KB

It's Your Life

with

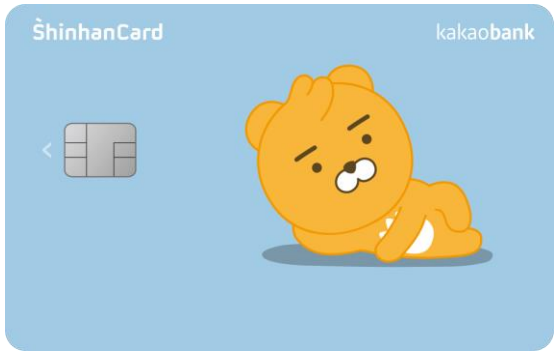
백엔드에서의

OOP 의 의미

OOP는 역할과 구현으로 간단하게 서비스를 변경 및 교체가 가능하다

여러분이 결제 서비스를 만든다고 해봅시다!

하지만 당연하게도
결제는 다양한 서비스를
지원해야 합니다



N Pay

toss pay

결제 라는 '역할'

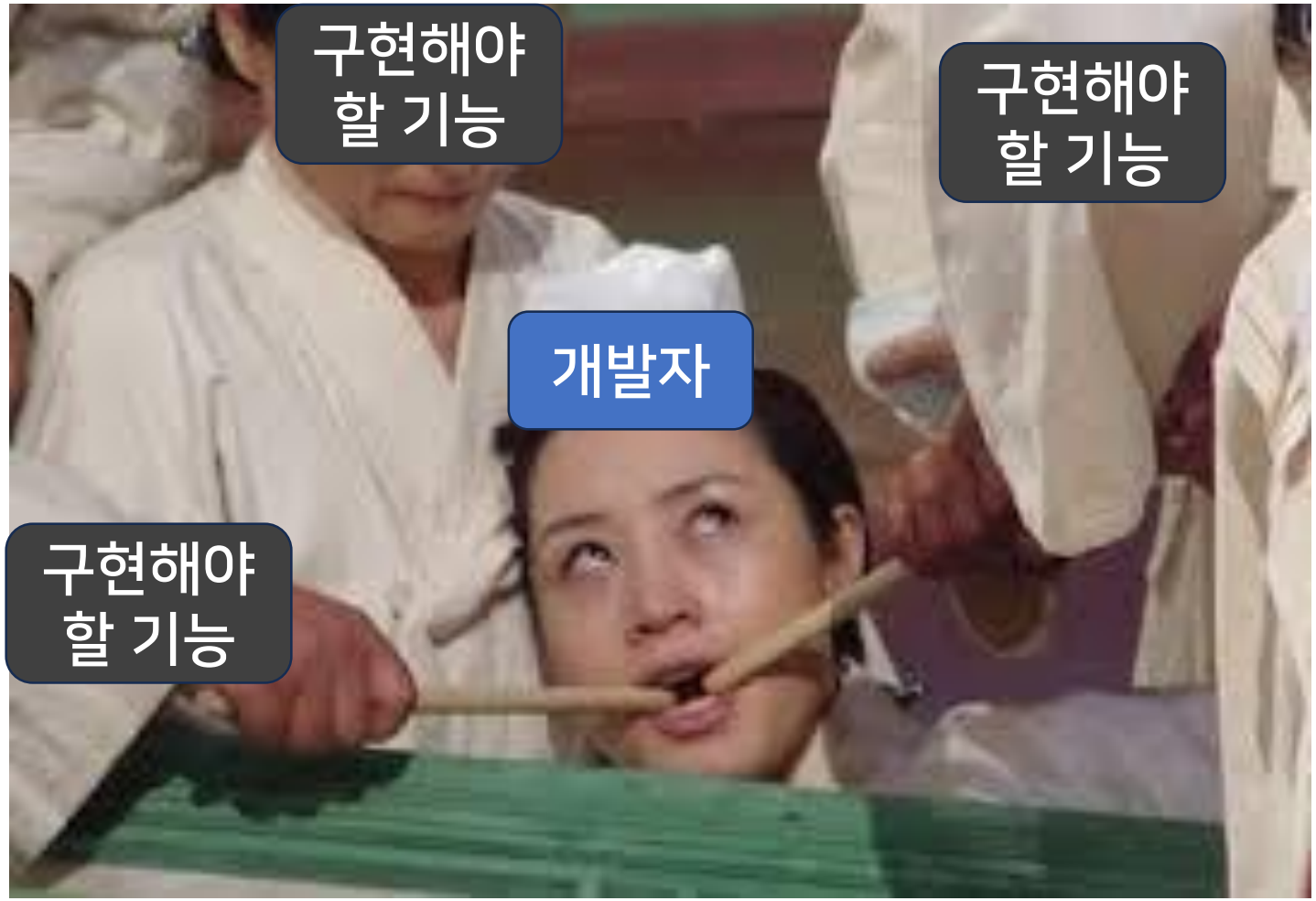
다양한 결제 수단이라는

‘구현’

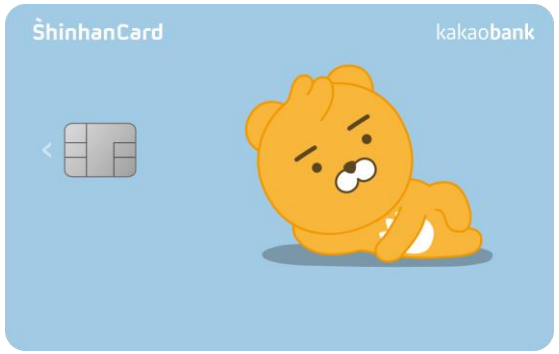
여기서

다형성이 떠오르시나요!?

