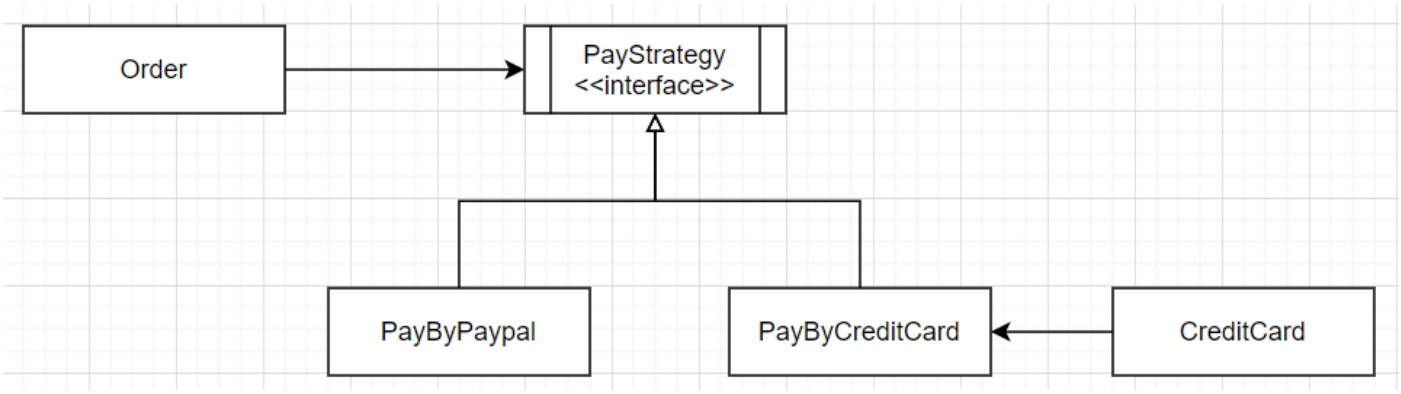
설명 1. Strategy Pattern :

- 행동을 클래스로 캡슐화하여 동적으로 행동 전략을 바꿀 수 있게 해주는 디자인 패턴입니다.
- 코드 블럭 내부를 일일이 수정하지 않고도 손쉽게 전략을 바꿀 수 있습니다.

문제 1. 전자 상거래 시스템

직장인 A씨는 회사를 나와 컴퓨터 부품을 파는 전자 상거래 애플리케이션을 만들기 위해 구체적인 시장 조사를 들어갔습니다. 일반고객들은 신용 카드를 이용한 온라인 결재 시스템을 이용하기도 했지만 페이팔이라는 결제 수단을 이용해서 결제를 한다는 것을 알았습니다. 그래서 두가지 결제 시스템을 활용하기로 하였지만 그러기에는 결제 방식을 제외하고 중복되는 코드가 많다는 것을 깨닿게 됩니다. 그래서 직장인 A씨는 해당 문제를 해결하기 위해 다음과 같이 설계를 진행합니다.



위의 설계를 바탕으로 코드와 StretageMain을 완성해주세요

PayStrategy

```
public interface PayStrategy {
    boolean pay(int paymentAmount);
    void collectPaymentDetails();
}
```

CreditCard

```
// 카드 데이터를 담는 클래스
public class CreditCard {
    private int amount;
    private String number;
    private String date;
    private String cvv;

    CreditCard(String number, String date, String cvv) {
        this.amount = 100000;
        this.number = number;
        this.date = date;
        this.cvv = cvv;
    }

    public void setAmount(int amount) {
        this.amount = amount;
    }

    public int getAmount() {
        return amount;
    }
}
```

