*객체지향 설계와 패턴*  
**LAB ASSIGNMENT #8**

## Weight 10% 마감 5월 9일(일)

**목적: 생성패턴(빌더, 팩토리 메소드, 추상팩토리)의 적용**

문제 #1: Java 언어에서 사용과 동작을 제어하는 규칙을 3~4개 나열하라.

1. 자기자신의 attribute를 가리키는 this, 수퍼클래스의 attribute나 메소드를 가리키는 super는 생성자 구현안에서 가장 첫번째로 등장해야한다

2. 생성자는 명시된 리턴타입이 없다

3. 생성자 이름과 클래스 이름은 동일하다

4. 생성자를 정의하지 않는다면 default 생성자가 제공된다

문제 #2: 빌더 패턴

다음과 같은 User 클래스를 설계하였다. 생략 가능한 파라미터를 고려하여 생성자를 다음과 같이 작성하였다고 하자.

public class User {

    private final String firstName;    //required

    private final String lastName;    //required

    private final int age;    //optional

    private final String phone;    //optional

    private final String address;    //optional

public User(String firstName, String lastName) {

    this(firstName, lastName, 0);

}

public User(String firstName, String lastName, int age) {

    this(firstName, lastName, age, "");

}

public User(String firstName, String lastName, int age, String phone) {

    this(firstName, lastName, age, phone, "");

}

public User(String firstName, String lastName, int age, String phone, String address) {

    this.firstName = firstName;

    this.lastName = lastName;

    this.age = age;

    this.phone = phone;

    this.address = address;

}

...

}

(1) 위와 같은 방식의 생성자 구현의 문제점은 무엇인가?

데이터가 일부는 제공되고, 다른 일부는 제공되지 않는 상태에서 객체를 만든다면 정보가 모두 존재하는 객체 뿐만 아니라 일부 정보만 가진 객체들 모두 생성할 수 있게된다.

(2) Java는 생성자를 정의하지 않으면 디폴트 생성자를 제공한다. 다음과 같이 제공되는 생성자를 이용한다면 문제점은 무엇인가? 두 가지 이상 들라.

첫번째. 인스턴스 생성이 일관적이지 않다. attribute를 일부만 가진 객체(혹은 전부 가지지 않은)가 생성되어도, client는 이미 인스턴스가 생성되었다고 생각한다

두번째. 인스턴스가 mutable하게 된다.

public class User {

    private String firstName; // required

    private String lastName; // required

    private int age; // optional

    private String phone; // optional

    private String address;  //optional

    public String getFirstName() {

        return firstName;

    }

    public void setFirstName(String firstName) {

        this.firstName = firstName;

    }

    public String getLastName() {

        return lastName;

    }

    public void setLastName(String lastName) {

        this.lastName = lastName;

    }

    public int getAge() {

        return age;

    }

    public void setAge(int age) {

        this.age = age;

    }

    public String getPhone() {

        return phone;

    }

    public void setPhone(String phone) {

        this.phone = phone;

    }

    public String getAddress() {

        return address;

    }

    public void setAddress(String address) {

        this.address = address;

    }

}

(3) 빌더 패턴을 이용하여 First Name, Last Name 만 있더라도 객체가 생성될 수 있도록 User 클래스를 수정하라.

첨부 코드 참조\_

문제 #3: 팩토리 패턴

다음은 모터를 구동하여 운영하는 엘리베이터를 간략화한 코드이다.

public enum Direction {UP, DOWN}

public enum MotorStatus {MOVING, STOPPING}

public abstract class Motor {

private MotorStatus motorStatus;

public Motor() {

motorStatus = MotorStatus.STOPPED;

}

public MotorStatus getMotorStatus() {

return motorStatus;

}

private void setMotorStatus(MotorStatus motorStatus) {

this.motorStatus = motorStatus;

}

private void move(Direction direction) {

MotorStatus motorStatus = getMotorStatus();

if (motorStatus == MotorStatus.MOVING)

return;

moveMotor(direction);

setMotorStatus(MotorStatus.MOVING);

}

}

public class LGMotor extends Motor {

protected void moveMotor(Direction direction) {

System.out.println(“move LG Motor “ + direction);

}

}

public class HyundaiMotor extends Motor {

protected void moveMotor(Direction direction) {

System.out.println(“move Hyundai Motor “ + direction);

}

}

public class ElevatorController {

private int id;

private int curFloor = 1;

private Motor motor;

public ElevatorController(int id, Motor motor) {

this.id = id;

this.motor = motor;

}

public void gotoFloor(int destination) {

if (destination = curFloor)

return;

Direction direction;

if (destination > curFloor)

direction = Direction.UP;

else

direction = Direction.DOWN;

motor.move(direction);

System.out.print(”Elevator [“ + id + “] floor:” + curFloor);

curFloor = destination;

System.out.println(“ ==> “ + curFloor + “ with “ + motor.getClass().getName());

motor.stop();

}

}

(1) 다음 클라이언트 프로그램의 실행 결과는 무엇인가? 이런 설계에 어떤 문제점이 있는가?

