Design Patterns Review Packet

Felipe Lima - 109290055

	Singleton	Prototype	Iterator	Factory	Flyweight
Purpose	Have only one instance of the class	Copy fields from one object to another	a way to access each memeber of a collection	To make objects type flexible	Save memory on apps that you need a lot of objects
Type of Design (Creational, Behavioral, Structural)	Creational	Creational	Behavioral	Creational	Structural
Programmer-discipline (or) Language enforced (for c++)	Language enforced	Programmer discipline	Language enforced	Programmer discipline	Programmer discipline
Example use case	Apps only need an instance of (App Store - Mac)	A new document sharing font and margins	Iterate through a vector, map, etc	Objects of time and calendar	Game design

Which design pattern does this code depict? Why did you pick the one that you did?

Singleton.
Private constructor and public getInstance().

Pattern.java

```
public class Pattern {
    private static Pattern instance = new Pattern();
    private Pattern(){
        System.out.println("Goodbye");
    }
    public static Pattern getInstance(){
        return instance;
    }
    public void showMessage(){
        System.out.println("Hello World!");
    }
}
```

PatternDemo.java

```
public class PatternDemo {
    public static void main(String[] args) {
        Pattern object = Pattern.getInstance();
        //show the message
        object.showMessage();
    }
}
```

What would be the output of the main function?

Goodbye
Hello World!

Which design pattern does this code depict? Why did you pick the one that you did?

Factory.
Pass in shape and depending on shape creates an object.

Shape.java

```
public interface Shape {
    void draw();
}
```

Rectangle.java

```
public class Rectangle implements
Shape {
     @Override
     public void draw() {
          System.out.println("Inside
          Rectangle.draw() method.");
     }
}
```

Circle.java

```
public class Circle implements Shape
{
     @Override
     public void draw(){
     System.out.println("Inside
     Circle.draw() method.");
     }
}
```

What would be the output of the main function?

Inside Circle.draw() method.

Inside Rectangle.draw() method.

ShapeChoice.java

```
public class ShapeChoice {
     //use getShape method to get object of type shape
     public Shape getShape(String shapeType){
          if(shapeType == null){
               return null;
          }
          if(shapeType.equalsIgnoreCase("CIRCLE")){
               return new Circle();
          }
          elseif(shapeType.equalsIgnoreCase("RECTANGLE")){
               return new Rectangle();
          }
          return null;
     }
}
```

PatternDemo.java

```
public class PatternDemo {
     public static void main(String[] args) {
          ShapeChoice shapeParty = new ShapeChoice();

          //get an object of Circle and call its draw method.
          Shape shape1 = shapeParty.getShape("CIRCLE"); //call draw
          method of Circle
          shape1.draw();

          //get an object of Rectangle and call its draw method.
          Shape shape2 = shapeParty.getShape("RECTANGLE"); //call draw
          method of Rectangle
          shape2.draw();
     }
}
```

Which design pattern does this code depict? Why did you pick the one that you did?

Prototype.
Uses Clone to create a new instance

What would pUniversity2 contain?

Everything on pUniversity

People.h

```
class IPerson {
public:
    virtual IPerson* Clone() = 0;

    IPerson(const string& sName, const int & id):
    name_(sName),id_(id) {}

private:
    string name_;
    int id_;
};
```

```
class Student: public IPerson
{
public:
    Student(const string& sName, int id):
         IPerson(sName, id){}
    IPerson* Clone(){
        return new Student(name_, id_ + 1);
    }
};
```

```
class Teacher: public IPerson
{
public:
    Teacher(const string& sName, int
id):IPerson(sName, id){}

    IPerson* Clone(){
        return new Teacher("Prof " + name_, id_);
    }
};
```

University.h

```
class University {
public:
    University(const string& sName):name_(sName){
    }
    University(const University& univ):name_(univ.name_){
        for (IPerson * ip : univ.members_){
            members_.push_back((*it)->Clone());
        }
    }
    void AddMember( IPerson* ptr){ members_.push_back(ptr); }
private:
    list<IPerson*> members_;
    string name_;
};
```

main.cpp

```
int main()
{
    University* pUniversity = new University("Oxford");
    IPerson* ptr1 = new Student("Messi",1);
    IPerson* ptr2 = new Student("Ronaldo",2);
    IPerson* ptr3 = new Teacher("Scolari",3);
    pUniversity->AddMember(ptr1);
    pUniversity->AddMember(ptr2);
    pUniversity->AddMember(ptr3);
    University* pUniversity2 = new University(*pUniversity);
    return 0;
}
```

Which design pattern does this code depict? Why did you pick the one that you did?

Iterator
creates an iterator

university.py

```
class University:

    def __init__(self, num_courses):
        self.courses = []
        for i in range(num_courses):
            self.courses.append("course " + str(i))

    def __iter__(self):
        return UniversityMystery(self)

class UniversityMystery:

    def __init__(self, univ):
        self.univ = univ
        self.index = 0

    def __next__(self):
        if self.index < len(self.univ.courses):
            result = self.univ.courses[self.index]
            self.index += 1
            return result
        raise StopIteration
```

example1.py

```
import university

u = University(6)
for c in u:
    print(c)
```

What would be the output of the example1.py?

Print course 1 - 6

What is happening with example2.py?

generate the fibonacci sequence until user interupts

example2.py

```
import time

def fib():
   a, b = 0, 1
   while True:
      yield b
      a, b = b, a + b

g = fib()

try:
   for e in g:
      print(e)
      time.sleep(1)

except KeyboardInterrupt:
   print("Calculation stopped")
```

If you finish, pick a language other than c++ and research how to flyweight design pattern is implemented. Briefly describe how this is done and give an example of the class(es) that you would need to implement. We recommend java or python, but you may look at any object oriented language that you are familiar with.