PayByPaypal

```java
public class PayByPayPal implements PayStrategy {
    // 고객정보 데이터 베이스
    private static final Map<String, String> DATA_BASE = new HashMap<>();
    private Scanner scan = new Scanner(System.in);
    // 고객 정보 이메일
    private String email;
    // 고객 정보 패스워드
    private String password;
    // 인증 여부
    private boolean signedIn;

    // 임시로 들어갈 데이터 <패스워드, 이메일>
    static {
        DATA_BASE.put("mike", "mike@naver.com");
        DATA_BASE.put("john", "john@daum.net");
    }

    /**
     * 사용자 데이터 수집.
     */
    @Override
    public void collectPaymentDetails() {
        while (!signedIn) {
            System.out.print("이메일을 입력하세요: ");
            email = scan.nextLine();
            System.out.print("패스워드를 입력하세요: ");
            password = scan.nextLine();
            if (  (1)  ) {
                System.out.println("데이터 확인에 성공하였습니다.");
            } else {
                System.out.println("잘못된 이메일 패스워드 형식입니다!");
            }
        }
    }
```

```java
/*
 * 고객 정보를 확인한다
 * */
private boolean verify() {
    setSignedIn(  (2)  );
    return signedIn;
}

private void setSignedIn(boolean signedIn) {
    this.signedIn = signedIn;
}

/**
 * 향후 쇼핑 시도를 위해 고객 데이터를 저장한다.
 */
@Override
public boolean pay(int paymentAmount) {
    if (signedIn) {
        System.out.println("페이팔로 " + paymentAmount + " 가 결제 되셨습니다.");
        return true;
    } else {
        return false;
    }
}
```

PayByCreditCard

```java
public class PayByCreditCard implements PayStrategy {

    private Scanner scan = new Scanner(System.in);
    private CreditCard card;
    //카드 정보를 수집한다.
    @Override
    public void collectPaymentDetails() {

        System.out.print("카드 번호를 입력해 주세요: ");
        String number = scan.nextLine();
        System.out.print("카드 만료 일자를 입력하세요 'mm/yy': ");
        String date = scan.nextLine();
        System.out.print("CVV 코드를 입력하세요: ");
        String cvv = scan.nextLine();
        // 카드에 정보 저장
        card = (3);
        // 카드 넘버 검증 절차 생략...
    }

    // 카드 확인 후 고객의 신용카드로 지불할 수 있습니다.
    @Override
    public boolean pay(int paymentAmount) {
        if (cardIsPresent()) {
            System.out.println(paymentAmount + " 가 카드로 결제되셨습니다.");
            card.setAmount(card.getAmount() - paymentAmount);
            return true;
        } else {
            return false;
        }
    }

    private boolean cardIsPresent() {
        return (4);
    }
```

StratagyMain

```
public class StretageMain {
    // 상품과 가격 정보
    private static Map<Integer, Integer> priceOnProducts = new HashMap<>();
    private Scanner scan = new Scanner(System.in);
    // 주문
    private static Order order = new Order();
    // PayStrategy 객체를 담을 전략 객체
    private static PayStrategy strategy;

    // 상품 내용
    static {
        priceOnProducts.put(1, 2200);
        priceOnProducts.put(2, 1850);
        priceOnProducts.put(3, 1100);
        priceOnProducts.put(4, 890);
    }

    public StretageMain() {
        while (!order.isClosed()) {
            int cost;

            String continueChoice;
            do {
                System.out.println("상품을 선택해주세요:");
                System.out.println("1 - Mother board");
                System.out.println("2 - CPU");
                System.out.println("3 - HDD");
                System.out.println("4 - Memory");
                int choice = scan.nextInt();
                cost = priceOnProducts.get(choice);
                System.out.print("갯수: ");
                int count = scan.nextInt();
                order.setTotalCost(cost * count);
                System.out.print("상품을 더 구매하시겠습니까? Y/N: ");
                scan.nextLine();
                continueChoice = scan.next();
            } while (                    (5)                    );
```

```
        if (strategy == null) {
            System.out.println("결재 방식을 선택해주세요:");
            System.out.println("1 - PalPay");
            System.out.println("2 - Credit Card");
            int paymentMethod = scan.nextInt();

            // 클라이언의 선택에 따라 다른 결제 전략을 도입한다.
            (6)
        }

        // 전략만이 지불을 처리하는 데 필요한 데이터를 알 수 있으며 order에서 처리가 가능하다.
        order.(7)

        scan.nextLine();
        System.out.print(order.getTotalCost() + " 지불됩니다. 계속하시겠습니까? P/C: ");
        String proceed = scan.nextLine();
        if (proceed.equalsIgnoreCase("P")) {
            // 결제 모듈 실행.
            if (strategy.pay(order.getTotalCost())) {
                System.out.println("결제가 성공적으로 진행되었습니다.");
            } else {
                System.out.println("결제가 실패하였습니다.");
            }
            order.setClosed();
        }
    }
}
```