변경하거나 확장할 때, LG나 Hyundail의외의 모터가 추가된다면 그때마다 코드를 고쳐야한다. 즉. 객체의 생성이 구체적인 것에 의존되어 있다. 이는 OCP와 DIP에 어긋난다.

public class Client {

public static void maint(String[] args) {

Motor lgMotor = new LGMotor();

ElevatorController controller1 = new ElevatorController(1, lgMotor);

controller1.gotoFloor(5);

controller1.gotoFloor(3);

Motor hyundaiMotor = new HyundaiMotor();

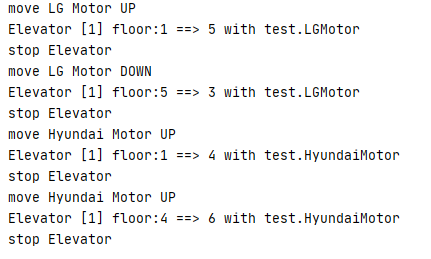
ElevatorController controller2 = new ElevatorController(1, hyundaiMotor);

controller2.gotoFloor(4);

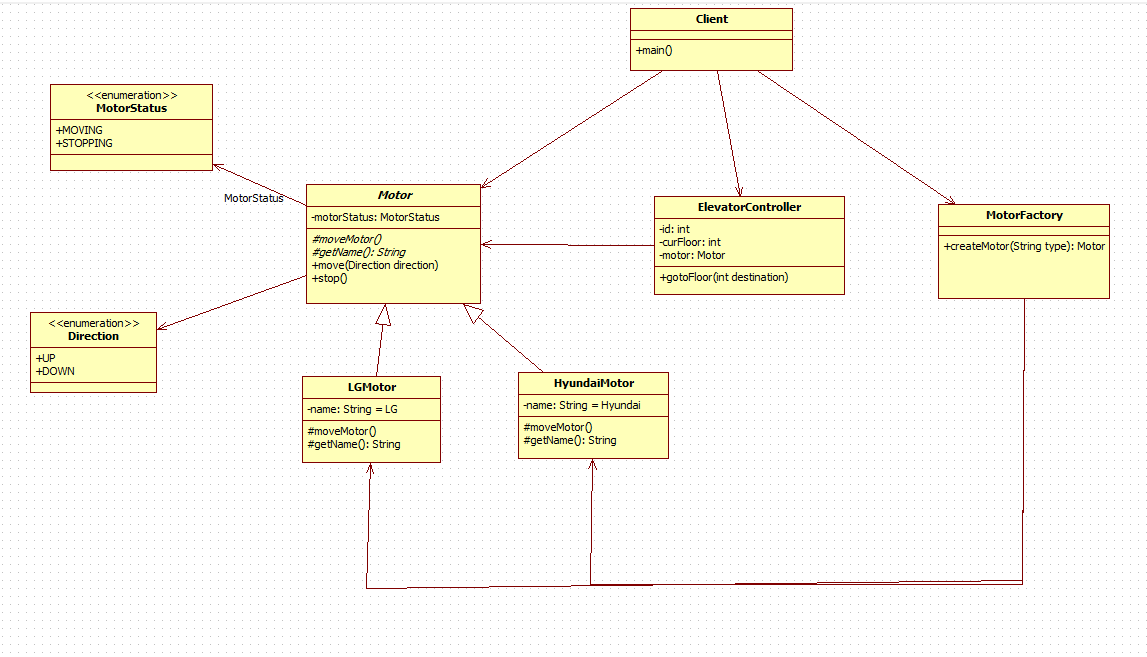
controller2.gotoFloor(6);

}

}



(2) 위 프로그램을 팩토리 패턴을 적용하여 설계를 개선하고 UML 클래스 다이어그램으로 표현하라.



(3) LGMotor와 HyundaiMotor 객체를 생성하는 MotorFactory 클래스를 구현하라.

첨부 코드 참조

**제출물**

제출하여야 할 것:

1. 모든 답은 빈칸에 쓸 것

2. UML 다이어그램은 StarUML을 이용하여 그린 후 복사할 것

3. 실행 결과는 이클립스 실행 결과를 캡처해도 좋음.

**평가**

과제 #9는 다음과 같은 기준으로 평가할 것임:

1. 원시코드의 정확성

2. 설계의 타당성

3. 설계, 구현 결과물에서 볼 수 있는 설계에 대한 이해 수준