*객체지향 설계와 패턴*  
**LAB ASSIGNMENT #10**

## Weight 10% 마감 5월 16일(일)

**목적: 프로토타입, 메멘토 패턴의 적용**

문제 #1: 프로토타입 패턴은 언제 사용하는가? (다중 선택 가능)

➀ 종류가 너무 많아 클래스로 정리되지 않는 경우

➁ 클래스로부터 인스턴스 생성이 어려운 경우

➂ framework와 생성할 인스턴스를 분리하고 싶은 경우

➃ 객체 하나로 인스턴스화 되는 것을 막기 위해

문제 #2: 프로토타입 패턴을 적용한 결과는? 해당되는 것을 모두 고르시오: 1,3

➀ 각 프로토타입 클래스가 클론 메소드를 구현한다.

➁ 특정 객체의 패밀리를 서로 협력하기 쉽게 한다.

➂ 클라이언트 코드가 새로운 프로토타입 인스턴스를 런타임에 등록 가능하게 한다.

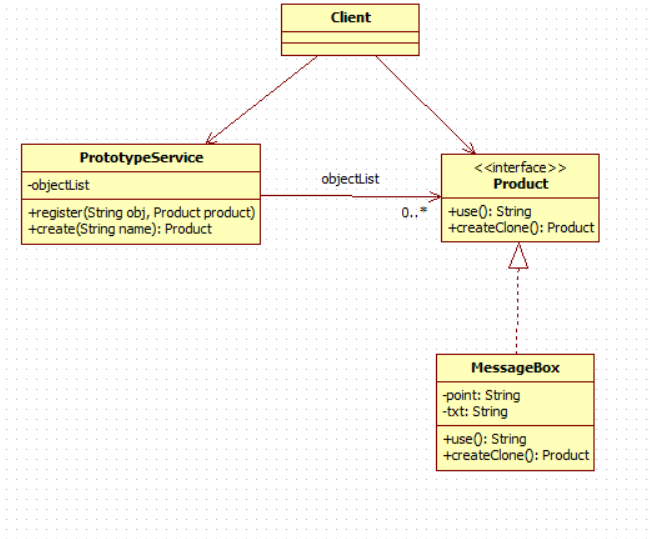
➃ 다른 팩토리 패턴과 비교하여 클래스 상속 구조를 줄인다.

문제 #3. 다음 프로그램은 문자열의 앞뒤를 \* 혹은 \_로 감싸는 프로그램이다. 즉, 인풋으로 TEST가 들어온다면 아웃풋으로는 \_TEST\_ 혹은 \*TEST\* 를 출력한다.

이를 프로토타입 패턴을 적용하여 설계하여라. 필요한 설계 요소는 다음과 같다.

* Product : 추상 메소드 use와 createClone이 선언되어 있는 인터페이스
  + use : 메서드가 해당하는 동작(앞뒤를 입력받은 문자로 감싼다)을 수행하게 하는 인터페이스
  + createClone : 자기 자신을 복사하여 새로운 인스턴스를 리턴하는 메소드
* PrototypeService : 복사할 객체를 등록하는 register와 입력받은 객체를 복사하여 생성하는 create 메서드가 있는 서비스
  + register : 복사할 객체를 등록한다. key는 객체의 다운캐스팅한 클래스 이름으로 한다.
  + create : 등록되어 있는 객체에서 클래스명으로 해당하는 객체를 복사한다.
* MessageBox : Product를 구현한 클래스

1) 위 설계에 대한 클래스 다이어그램을 그려라.



2) Product, PrototypeService, MessageBox 를 각각 코딩하라.

public interface Product {  
 String use();  
  
 Product createClone();  
}

public class MessageBox implements Product {  
 private String point;  
 private String txt;  
  
 public MessageBox(String txt, String point) {  
 this.point = point;  
 this.txt = txt;  
 }  
  
 @Override  
 public String use() {  
 return point + txt + point;  
 }  
  
 @Override  
 public Product createClone() {  
 return new MessageBox(txt, point);  
 }  
}

public class PrototypeService {  
 private HashMap objectList = new HashMap<>();  
  
 public void register(String obj, Product product) {  
 objectList.put(obj, product);  
 }  
  
 public Product create(String name) {  
 return ((Product) objectList.get(name)).createClone();  
  
 }  
}

문제 #4: 프로토타입 패턴

프로토타입 패턴을 이용하는 고객 정보 기재 어플리케이션을 생각해보자. 이 프로그램은 Universal Retailer의 고객 데이터베이스에 고객 정보를 입력하기 쉽게 한다. 이 회사는 하나의 데이터베이스만을 갖고, 회사 사원에게는 할인을 해준다고 가정한다. 또 사원들의 봉급에서 할부 금액이 직접 납입된다. 이 회사의 정책은 비용을 줄이기 위해 사원의 이름을 하나의 묶음(batch)으로 제출한다. 사원을 세 카테고리로 나누고, 모든 사원을 각 카테고리에 맞게 선택한다.

