A comparative Study on OOPS Vs FP

In this Article, we will discuss the comparison between Object Oriented Programming and Functional programming. Here I will not discuss which style is good or you should adopt this style blah blah.Each style has it advantages and It is purely based on the situation which to use when.

But I am interested to see the reader's view on the comparative study.

Let's start the comparison on some common aspects of programming.

1. **Behaviour**:

***What is programming?***

**“Get the result based on an action acted upon a data”**

So according to the definition, we can extract two fundamental parts.

a.Data.

b.Action/behavior.

Every programming languages based on data and behavior. So the essence of programing is ***How a behavior/action is acting on a data****,* all the different styles used in FP and OOPS are discovered to maintain How judiciously action can be taken on data to return the desired result.

**OOPS** : In Object oriented programming we bind the data and behavior together, we make them an unit . By doing this we make a logical boundary on data and behavior. We call that boundary as a class, data as properties and behaviors as methods. The main advantage of this technique is , as we encapsulate data and behavior as a single unit. So We can easily control which behavior or data will be exposed to outside world , the simple term other class using access specifiers. Having said that a question may be popped up in our mind, what will happen data/properties will change by the caller/client program. Here the concept of State is coming, when an Object is created it holds certain value for each property(can be default) as a whole Object is a State representation. So if any property changes it change its state and we call it Mutable. OOPS handles mutability so we can think OOPS as a stateful programming. As OOPS deals with State and Object are mutable so in a Complex scenario it is very hard to maintain an Object's state , in Multithreaded environment where every thread try to change Object state creates more complex problems and leads to dirty state or unintentional change of the state, To get rid of that Different techniques like Lock,Synchronize,Immutability,Defensive copy etc are created.

**FP**: Functional Programming is exactly opposite to the OOPS it’s segregate the behavior and data. It is work in a Declarative style. Behaviour is declared as function and data come as an input of that function and function is applied on the data and produces a new Set of data. So functions are work like an engine who takes input and produces an output. The function can be in Pipeline One functions output can be the input to another function. But it works in Map techniques where input state is not changed but if any data on input is needed to modify by the function it creates a duplicate data structure like input and put the changed data there. So Input state is never changed. So Pure functional programming language is stateless, and data are immutable. So in a case of Multi-Threading environment, it works fine as they are immutable so inherently thread safe. As Functional programming does not maintain state so for a stateful application it is complex to use it has to pass data structure every time as an input so functions can put the modified state in a new data structure which is the duplicate of passing structure and return the mimic structure.

**Example**: Say a functional program need to add 10 to each element in an array, then it will pass the array in a function, looping through it add 10 to each element store it in a new array return the duplicate new array.

Whereas in an OOPs we create a class which has property call array and a method call addTen() , inside the method, it loops through the array add 10 to each element and reassign it to the array property.

Code :

FP Style:

public int[] addTenToElement(int[] arr)

{

int result[] = new int[arr.length];

int index=0;

for(int i : arr)

{

result[index]= i+10;

index++;

}

return result;

}

public void print(int[] arr)

{

for(int i : arr)

{

System.out.print(i + ",");

}

System.out.println();

}

OOP’s Style

package com.example.basic;

public class OOPStyle {

private int[] arr;

public int[] addTen() {

int index = 0;

for (int i : arr) {

arr[index] = i + 10;

index++;

}

return arr;

}

public void print(int[] arr) {

for (int i : arr) {

System.out.print(i + ",");

}

System.out.println();

}

public int[] getArr() {

return arr;

}

public void setArr(int[] arr) {

this.arr = arr;

}

public static void main(String[] args) {

OOPStyle op = new OOPStyle();

int[] arr = { 1, 2, 3, 4 };

op.setArr(arr);

int[] res = op.addTen();

System.out.println("Old Value has been changed");

op.print(arr);

System.out.println("new Value");

op.print(res);

}

}

**2. Usage :** In this section, we will discuss which to use when. From Very top level if we look at business needs it can be two types

1. Business wants to add new functionality.
2. Business wants to increase its support.

**OOP :** As OOPs supports four things encapsulation,Polymorphism ,Inheritance and Abstraction so by the help of that we can hide the data and behavior from outside world, by encapsulation we can manage which data and behavior are exposed to the client and By Polymorphism we can segregate the definition from its implementation. So generally we expose an interface with set of methods to the client, so-called “API” so the client knows which operation it can perform on modules but does not aware about actual implementation or how the things get done, we called it Abstraction. So any Data/Object which match API’s operations can be added easily , just to create a new class and provide the implementation. But Client does not know anything about that implementation. So a business which wants to support different specifications of a Data structures can use OOPS. on another hand If business wants various functionality on limited set of data structure ,OOPs is not fruitful as to add the new function you have to alter the Data Structure as data and behavior clubbed in a class often breaking Open/Closed responsibility , can lead to breaking Single responsibility. To avoid that we use different patterns like Skeletal,Bridge etc.

