Design Patterns Review Packet

	Singleton	Prototype	Iterator	Factory	Flyweight
Purpose	only one object needed	an object independent on the class	travese the data structure	create and maintain elements	separate the common feature and unque features
Type of Design (Creational, Behavioral, Structural)	creational	creational	behavior	structural	creational
Programmer-discipline (or) Language enforced (for c++)	language enforced	language enforced	programmer discipline	programmer discipline	programmer discipline
Example use case	in a solar system, we only need one sun, one earth in our model.	a class for selling computer, they all have same attribute, like brand or manufacture date	we define a data structure and we also need to traverse the data structure	when we only want few default element	game object

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

singleton
constructor is in the priivate

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?

Hello World!

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

factory
it provides few
elements

What would be the output of the main function?

Inside Circle.draw() method
Inside Rectangle.draw() method

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.");
     }
}
```

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
clone method

What would pUniversity2 contain?

member ptr1,2,3
name = “Oxford”

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
__iter__

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?

course0
course1
course2
course3
course4
course5

What is happening with example2.py?

it keeps printing fibonacci number

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.

python

python will need to implement a flyweight factory to maintain and store the created objected that has common

```
class FlyWeight:
def __init__(self, s):
self.s = s

def say(self):
print(f"{self.s}:"i'm in factory")


class FlyWeightFactory:
def __init__(self):
self.d = {}

def getFlyWeight(self, s):
if s in self.d:
return self.d[s]
else:
r = FlyWeight(s)
self.d[s] = r
return r


if __name__ == '__main__':
f = FlyWeightFactory()
d1 = f.getFlyWeight('one')
d1.say()
d2 = f.getFlyWeight('two')
d2.say()
d3 = f.getFlyWeight('one')
d3.say()
```