① “높은 거래량” 고객은 일년 간 $2,000 초과 구매 시, 추가 5%의 할인을 받는다. 하지만 $50이 안될 경우에는 5%를 더 지불해야 한다.

② “중간 거래량” 고객은 어떤 추가 할인이나 추가 지불은 없지만, 일년 간 거래량이 $30이 안될 경우, 삭제된다.

③ “낮은 거래량” 고객은 매년 서비스 요금으로 $10이 청구된다. 그러나 추가 할인이나 어떤 제약도 없다.

Universal의 매달 청구 과정은 데이터베이스에 있는 모든 고객에게 청구서를 메일로 보낸다. 다음과 같은 송장(invoice)이 출력된다.

. . .

John Doe of Universal Electric; deduct $50.00 from paycheck; . . .

Betty Don of Ultra-paramount; deduct $35.00 from paycheck; . . .

John Dorr of Universal Electric; deduct $50.00 from paycheck; . . .

. . .

다음의 사용 사례는 이 시스템의 주요 기능을 설명한다.

“Company entry" 사용 사례

전제 조건 : 없음

1. 어플리케이션은 일련의 고객을 후원하는 회사의 이름을 요청한다.

2. 사용자는 회사의 이름을 제공한다.

3. 어플리케이션은 회사에 관한 광범위한 정보와 고객에게 청구할 수단을 요청한다.

4. 사용자는 회사 정보를 입력한다.

5. 새로운 고객에게는 “Customer entry" 사용 사례를 적용한다.

6. 어플리케이션은 이 단계에서 만들어진 모든 고객의 기록을 모니터에 출력한다.

“Customer entry" 사용 사례

전제 조건 : 어플리케이션이 새로운 고객을 받을 준비가 되어있어야 한다.

1. 어플리케이션은 고객에 대한 정보를 요청한다.

2. 사용자 정보를 입력한다.

3. 어플리케이션이 정보를 다시 화면에 보여준다.

4. 어플리케이션은 모든 고객과 회사 정보를 담고 있는 고객 기록을 고객 데이터베이스에 넣는다.

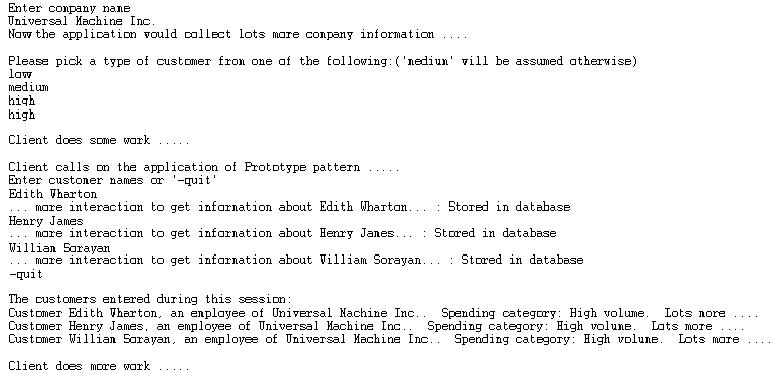
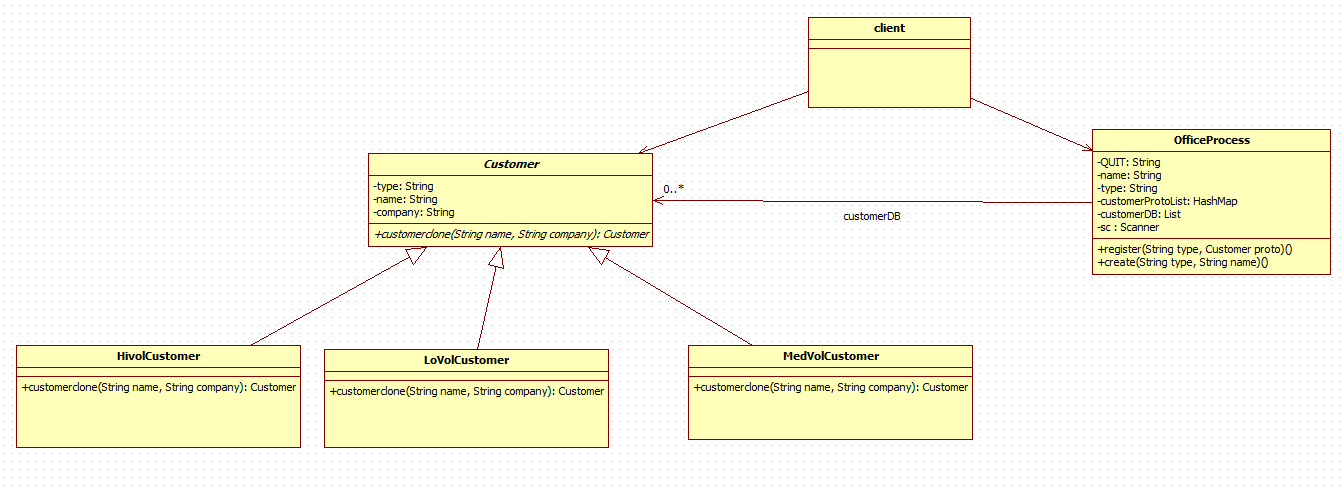


그림 4-12 고객 정보 기재 어플리케이션의 일반적인 시나리오

(1) 프로토타입 패턴을 적용하여 실행시간에 특정한 정적(static) Customer 객체(prototype)를 생성하는 어플리케이션을 디자인하라.



