**Programming Paradigms:** Programming languages based on their features.

**Common Paradigms**:

**Disadvantages of Object-Oriented Programming:**

* Complex Design
* Slow in performance
* Very High Learning Curve for Programmer
* Have side effects, One Object behavior can affect Another Objects State.
* Because of Side Effects, Debugging becomes difficult.
* Parallel Programming is difficult.

The basic element of Functional Programming is Variable and Functions. Where as in OOP it’s object and methods. In functional programming we deal with functions.

**Advantages of Functional Programming:**

* No Side Effects
* Easy to Understand Code. In short, Clean Code
* Learning curve is low as compare to OOP
* Debugging is easy
* Parallel Programming in Easier
* Testing becomes easier
* Programs are bullet proof: Because code is less complex as it has pure little-little functions
* Functions are First Class Citizens

Before Java 8:

* We can only store a variable/object to another variable:

**int** number = 7;

**Integer** value = number;

* We can only return variable and pass variable from/in a method:

**public** **Integer** add(**Integer** num1, **Integer** num2) {

**Integer** result = num1 + num2;

**return** result;

}

In Java 8:

* The Java Team introduce Functional Programming in Java Environment, Because of the countless benefits of functional Programming.
* Hence, to achieve functional programming in Java, they have made methods as a first-class citizen in Java 8. Means, now we can assign a method to a variable, we can pass it to another method and we can also return it from a method.
* To achieve Functional Programming, Java 8 introduced Lambda Expression.

**Lambda Expression:**

In the world of programming the word “Lambda” refers to an anonymous function, Function with no name.

To work with Lambda Functions, we need to understand something called as Functional Interface.

**Functional Interface:** A interface with just one abstract method.

To Understand Let’s create our own Functional Interface.

@FunctionalInterface

**interface** Calculator {

**public** **int** calculate(**int** num1, **int** num2);

**default** **void** defaultMethod() {

String msg = "Can have default method";

}

**static** **void** staticMethod() {

String msg = "Can have static method";

}

}

/\*\*

\* Declaring an anonymous function/Lambda Expression and

\* Storing it like a variable.

\*/

Calculator add = (i,j) -> {

**return** i+j;

};

Calculator multiply = (i,j) -> {

**return** i\*j;

};

/\*\*

\* Accepting an Anonymous Function as Functional Interface

\* with two other int variables

\* **@param** calc

\* **@param** num1

\* **@param** num2

\*/

**public** **static** Calculator showResult(Calculator calc, **int** num1, **int** num2) {

System.***out***.println(calc.calculate(num1, num2));

/\*\*

\* Returning a anonymous function as an Functional Interface

\*/

**return** calc;

}

/\*\*

\* Passing a function to a method just

\* like a variable to do the operation.

\*/

*showResult*(add, 4, 3);

*showResult*(multiply, 7, 3);