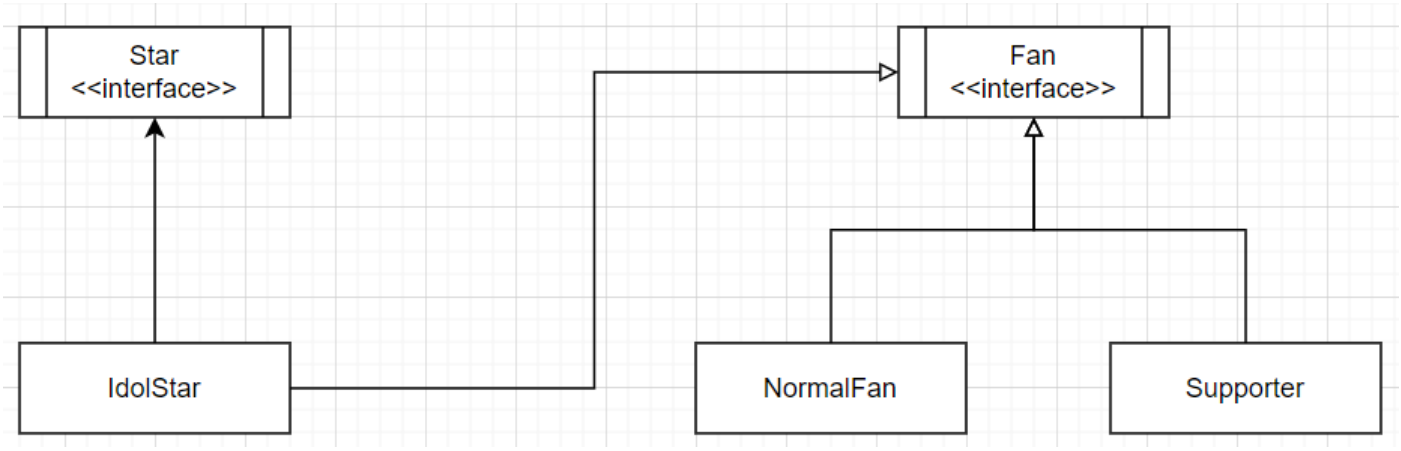
```
상품을 선택해주세요:
1 - Mother board
2 - CPU
3 - HDD
4 - Memory
1
갯수: 1
상품을 더 구매하시겠습니까? Y/N: Y
상품을 선택해주세요:
1 - Mother board
2 - CPU
3 - HDD
4 - Memory
2
갯수: 2
상품을 더 구매하시겠습니까? Y/N: N
결재 방식을 선택해주세요:
1 - PalPay
2 - Credit Card
1
이메일을 입력하세요: mike@naver.com
패스워드를 입력하세요: mike
데이터 확인에 성공하였습니다.
5900 지불됩니다. 계속하시겠습니까? P/C: P
페이팔로 5900 가 결제 되셨습니다.
결제가 성공적으로 진행되었습니다.
```

설명 2. Observe Pattern

- 한 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체들한테 연락이 가고 자동으로 내용이 갱신되는 방식으로, 일대다 의존성을 정의합니다.