(2) 위 설계를 이용하여 Customer, HiVolCustomer, MedVolCustomer, LoVolCustomer 클래스를 코딩하라.

<Customer 클래스>

|  |
| --- |
| public abstract class Customer {  private String type;  private String name;  private String company;   public Customer(String name, String company) {  this.name = name;  this.company = company;  }   abstract Customer customerclone(String name, String company);   public String getType() {  return type;  }   public String getName() {  return name;  }   public String getCompany() {  return company;  } } |

<OfficeProcess 클래스>

|  |
| --- |
| public class OfficeProcess {  private static String *QUIT* = "-quit";  Scanner sc = new Scanner(System.*in*);  private String name;  private String type;  private HashMap customerProtoList = new HashMap<>();  private List<Customer> customerDB = new ArrayList<>();   public void register(String type, Customer proto) {  customerProtoList.put(type, proto);  }   public void setName() {  System.*out*.println("Enter Company Name");  String name = sc.nextLine();  this.name = name;  System.*out*.println("Now application would collect lots more company information...");  }   public void setType() {  System.*out*.println("Please pick a type of customer from one of the following : ('medium' will be assumed otherwise)");  System.*out*.println(customerProtoList.keySet());  String type = sc.nextLine();  this.type = type;  System.*out*.println();  }    public Customer create(String type, String name) {  return ((Customer) customerProtoList.get(type)).customerclone(name, this.name);  }   public void setCustomer() {  System.*out*.println("Client does some work...\n");  System.*out*.println("Enter Customer names or '" + *QUIT* +"'");  String customerName;  while (true) {  customerName = sc.nextLine();  if (customerName.equals(*QUIT*)) {  System.*out*.println("The customers entered during this session : ");  for (Customer customer : customerDB) {  System.*out*.println("Customer " + customer.getName() + ", an employee of "  + customer.getCompany() + ".. Spending category : "  + customer.getType() + " volume. Lots more...");  }  System.*out*.println("Client does more work...");  break;  }else{  System.*out*.println("...more interaction to get information about " + customerName + " ... : Stored in database");  Customer result = this.create(type, customerName);  customerDB.add(result);    }  }  } } |

<HiVolCustomer 클래스>

|  |
| --- |
| public class HivolCustomer extends Customer {  private String type = "high";  private String name;  private String company;   public HivolCustomer(String name, String company) {  super(name, company);  }   @Override  Customer customerclone(String name, String company) {  return new HivolCustomer(name, company);  }  } |

<MedVolCustomer 클래스>

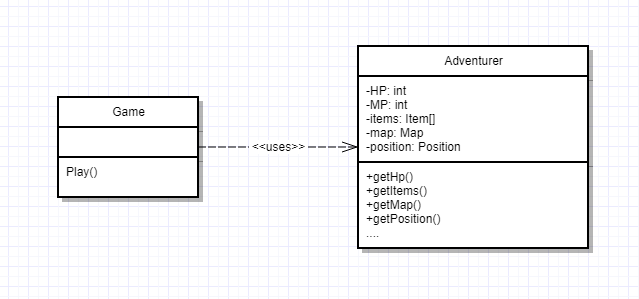
|  |
| --- |
| public class MedVolCustomer extends Customer {  private String type = "medium";  private String name;  private String company;   public MedVolCustomer(String name, String company) {  super(name, company);  }   @Override  Customer customerclone(String name, String company) {  return new MedVolCustomer(name, company);  }   } |

<LoVolCustomer 클래스>

|  |
| --- |
| public class LoVolCustomer extends Customer {  private String type = "low";  private String name;  private String company;   public LoVolCustomer(String name, String company) {  super(name, company);  }   @Override  Customer customerclone(String name, String company) {  return new LoVolCustomer(name, company);  }  } |

문제 #5: 메멘토 패턴

어떤 스토리가 있는 RPG게임을 생각해보자. 사용자는 탐험가를 통해 게임을 플레이할 것이며 각 단계의 스테이지를 깨나가면서 캐릭터는 성장하게 될 것이다. 몬스터들을 처치하면서 아이템도 오를 것이고 HP와 MP도 사용할 것이고 탐험가는 맵을 탐험하면서 위치도 계속해서 변경된다.



1. 위와 같은 설계에 대한 문제점을 설명하라.
2. 다음과 같이 Snapshot 클래스를 상태로 저장하는 메멘토 패턴을 적용하여 다시 설계하라.

struct StateSnapshot {

public int Hp { get; set; }

public int Mp { get; set; }

public Position Position { get; set; }

public Map CurrentMap { get; set; }

public IEnumerable<Item> Items { get; set; }

}

1. 위 설계에 대한 코딩을 작성하라.

**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