binds together the code and the data it manipulates. Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.

- So, the terms “encapsulation” and “data-hiding” are used interchangeably.

In order to achieve encapsulation in java follow certain steps as proposed below:

- Declare the variables as private
- Declare the setters and getters to set and get the variable values

```java
class Employee {
    private int empid;
      private String ename;
}
```

Pillar 3: Inheritance

It is the mechanism in Java by which one class is allowed to inherit the features (fields and methods) of another class. We are achieving inheritance by using **extends** keyword. Inheritance is also known as “**is-a**” relationship.

- **Superclass:** The class whose features are inherited is known as superclass (also known as base or parent class).
- **Subclass:** The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods

```
//base class or parent class or super class
class A{
  //parent class methods
  void method1(){}
  void method2(){}
}

//derived class or child class or base class
class B extends A{  //Inherits parent class methods
  //child class methods
  void method3(){}
  void method4(){}
}
```

There are 5 different types of inheritance in java as follows:

- **Single Inheritance:** Class B inherits Class A using extends keyword
- **Multilevel Inheritance:** Class C inherits class B and B inherits class A using extends keyword
- **Hierarchy Inheritance:** Class B and C inherits class A in hierarchy order using extends keyword
- **Multiple Inheritance:** Class C inherits Class A and B. Here A and B both are superclass and C is only one child class. Java is not supporting Multiple Inheritance, but we can implement using Interfaces.
- **Hybrid Inheritance:** Class D inherits class B and class C. Class B and C inherits A. Here same again Class D inherits two superclass, so Java is not supporting Hybrid Inheritance as well

Pillar 4: Polymorphism

It refers to the ability of object-oriented programming languages to differentiate between entities with the same name efficiently. The ability to appear in many forms is called **polymorphism**.

```
sleep(1000) //millis

sleep(1000,2000) //millis,nanos
```

Polymorphism in Java is mainly of 2 types:

1. *Overloading – methods with more parameters*
2. *Overriding - overriding methods of parents class*

There are two types of polymorphism as listed below:

1. Static or Compile-time Polymorphism
2. Dynamic or Run-time Polymorphism

Static or Compile-time Polymorphism when the compiler is able to determine the actual function, it's called **compile-time** polymorphism. Compile-time polymorphism can be achieved by **method overloading** in java. When different functions in a class have the same name but different signatures, it's called method overloading. A method signature contains the name and method arguments. So, overloaded methods have different arguments. The arguments might differ in the numbers or the type of arguments.

Dynamic or Run-time Polymorphism Dynamic (or run-time) polymorphism occurs when the compiler is not able to determine at compile-time which method (superclass or subclass) will be called. This decision is made at run-time. Run-time polymorphism is achieved through method overriding, which happens when a method in a subclass has the same name, return type, and parameters as a method in its superclass. When the superclass method is overridden in the subclass, it is called method overriding.

There are seven qualities to be satisfied for a programming language to be pure object-oriented. They are:

1. Encapsulation/Data Hiding
2. Inheritance
3. Polymorphism
4. Abstraction
5. All predefined types are objects
6. All user defined types are objects
7. All operations performed on objects must be only through methods exposed at the objects.

Java supports properties 1, 2, 3, 4 and 6 but fails to support properties 5 and 7

The static keyword: When we declare a class as static, then it can be used without the use of an object in Java. If we are using static function or static variable then we can't call that function or variable by using dot(.)

Test t = new Test();

Naming Conventions in Java

- **Class:** If you are naming any class then it should be a noun and so should be named as per the goal to be achieved in the program
- **Interface:** If you are naming an interface, it should look like an adjective such as consider the existing ones: Runnable, Serializable
- **Methods:** Now if we do look closer a method is supposed to do something that it does contains in its body henceforth it should be a verb.
-

Syntax of Method

```
<access_modifier> <return_type> <method_name>( list_of_parameters)
{
    //body
}
```

Advantage of Method

- Code Reusability
- Code Optimization

Java Constructors

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

```
class Geek
{
    .......
    // A Constructor
    Geek() {
    }
    .......
}
```

The constructor(s) of a class must have the same name as the class name in which it resides.

- Default Constructor -Geek()
- Parameterized Constructor – Geek(int a , int b)
- Copy Constructor – Geek(Geek obj)