문제 2. 아이돌 실시간 커뮤네케이션 플랫폼

K 엔터테인먼트 회사는 자신들이 운영하는 게시판에 실시간 아이돌 스타와 소통을 할 수 있는 방송 시스템을 개발하고자 한다. 하지만 모든 사람들이 모든 아이돌 스타를 좋아하지는 않기 때문에 실시간 방송을 듣고 싶은 사람에게만 아이돌과 실시간 소통 방송을 볼 수 있도록 만들려고 한다. 팬은 크게 두가지로 나뉠 수 있는데 하나는 일반 아이돌 펜이고 다른 하나는 직,간접적으로 특정 아이돌을 서포트 해주는 서포터즈가 존재한다. 두 회원의 기능이나 인터페이스는 확연히 다르기 때문에 아이돌 방송 시청 시 팔로우와 언팔로우 기능을 넣기 위해 두 객체를 하나로 합치는건 불가능해서 결국 아래와 같이 설계하기로 결정했다.



다음의 구조를 보고 해당 클래스들과 Main문을 완성하라

Fan

```
public interface Fan {
    public void update(String msg);
}
```

NormalFan

```
public class NormalFan implements Fan {

    public void update(String msg) {
        System.out.println("Normal Fan에게 보내는 메시지 : "+msg);
    }

}
```

Supporter

```
public class Supporter implements Fan {

    public void update(String msg) {
        System.out.println("서포터즈에게 보내는 메세지:"+msg);
    }

}
```

Star

```
public interface Star {
    public void follow(Fan fan);
    public void unfollow(Fan fan);
    public void notifyFans(String msg);
}
```

IdolStar

```
public class IdolStar implements Star {

    ArrayList<Fan> fans = new ArrayList<>();

    // 해당 스타의 방송을 팔로우 합니다
    public void follow(Fan fan) {
        fans.add(fan);
    }

    // 해당 스타의 방송을 언팔로우 합니다.
    public void unfollow(Fan fan) {
        [redacted]
    }

    // 팬들에게 방송합니다.
    public void notifyFans(String msg) {
        [redacted];
    }

}
```

PatternMain

```
public static void main(String[] args) {
    Fan normal = new NormalFan();
    Fan sup = new Supporter();

    Star kim = new IdolStar();

    kim.follow(normal);
    kim.follow(sup);
    kim.notifyFans("안녕하세요 모두들!!");

    kim.unfollow(normal);
    kim.notifyFans("서포터즈들 감사합니다!!");
}
```

```
Normal Fan에게 보내는 메시지 : 안녕하세요 모두들!!
서포터즈에게 보내는 메세지:안녕하세요 모두들!!
서포터즈에게 보내는 메세지:서포터즈들 감사합니다!!
```

설명 3. Decorator Pattern

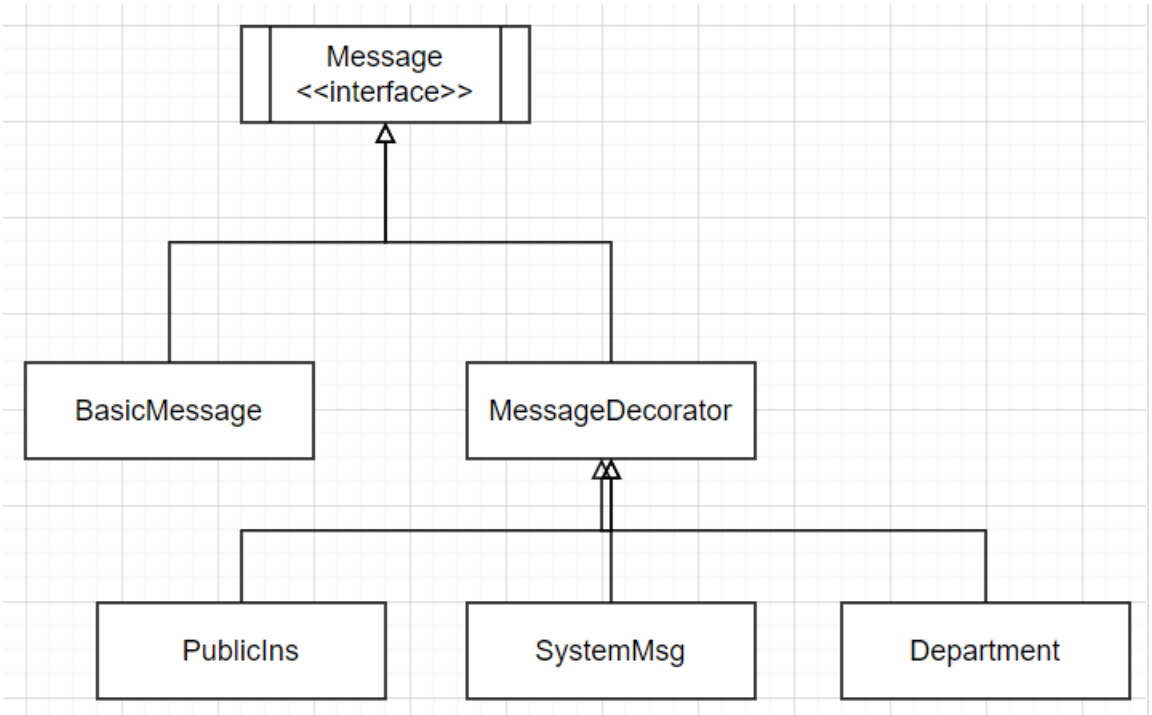
- 객체에 추가적인 요건을 동적으로 첨가합니다.
- 쉽게 말해 장식과 같이 하나의 객체에 여러 기능을 꾸며줍니다.
- 데코레이터는 서브클래스를 만드는 것을 통해서 기능을 유연하게 확장할 수 있는 방법을 제공합니다.

문제 3. 메시지 허브 시스템

Z회사에서 A 공공기관에 납품하는 내 외부의 데이터를 중계해주는 시스템을 만들었다. 이 시스템은 메시지를 안에 있는 시스템들끼리 통신하게 할 수도 있지만 외부의 다른 공공기관에도 보낼 수 있도록 설계되었다. 그러나 만들고 해당 시스템을 공기관에서 돌리면서 문제가 발생했다. A 공공기관에서는 Z회사에게 다음과 같은 추가 요청을 이야기했다.

1. 내부의 시스템과 통신할 때는 메시지 앞에 시스템 명(예를들면 "시스템A-") 를 붙여서 메시지를 보낼 것
2. 외부의 공공기관과 통신할 때는 메시지 앞에 공공기관 명과 시스템 명("공공B-시스템A-")을 붙여서 메시지를 보낼 것
3. 만약 공공기관도 시스템도 아닌 부서에 메시지를 넣어 전송할 경우 메시지 뒤에 부서 명("-부서A")을 붙여서 메시지를 보낼 것

이미 모든 개발이 완료된 상황에서 이 요구조건을 위해 기능을 전부 틀어버릴 수는 없고 하는 수 없이 Z회사의 개발자들은 다음과 같이 모델을 개발하기로 한다.



위의 구조를 보고 시스템을 완성하라

Message

```
public interface Message {
    public void operate();
}
```

BasicMessage

```
public class BasicMessage implements Message {

    String text = "";

    public BasicMessage(String text) {
        this.text = text;
    }

    public void operate() {
        System.out.print(text);
    }

}
```

MessageDecorator

```
public class MessageDecorator implements Message {

    private Message message;

    public MessageDecorator(Message message) {
        this.message = message;
    }

    public void operate() {
        message.operate();
    }
}
```

PublicIns

```
public class PublicIns extends MessageDecorator {

    String subText = "";

    public PublicIns(Message message, String subText) {
        super(message);
        this.subText = subText;
    }

    public void operate() {
        System.out.print(subText+"-");

    }
}
```

SystemMsg

```
public class SystemMsg extends MessageDecorator {

    String subText = "";

    public SystemMsg(Message message, String subText){
        super(message);
        this.subText = subText;
    }

    public void operate() {
        System.out.print(subText+"-");

    }

}
```

Department

```
public class Department extends MessageDecorator {

    String subText = "";

    public Department(Message message, String subText) {
        super(message);
        this.subText = subText;
    }

    public void operate() {

        System.out.print("-"+subText);
    }
}
```

