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 steps to make a pizza is available. FP takes a different approach, as it only contains the materials are. It doesn't care what you do with these materials as long as it is all what you need.

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

Let's start with a complete C++ program

```cpp
#include <iostream>
#include <string>

class Pastry
{
public:
  virtual void bake_me_baby()
  {
    prepare_dough();
    add_sauce();
    add_toppings();
    bake();
  }

protected:
  virtual void prepare_dough() = 0;
  virtual void add_sauce() = 0;
  virtual void add_toppings() = 0;
  virtual void bake() = 0;
};

class Pizza : public Pastry
{
protected:
  int time = 0;
  std::string state = "Raw";

protected:
  void prepare_dough()
  {
    if (time != 0)
      return;
    time = 1;
    state = "Prepared dough";
```

```
  }
  void add_sauce()
  {
    if (time != 1)
      return;
    time = 2;
    state = "Added sauce";
  }
  void add_toppings()
  {
    if (time != 2)
      return;
    time = 3;
    state = "Added toppings";
  }
  void bake()
  {
    if (time != 3)
      return;
    time = 4;
    state = "Baked the hell out of this";
  }

public:
  void bake_me_baby()
  {
    prepare_dough();
    std::cout << time << " " << state << std::endl;
    add_sauce();
    std::cout << time << " " << state << std::endl;
    add_toppings();
    std::cout << time << " " << state << std::endl;
    bake();
    std::cout << time << " " << state << std::endl;
  }
};

int main(int argc, char **argv)
{
  Pastry *pizza = new Pizza();
  pizza->bake_me_baby();
  delete pizza;
  return 0;
}
```

Output of this program

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

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

```haskell
module Main (main) where

data Pastry = Pizza {time :: Int, state :: String} deriving (Show)

prepareDough :: Pastry -> Pastry
prepareDough pizza@(Pizza t _)
  | t /= 0 = pizza
  | otherwise = pizza {time = 1, state = "Prepared dough"}

addSauce :: Pastry -> Pastry
addSauce pizza@(Pizza t _)
  | t /= 1 = pizza
  | otherwise = pizza {time = 2, state = "Added sauce"}

addToppings :: Pastry -> Pastry
addToppings pizza@(Pizza t _)
  | t /= 2 = pizza
  | otherwise = pizza {time = 3, state = "Added toppings"}

bake :: Pastry -> Pastry
bake pizza@(Pizza t _)
  | t /= 3 = pizza
  | otherwise = pizza {time = 4, state = "Baked the hell out of
  this"}

bakeMeBaby :: Pastry -> IO ()
bakeMeBaby pizza = do
  let pizza1 = prepareDough pizza
  print pizza1
  let pizza2 = addSauce pizza1
  print pizza2
  let pizza3 = addToppings pizza2
  print pizza3
  let pizza4 = bake pizza3
  print pizza4

main :: IO ()
main = do
  let pizza = Pizza 0 "Raw"
  bakeMeBaby pizza
```

Output of this program

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

2 Conclusion

With the pizza making out of the way, the two approaches feels like one. Obviously, the process must be the same, but the way the pizza, or the data, here is treated differently.

With that said, here is an overkill comparison table.

	OOP	FP
Pros	OOP objects contains both the data (attributes) and things that it can do (methods)	FP decouples the data from the functions
	Any changes that are applied is reflected on the object itself	Data in FP is not intended to change, if it does, new data is just created
	OOP organize everything into hierarchies of abstract objects	FP is natural for the human brain, as in, our thought process is centered around "doing" things
Cons	OOP objects are abstract and potentially complex, wasting a lot of time in abstractions and design patterns instead of solving the problem	A good FP program requires strict state and I/O management, which requires brain-breaking abstraction inversion
	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	FP data objects are immutable, meaning that it is bad for simulating anything that involves objects that change like GUI
	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	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

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 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. The teacher was not very enthusiastic when it comes to FP either.

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.

```python
class iostream:
    def __lshift__(self, other):
        print(other, end='')
        return self

    def __repr__(self):
        return ''


if __name__ == "__main__":
    cout = iostream()
    endl = '\n'
    cout << "Hello" << ", " << "World!" << endl
```

Thanks, and have fun.