결제라는 역할은!? 인터페이스를 사용



실제 구현은!? 클래스와 인스턴스를 사용

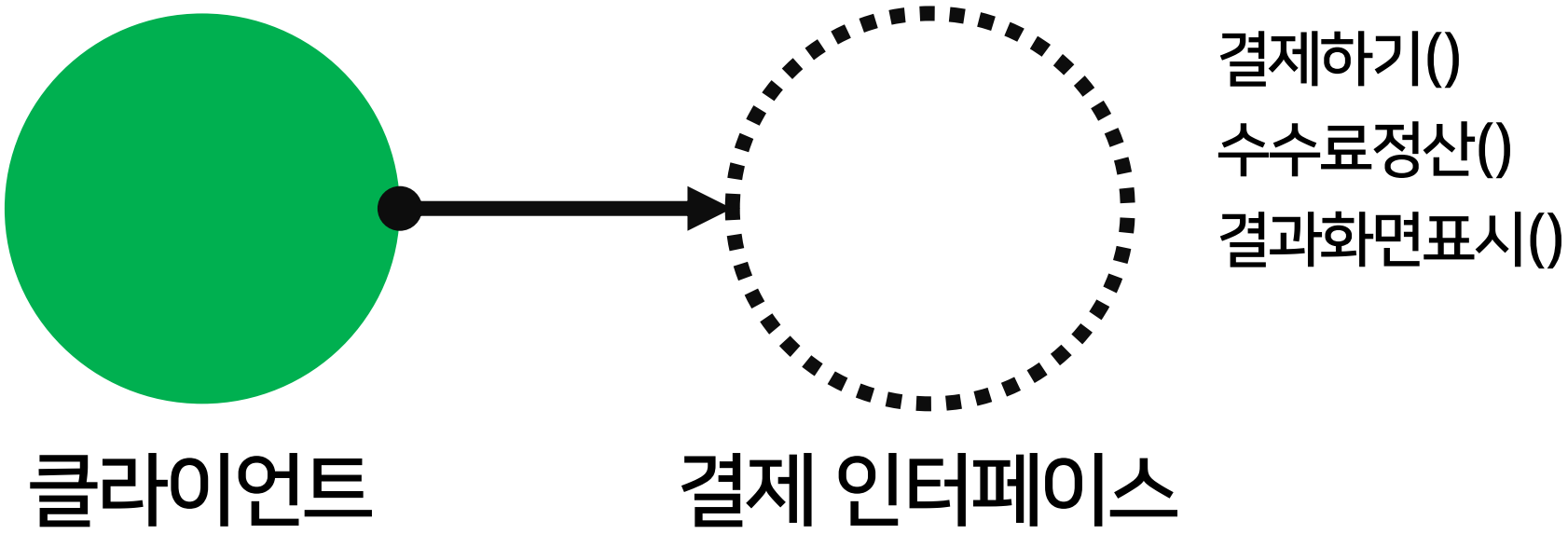


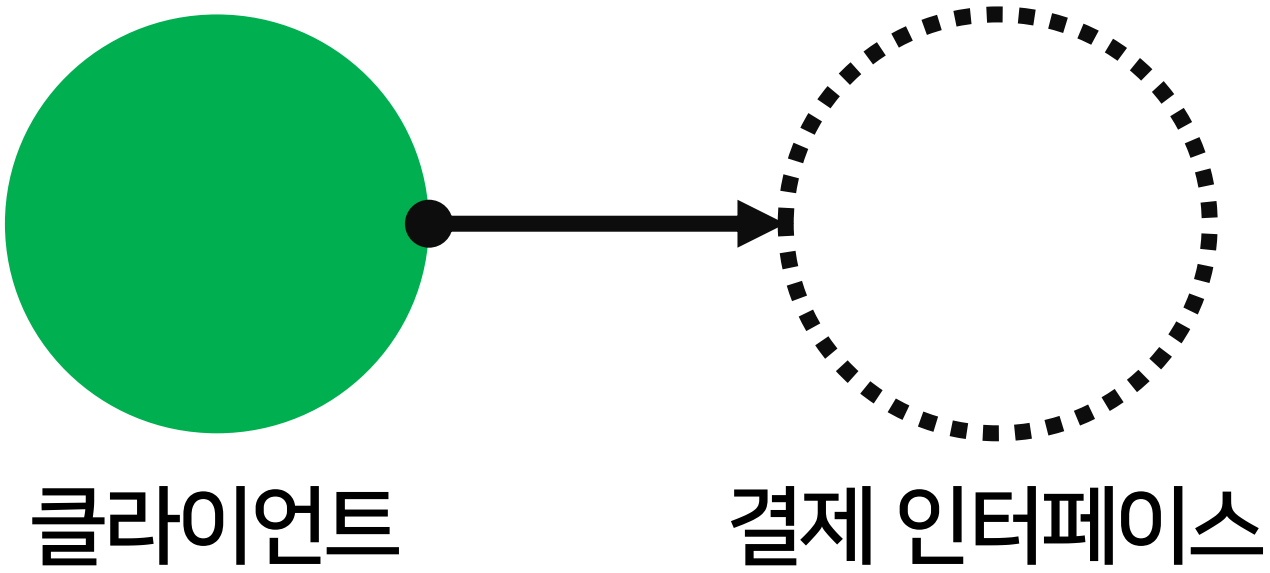