DecoratorMain

```
public static void main(String[] args) {

    System.out.println("1. 그냥 전달되는 일반 메시지");
    Message basic = new BasicMessage("그냥 보내는 일반 메시지");
    basic.operate();

    System.out.println();
    System.out.println("----------------------");

    System.out.println("2. 시스템에 전달되는 메시지");
    Message system1 = new SystemMsg(
            new BasicMessage("A시스템에 보내는 쿼리"),"시스템A");
    system1.operate();

    System.out.println();
    System.out.println("----------------------");

    System.out.println("3.공공B에 시스템B에 보내는 메시지");
    Message system2 =
            new PublicIns(
            new SystemMsg(
            new BasicMessage("공공B 기관 시스템B에 보내는 메시지"),
                        "시스템B"),
                    "공공B");
    system2.operate();

    System.out.println();
    System.out.println("----------------------");

    System.out.println("4. 부서A에 전달되는 메시지");
    Message dep1 = new Department(
            new BasicMessage("A부서에 보내는 쿼리"),"부서A");
    dep1.operate();
}
```

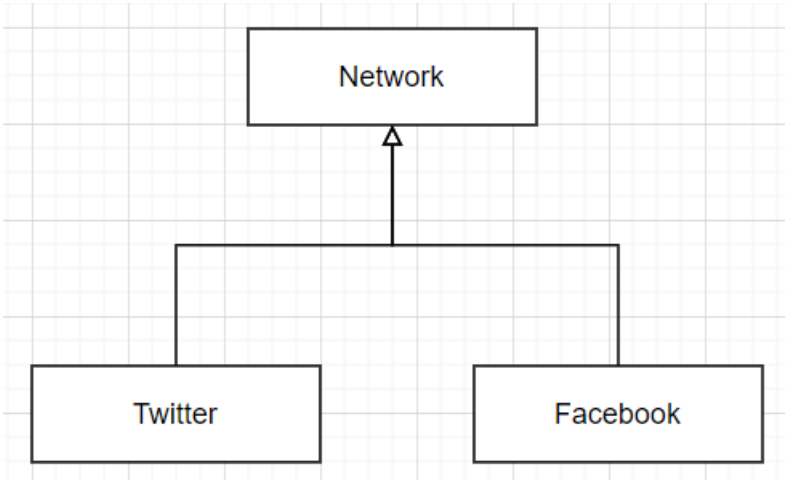
```
1. 그냥 전달되는 일반 메시지
그냥 보내는 일반 메시지
----------------------
2. 시스템에 전달되는 메시지
시스템A-A시스템에 보내는 쿼리
----------------------
3.공공B에 시스템B에 보내는 메시지
공공B-시스템B-공공B 기관 시스템B에 보내는 메시지
----------------------
4. 부서A에 전달되는 메시지
A부서에 보내는 쿼리-부서A
```

실행 4. Template Method Pattern

- 메서드에서 알고리즘의 골격을 정의한다
- 알고리즘의 여러 단계 중 일부는 서브클래스에서 구현할 수 있다.
- 이때 알고리즘의 구조를 유지하며 서브클래스에서 특정 단계를 재정의할 수 있다.

문제 4. 소셜 네트워크 자동 메크로 프로그램

태영씨는 소셜 네트워크에 본인의 사업을 광고하기 위해 밤낮으로 소셜 네트워크를 로그인하며 광고를 올렸다. 하지만 소셜 네트워크에 본인의 사업 광고를 일일히 달기에는 번거롭고 사람을 쓰자니 사업 초반에 인건비를 그만큼 들여가며 광고를 진행하기에는 무리가 있었다. 해서 태영씨는 자동으로 광고를 달아주는 메크로 프로그램을 제작하기로 결정한다. 그런데 문제가 생긴 것이 태영씨가 사용하는 소셜 네트워크는 페이스북과 트위터였는데 이 두가지는 글을 쓰는 메커니즘이 판이하게 달랐다. 결국 태영씨는 이것을 극복하기 위해 공통로직을 제외한 나머지 로직을 추상화하여 다음과 같이 로직을 완성시킨다.