**FP** : AS Functional programming supports passing function as an argument of another function we can reuse a function also we can create a complex function using reusable basic function so it is very easy to create a functional library for future use. As we can use the function as a pipeline. A function takes another function called higher order functions. So If Business requirements are adding new functionality Functional programming is a good fit. But on another hand if the Business requirement is added new specific data structure then in Functional programming is tough as it needs to change the function implementation as a function takes data structure as input and acted on them so change or add in Data structure into the system means the change in function implementation.

Having said that Business actually needs both add new function and support many data structure so then the Architect should do a comparative study between FP and OOPs based on requirements or can choose such language which serves both Hybrid languages like java8 , scala etc.

**3. Data Structure :**

**OOPs :** As OOPs support Class so we can bound different properties to it and also exposes behavior on it so by this we can create a complex rich data Structure with lots of inbuilt functions.So as a User/caller of OOPs programming we can use the rich data structure with inbuilt functions .

**FP:** FP segregates the behavior from data so in FP data means only records so it uses basic Data Structure like an array , hashtable but for the rich structure, you have to define it by own using this basic data structure.

**4. Function:**

**OOPs: In** OOPs we called function as methods a method is same like functional programming's function but it has two differences.

1. It should be enclosed by a Class and can operate on the data members of the implicit instance of this class (this in java).
2. Method may accept data structure and act on it or it acting on implicit reference unlike FP, where in a function, we must have to pass a data structure to operate on.

So a method working on data member and change it's value so generally the method is not Idempotent in nature, so if we call same method repetitively we can get different results as it can mutate the state of instance. In OOPs methods is use to delegates the responsibility it is more like subroutine function.

Another problem of method in a pure OOps is ,it can’t pass method to another method as an argument so can not push behavior on the fly. As a result, if we need to perform the different type of operation the same data set we have to define the separate method for that. Although you can declare an Interface and then declare a method which takes this interface and then on the fly create Anonymous implementation and pass it as argument to the method and method uses that implementation to achieve dynamic operation on the same dataset.

**FP** : In Functional programming function are act as a mathematical function it takes an input and produces some output . The function is idempotent is same data set has been passed repetitively we can get the same result. In a function, you must have to pass an input and function works on that input this input can be a data or a function which provides an output which can act as input.

So in a mathematical term

f(x) takes input as x and if d(y) produces x, we can write

f(x) = f(d(y)).

Also, an important feature of functional programming is Currying ,Currying provides a way for working with functions that take multiple arguments, and using them in frameworks where functions might take only one argument.

So in a mathematical term

In add(x,y)=x+y;

And for value for x=2 and y =4;

We can rewrite function by replacing x first as Currying is a method for producing a chain of functions, each taking exactly one argument. The construction is achieved by "hiding" all but one argument in another, new, curried function, whose job it is to return functions of the remaining arguments. This is explicitly (but informally) illustrated next.

add(2,y) = 2+y as it is a now depends on the only y so it is a function of y

So I can rewrite

add(2,y)= addx(y) = 2+y; here x is a subscript denotes it compute x and return a new curried function addx which takes only y as an argument.

No replace y

add(2,4) = addx(4)= 2+4;

We got the same result if we replace x,y at a one go

The benefits of this are partial compute is if function takes multiple parameters and you don't pass all value still it computes a result. But in OOPs, you have to provide overloaded methods for that.

**Example :** Say a For a Calculator How we will implement the same in Functional and OOP.

I use java 8 in both cases for understanding purpose although java 8 uses functional style by lambda using Functional interface. But more or less we are achieving the functional style.

**FP:**

interface Operation{

public int operate(int a,int b);

}

public class Calculator{

public void calculate(int a,int b,Operation op)

{

int res = op.operate(a,b);

System.out.println(res);

}

public static void main(String args[])

{

Calculator cal = new Calculator();

cal.calculate(10,20,(x,y)->{return x\*y;});//pass method runtime

cal.calculate(10,20,(x,y)->{return x+y;});

}

}

**OOPs**

public class OOPsCalculator{

int operand1;

int operand2;

public void setOperand(int a,int b)

{

this.operand1 =a;

this.operand2 =b;

}

public int add()

{

return operand1+operand2;

}

public int multiply()

{

return operand1\*operand2;

}

public static void main(String args[])

{

OOPsCalculator cal = new OOPsCalculator();

cal.setOperand(10,20);

int add = cal.add();

System.out.println(add);

int mul = cal.multiply();

System.out.println(mul);

}

}

In both cases output is 30,200.

We can improve this solution by creating an anonymous class of Operation interface and pass the same to operate function.

**5. Composition**

**OOPs**: OOPs maintain composition using Inheritance and association , to be simple , IS-A and HAS-A, as OOPs binds data and behaviour in a single unit so in OOPs composition not only deals with data also with behaviours also so Composition has different shades here , to find what behaviour lies in which Object in a complex system often cause problem and if one Object depends on other then it depends on another the object graphs create lots of problems to manage different behavioural pattern are discovered for that. In a one word Composition in OOPs is not so easy.

**FP**: As in a functional program takes data and produce a new set of data and if that data is compatible with other functions input we can easily compose them. The functional program can use the pipeline of methods and can pass the function as an input of another function so the notion of a functional program is composition. So composition plays very well in Functional programming.

These are the basic comparison I can found if anyone found any other major difference I missed please put it on Comment section so I can update it.