카드 결제
기능 인스턴스

카카오 결제
기능 인스턴스

N Pay

네이버 결제
기능 인스턴스


결제하기()
수수료정산()
결과화면표시()
클라이언트
결제 인터페이스

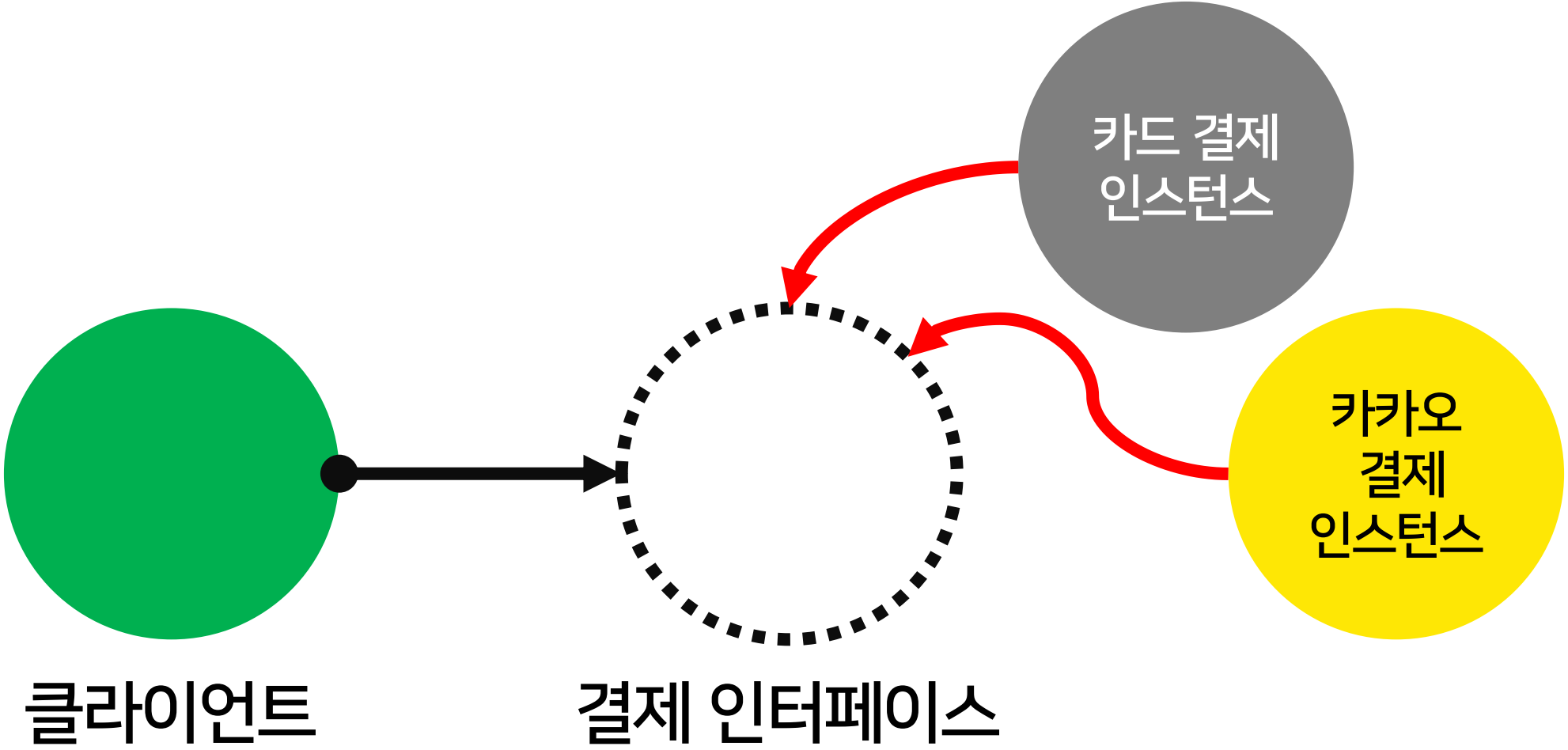


카드 결제 인스턴스

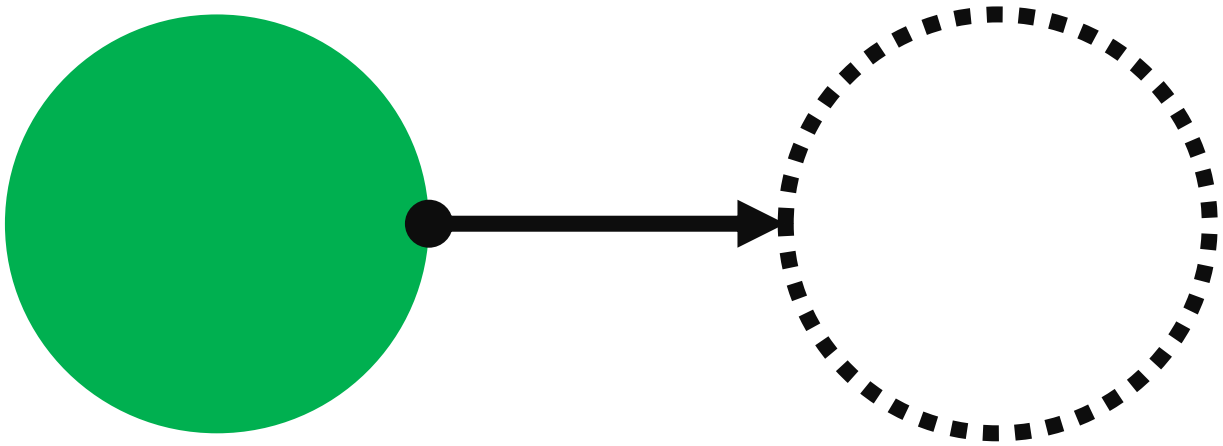
결제하기()
수수료정산()
결과화면표시()