위의 그림을 참조하여 아래 로직과 메인을 완성시켜 구현하라

Network

```
public abstract class Network {
    String userName;
    String password;

    Network() {}

    /**
     * 어떤 SNS 서비스를 사용을 떠나서 해당 메서드를 실행하여 포스팅한다.
     */
    public boolean post(String message) {
        // 포스팅 전 로그인 시도를 한다.
        if (logIn(this.userName, this.password)) {
            // 데이터를 전송한다.
            boolean result = sendData(message.getBytes());
            // 로그아웃
            logOut();
            return result;
        }
        return false;
    }

    abstract boolean logIn(String userName, String password);
    abstract boolean sendData(byte[] data);
    abstract void logOut();
}
```

FaceBook

```java
public class Facebook extends Network {
    public Facebook(String userName, String password) {
        this.userName = userName;
        this.password = password;
    }

    public boolean logIn(String userName, String password) {
        System.out.println("유저 인증 확인");
        System.out.println("이름: " + this.userName);
        System.out.print("비밀번호: ");
        for (int i = 0; i < this.password.length(); i++) System.out.print("*");
        simulateNetworkLatency();
        System.out.println("\n\nFacebook 로그인 성공");
        return true;
    }

    public boolean sendData(byte[] data) {
        boolean messagePosted = true;
        if (messagePosted) {
            System.out.println("Message: '" + new String(data) + "' 해당 글이 Facebook에 게시됨");
            return true;
        } else return false;
    }

    public void logOut() {
        System.out.println("User: '" + userName + "' 로그아웃 됨");
    }
```

```java
private void simulateNetworkLatency() {
    try {
        int i = 0;
        System.out.println();
        while (i < 10) {
            System.out.print(".");
            Thread.sleep(500);
            i++;
        }
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}
```

Twitter

```
public class Twitter extends Network {

    public Twitter(String userName, String password) {
        this.userName = userName;
        this.password = password;
    }

    public boolean logIn(String userName, String password) {
        System.out.println();
        System.out.println("유저 인증을 확인합니다.");
        System.out.println("유저명: " + this.userName);
        System.out.print("패스워드: ");
        for (int i = 0; i < this.password.length(); i++) System.out.print("*");
        simulateNetworkLatency();
        System.out.println("\n\n트위터에 정상적으로 로그인이 되었습니다.");
        return true;
    }

    public boolean sendData(byte[] data) {
        boolean messagePosted = true;
        if (messagePosted) {
            System.out.println("Message: '" + new String(data) + "' ] 트위터에 해당 글이 게시되었습니다.");
            return true;
        } else return false;
    }

    public void logOut() {
        System.out.println("User: '" + userName + "' 트위트에서 로그아웃 되었습니다.");
    }
```

```
private void simulateNetworkLatency() {
    try {
        int i = 0;
        System.out.println();
        while (i < 10) {
            System.out.print(".");
            Thread.sleep(500);
            i++;
        }
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}
```

TemplateMain

```
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    Network network = null;
    System.out.println("유저명 : ");
    String userName = scan.nextLine();
    System.out.println("패스워드 : ");
    String password = scan.nextLine();

    // 메시지 입력.
    System.out.println("메시지 입력: ");
    String message = scan.nextLine();

    System.out.println("메시지를 입력할 소셜 네트워크를 선택해주세요");
    System.out.println("1 - Facebook");
    System.out.println("2 - Twitter");
    int choice = Integer.parseInt(scan.nextLine());

    // 네트워크 연결 후 메시지 보내기.
    network.post(message);
}
```

```
유저명 :
kgh
패스워드 :
1234
메시지 입력:
이것은 광고입니다
메시지를 입력할 소셜 네트워크를 선택해주세요
1 - Facebook
2 - Twitter
1
유저 인증 확인
이름: kgh
비밀번호: ****
..........

Facebook 로그인 성공
Message: '이것은 광고입니다' 해당 글이 Facebook에 게시됨
User: 'kgh' 로그아웃 됨
```