MONOGRAPH

FUNCTIONAL PROGRAMMING LANGUAGES

By Mariya Khurshid

INTRODUCTION:

In computer science, **functional programming** is a **programming** paradigm where programs are constructed by applying and composing functions. When a pure function is called with some given arguments, it will always return the same result, and cannot be affected by any mutable state or other side effects.

Functional programming is a programming paradigm in which we try to bind everything in pure mathematical functions style. It is a declarative type of programming style. Its main focus is on "what to solve" in contrast to an imperative style where the main focus is "how to solve". It uses expressions instead of statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables.

Functional programming languages are specially designed to handle symbolic computation and list processing applications. **Functional programming** is based on mathematical functions. Some of the popular **functional programming** languages include: Lisp, Python, Erlang, Haskell, Clojure, etc.

ADVANTAGES OF FUNCTIONAL PROGRAMMING LANGUAGES:

- Pure functions are easier to reason about.
- Testing is easier, and pure functions lend themselves well to techniques like property-based testing.
- Debugging is easier.
- Programs are more bulletproof.
- Programs are written at a higher level, and are therefore easier to comprehend.

USES OF FUNCTIONAL PROGRAMMING LANGUAGES:

- Some tasks with complex math.
- Tasks related to language processing both native languages and formal languages.
- Concurrency and parallelism.
- Analysis and building complex data structures.

FUNCTIONAL PROGRAMMING IS BASED ON LAMBDA CALCULUS: Lambda calculus is framework developed by Alonzo Church to study computations with functions. It can be called as the smallest programming language of the world. It gives the definition of what is computable. Anything that can be computed by lambda calculus is computable. It is equivalent to Turing machine in its ability to compute. It provides a theoretical framework for describing functions and their evaluation. It forms the basis of almost all current functional programming languages. Fact: Alan Turing was a student of Alonzo Church who created Turing machine which laid the foundation of imperative programming style.

Programming Languages that support functional programming:

Haskell, JavaScript, Scala, Erlang, Lisp, ML, Clojure, OCaml, Common Lisp, Racket.

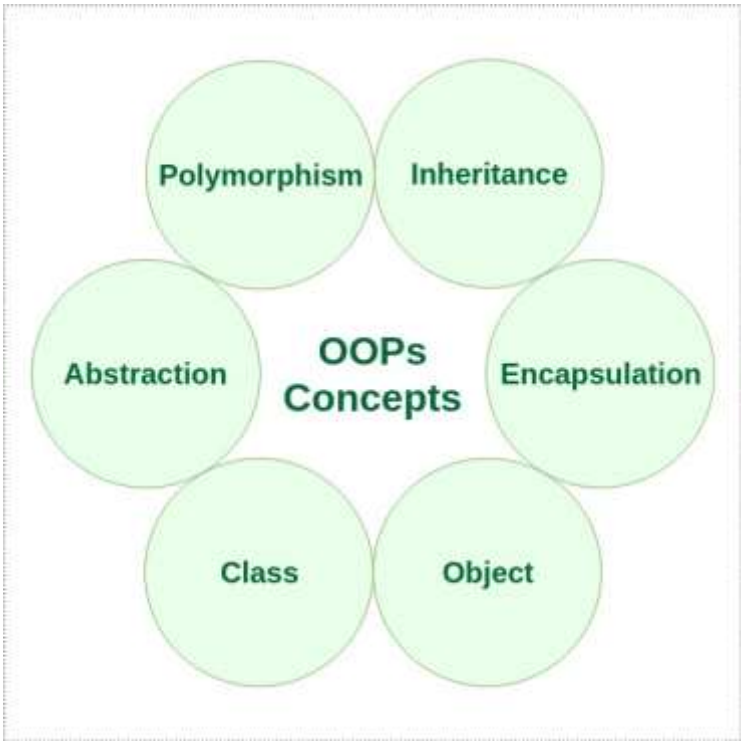
OOPS CONCEPT

INTRODUCTION:

Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOPs Concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Class
- Object



1.<u>Polymorphism</u>: Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently.

1. Polymorphism in Java are mainly of 2 types:
    - Overloading in Java
    - Overriding in Java
2. **Inheritance:** Inheritance is an important pillar of OOP(Object Oriented Programming). 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.

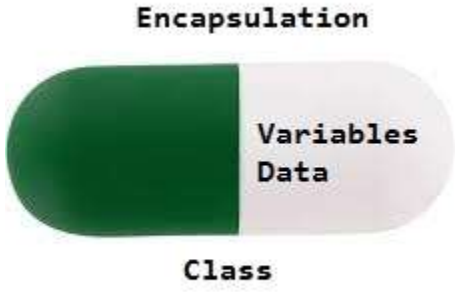
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is **extends**.

Syntax:

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

3. **Encapsulation:** Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.
   - Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
   - As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
   - 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.



4. **Abstraction:** Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
   Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

   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.

5. **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 (Refer this for details).
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, { }.

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:

0. **State** : It is represented by attributes of an object. It also reflects the properties of an object.
1. **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
2. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog