

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



Advanced Programming (CO2039)

Report (Semester 202, Duration: 01 weeks)

OOP vs FP

Advisor: Mr. Lê Lam Sơn

Student Name: Nguyễn Hoàng

Student ID: 1952255

HO CHI MINH CITY, AUGUST 2021

1 OOP and FP in baking a pizza

OOP makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.

What? Alright that sounds a bit rough, let's rephrase this a bit. OOP aims to model the world in self-contained entities, and affects change by modifying the state of itself or other entities. FP on the other hand aims to not modify the original data, but rather creates new data given some existing data.

To demonstrate this, we will try to make a pizza. With OOP, a big box or object with all the materials to create a pizza is available, and the helper methods will slowly transform them into a complete pizza. FP will take a different approach, as materials are given to each stage/step/activity in order to be used in the next activity until the final product is achieved.

We will try to describe this pizza making progress programmatically using C++ and Haskell.

Let's start with a complete C++ program

```
1  #include <iostream>
2  #include <string>
3
4  class Pastry
5  {
6  public:
7      virtual void bake_me_baby()
8      {
9          prepare_dough();
10         add_sauce();
11         add_toppings();
12         bake();
13     }
14
15     protected:
16         virtual void prepare_dough() = 0;
17         virtual void add_sauce() = 0;
18         virtual void add_toppings() = 0;
19         virtual void bake() = 0;
20 };
21
22 class Pizza : public Pastry
23 {
24     protected:
25         int time = 0;
26         std::string state = "Raw";
27
28     protected:
29         void prepare_dough()
30         {
31             if (time != 0)
32                 return;
33             time = 1;
```

```
34     state = "Prepared dough";
35 }
36 void add_sauce()
37 {
38     if (time != 1)
39         return;
40     time = 2;
41     state = "Added sauce";
42 }
43 void add_toppings()
44 {
45     if (time != 2)
46         return;
47     time = 3;
48     state = "Added toppings";
49 }
50 void bake()
51 {
52     if (time != 3)
53         return;
54     time = 4;
55     state = "Baked the hell out of this";
56 }
57
58 public:
59     void bake_me_baby()
60     {
61         prepare_dough();
62         std::cout << time << " " << state << std::endl;
63         add_sauce();
64         std::cout << time << " " << state << std::endl;
65         add_toppings();
66         std::cout << time << " " << state << std::endl;
67         bake();
68         std::cout << time << " " << state << std::endl;
69     }
70 };
71
72 int main(int argc, char **argv)
73 {
74     Pastry *pizza = new Pizza();
75     pizza->bake_me_baby();
76     delete pizza;
77     return 0;
78 }
```

Output of this program

```
1 1 Prepared dough
2 2 Added sauce
3 3 Added toppings
4 4 Baked the hell out of this
```

Nice! Let's do this again, but with Haskell

```
1 module Main (main) where
2
3 data Pastry = Pizza {time :: Int, state :: String} deriving (Show)
4
5 prepareDough :: Pastry -> Pastry
6 prepareDough pizza@(Pizza t _)
7   | t /= 0 = pizza
8   | otherwise = pizza {time = 1, state = "Prepared dough"}
9
10 addSauce :: Pastry -> Pastry
11 addSauce pizza@(Pizza t _)
12   | t /= 1 = pizza
13   | otherwise = pizza {time = 2, state = "Added sauce"}
14
15 addToppings :: Pastry -> Pastry
16 addToppings pizza@(Pizza t _)
17   | t /= 2 = pizza
18   | otherwise = pizza {time = 3, state = "Added toppings"}
19
20 bake :: Pastry -> Pastry
21 bake pizza@(Pizza t _)
22   | t /= 3 = pizza
23   | otherwise = pizza {time = 4, state = "Baked the hell out of
24   this"}
25
26 bakeMeBaby :: Pastry -> IO ()
27 bakeMeBaby pizza = do
28   let pizza1 = prepareDough pizza
29   print pizza1
30   let pizza2 = addSauce pizza1
31   print pizza2
32   let pizza3 = addToppings pizza2
33   print pizza3
34   let pizza4 = bake pizza3
35   print pizza4
36
37 main :: IO ()
38 main = do
39   let pizza = Pizza 0 "Raw"
40   bakeMeBaby pizza
```

Output of this program



```
1 Pizza {time = 1, state = "Prepared dough"}  
2 Pizza {time = 2, state = "Added sauce"}  
3 Pizza {time = 3, state = "Added toppings"}  
4 Pizza {time = 4, state = "Baked the hell out of this"}
```



2 Conclusion

With the pizza making out of the way, there are definitely some things noticeable between the two approaches. Obviously, the procedure of the process does not change, but the way the materials or the pizza, otherwise known as the data, are handled and processed is different.

	OOP	FP
Pros	<p>OOP objects contains both the data (attributes) and things that it can do (methods)</p> <p>Any changes that are applied is reflected on the object itself</p> <p>OOP organize everything into hierarchies of abstract objects</p>	<p>FP decouples the data from the functions</p> <p>Data in FP is not intended to change, if it does, new data is just created</p> <p>FP is natural for the human brain, as in, our thought process is centered around “doing” things</p>
Cons	<p>OOP objects are abstract and potentially complex, wasting a lot of time in abstractions and design patterns instead of solving the problem</p> <p>OOP encourages sharing of mutable state and introduces additional complexity with its numerous design pattern, thus making common development practices, like refactoring and testing, needlessly hard</p> <p>Concurrency is basically impossible because the output of a method depends on the state of the object, unless we make the object immutable, which is FP</p>	<p>A good FP program requires strict state and I/O management, which requires brain-breaking abstraction inversion</p> <p>FP data objects are immutable, meaning that it is bad for simulating anything that involves objects that change like GUI</p> <p>FP has a strong mathematical foundation, meaning that it has a high barrier to entry, requiring you to learn concepts from abstract algebra to category theory</p>

You can find the full source code of this report on [my GitHub repository](#).

Easter egg

Congratulations! You have actually read my report. This section serves no more than empty space that I type when I can't get some thing out of my mind.

Yes, I am aware that this section creates the feeling that I am not taking this report seriously. Yes, I am aware that the comparison table is overkill and not very related to the pizza problem, but FP is not something anyone can learn in 14 weeks, not to add that I had to learn it myself during the course. That means the teacher did not help much.

During the one-week duration to write this report, I was told that someone posted this report [on Chegg](#). Congratulations to whoever created this assignment, I guess.

With that having been said, please enjoy this little abomination I created in Python.

```
1  class iostream:
2      def __lshift__(self, other):
3          print(other, end='')
4          return self
5
6      def __repr__(self):
7          return ''
8
9
10 if __name__ == "__main__":
11     cout = iostream()
12     endl = '\n'
13     cout << "Hello" << ", " << "World!" << endl
```