***@ Programming paradigm***

A **programming paradigm** defines the methodology of designing and implementing programs using the key features and other building blocks of a programming language.  
  
A **programming paradigm** gives you an idea how problems are generally analysed and solved in a particular programming language.  
  
Below are a few commonly used programming paradigms :

1. Procedural Programming
2. Object Based Programming
3. Object Oriented Programming

**# Procedural programming** lays more emphasis on **procedure** than **data**. **C** language uses procedural programming style.  
  
**C++** and **Java** can also be used as a procedural programming language with some enhanced features such as **type checking**, **reference variables**, **inline functions**, **default arguments** etc.  
  
Below are a few **pros** and **cons** of procedural programming languages:  
**Pros**

1. For smaller programs the code is usually small, more readable and result oriented.
2. They are usually very efficient since they are written for a very specific purpose.

**Cons**

1. Since programs are function based, they do not relate to a real-life object.
2. When large amount of code is written inside functions, it becomes very difficult to maintain or make modifications.
3. Changing the way data is stored can required large amount of changes in code, unlike in Object oriented programs, making it difficult to maintain large systems.

# **Object based programming** is a paradigm that implements some features of object-oriented programming but not all.

In object based programming, data and its associated meaningful functions are enclosed in one single entity called class.  
  
**Classes** enforce information hiding and abstraction thereby separating the implementation details.  
  
In object based programming, whenever there is any **change** in the definition of type, user's interface remains unaffected generally.

The object based programming can be thought of as a **subset** of object oriented programming as it implements some of the features implemented by object oriented programming like **information hiding**, **abstraction**, **classes**, **function overloading**, etc.

The advantages of object based programming are:

* it overcomes most shortcomings of procedural programming
* it localizes changes and hides implementation details from user
* it supports user-defined types
* implements information hiding and abstraction

However, object based programming suffers from one major **limitation** and that is its inability to represent real world relationships that exist among object.

**Object oriented programming** paradigm is a superset of object based programming. It offers all the features of object-based programming and overcomes its limitation by implementing **inheritance**.

@ Java ***OOPs***

The **object oriented programming (OOP)** approach views a problem in terms of **objects** or **entities** involved rather than the procedures to be implemented.

* In an object-oriented programming language like Java, a class acts as a type, a **template** a blueprint, from which instances of that type can be created.
* A class can contain members such as fields, methods, nested classes, interfaces, instance, static initializers and constructors.
* While programming using **OOP** approach, the **characteristics** of an object are represented by its **data** and its **behaviour** is represented by its **functions**.
* Therefore, in object oriented programming object represents an entity that can store **data** and has its interface with external world through **functions**.
* The object oriented programming evolved to overcome the **limitations** of procedural programming approach.

The fundamental concepts in **OOP** are:

* Data Abstraction
* Data Encapsulation
* Modularity
* Inheritance
* Polymorphism

Topics in OOPs :- class, objects & methods, Code Reusability[ inheritance, Composition, Polymorphism ], Security[ Abstraction, Data Hiding, Encapsulation, Tightly coupled class].

* Data abstraction refers to the act of representing essential features without including the background details and explanations.

Ex: Let us consider an example of **switch board**, you can only press certain switches according to your requirement. What is happening inside, how it is happening, you need not know.  
  
This is **abstraction**, you know only essential things to operate on **switch board** without knowing the background details of switch board.

* Data encapsulation is the way of combining both data and functions that operate on the data under a single unit.

In this, the **data** is not accessed to the outside world directly and only through member functions, the data can be accessed.

The insulation of data from direct access by the program is called **data hiding**.

Abstraction and encapsulation are complementary concepts, where as **abstraction** focuses upon the observable behaviour of an object and **encapsulation** focuses upon the implementation that gives rise to this behaviour.

**Encapsulation** is most often achieved through **information hiding**, which is the process of hiding all the secrets of an object that do not contribute its essential characteristics, the structure of an object is hidden as well as the implementation of its methods.

# Constructor

* Default Constructor / No-argument Constructor

All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

*Default constructor provides the default values to the object like 0, null, etc. depending on the type.*

* A constructor is a special method (block of code) which is used to create instances of a class.
* A constructor is also used to initialize the fields of a class
* A constructor's name **must** always **be the same** as the class name
* A constructor is a special method which does not have a **return type.**
* we usage of this to differentiate the identifiers with same names in different scopes.
* A constructor must be written inside the class body.
* User Defined Constructor/Parameterized Constructor: Constructors can also take parameters, which is used to initialize attributes.

***Remember: Does constructor return any value?***

*There are no “return value” statements in the constructor, but the constructor returns the current class instance. We can write ‘return’ inside a constructor.*

# Methods

In Java, a method (also called as function) is a named block of code that performs a task.

There are 5 basic parts to a method:

1. Method Name - the name used identify and invoke/execute the method.
2. Method Body - contains the code statements which will be executed when the method is called/invoked.
3. Method Parameters - [***optional***] contain the input values which are used by the statements in the method body.
4. Method Return Type - indicates what **type** of value is returned, if the method returns nothing, then void is used as the return type.
5. Modifiers - [***optional***] modifiers constitute one or more keywords that inform how the method can be used. We will learn more about them later.

toString()

* toString() is a special method used to convert an object's state information into a human readable text.

The toString() method does not take any parameters and will always return a String.

The toString() method like the constructor of a class is part of the class body. It must be written inside a class scope (inside the opening and closing brace of the class).

Accessor Methods

The state information of an instance of class is stored in its instance fields. These fields are usually marked as **private.**

* These fields are initialized using the constructor. Hence, after the values are initialized, getXXX and setXXX methods are written to retrieve and modify their values.
* A setter method will always have void as return type and takes the field's type as an argument into the method.
* A getter method of a field will always have the field's type as the return type and will not have any argument.
* These getter and setter methods are also called as accessor methods.
* Note how the getter and setter method names follow [camel-case](http://en.wikipedia.org/wiki/CamelCase) naming convention.

# Packages

* java.util - this package contains utility and data structure related classes like Date, ArrayList, HashSet, etc., which we will learn later.
* java.io - this package contains classes which help in reading input and writing output to files and streams.
* java.net - this package contains networking related classes.
* **Note:** java.lang package which contains all commonly used Java classes is by default imported. Which means for using a String or an Integer or the System class and the like, one need not import the java.lang package.

# Access Control Modifiers

The four different access control modifiers are:

* **private** - marks the member as accessible only in the declaring class
* **package/default** - marks the member as accessible to classes in the same package
* **protected** - marks the members accessible in subclasses of the class and to classes in the same package
* **public** - marks the members as accessible to all classes

# Strings

* An instance of String is immutable. Meaning, once a string object is created in memory, the contents of the string cannot be changed.  However, str can be made to point to some other string object
* the correct way of comparing contents of two strings is to use equals method.
* The way to escape a double quote is by providing the escape character \ before the double quote.

# String Methods

* length() :
* charAt() :
* toLowerCase() :
* toUperCase() :
* startsWith() :
* endsWith() :
* equals() :
* equalsIgnoreCase() :
* compareTo() : compare string lexicographical.
* indexOf() :
* lastIndexOf() :
* substring() :
* split()
* trim():
* rreplace():

# ERROR

* Whenever on compilation, you see an error saying: xyz cannot be resolved, it means that the compiler is unable to resolve the symbol xyz to a type.; On such errors first verify if the symbol xyz is spelt correctly.