카카오 결제 인스턴스

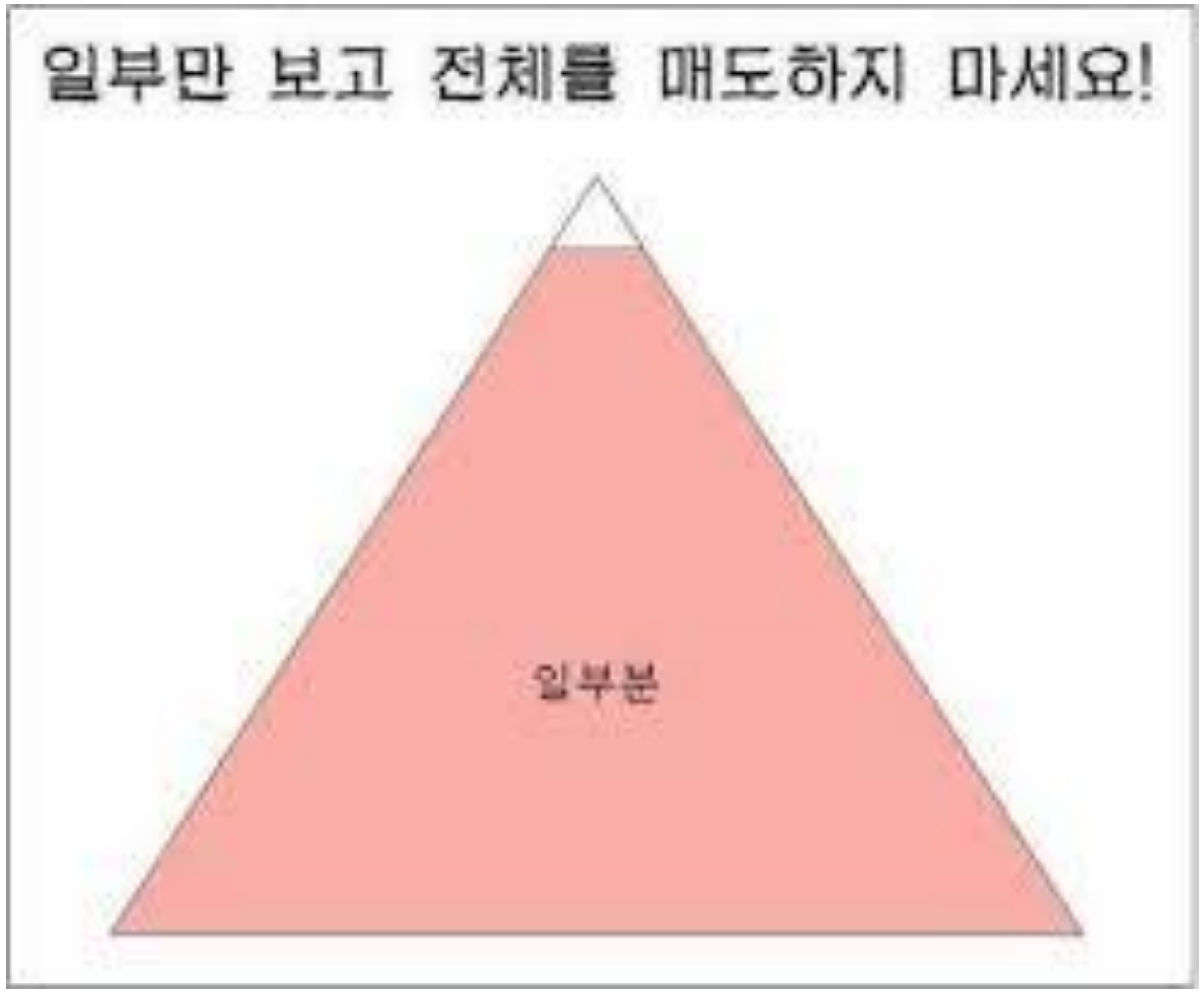
결제하기()
수수료정산()
결과화면표시()


카드 결제
인스턴스
카카오
결제
인스턴스
클라이언트
결제 인터페이스

카드 결제
인스턴스

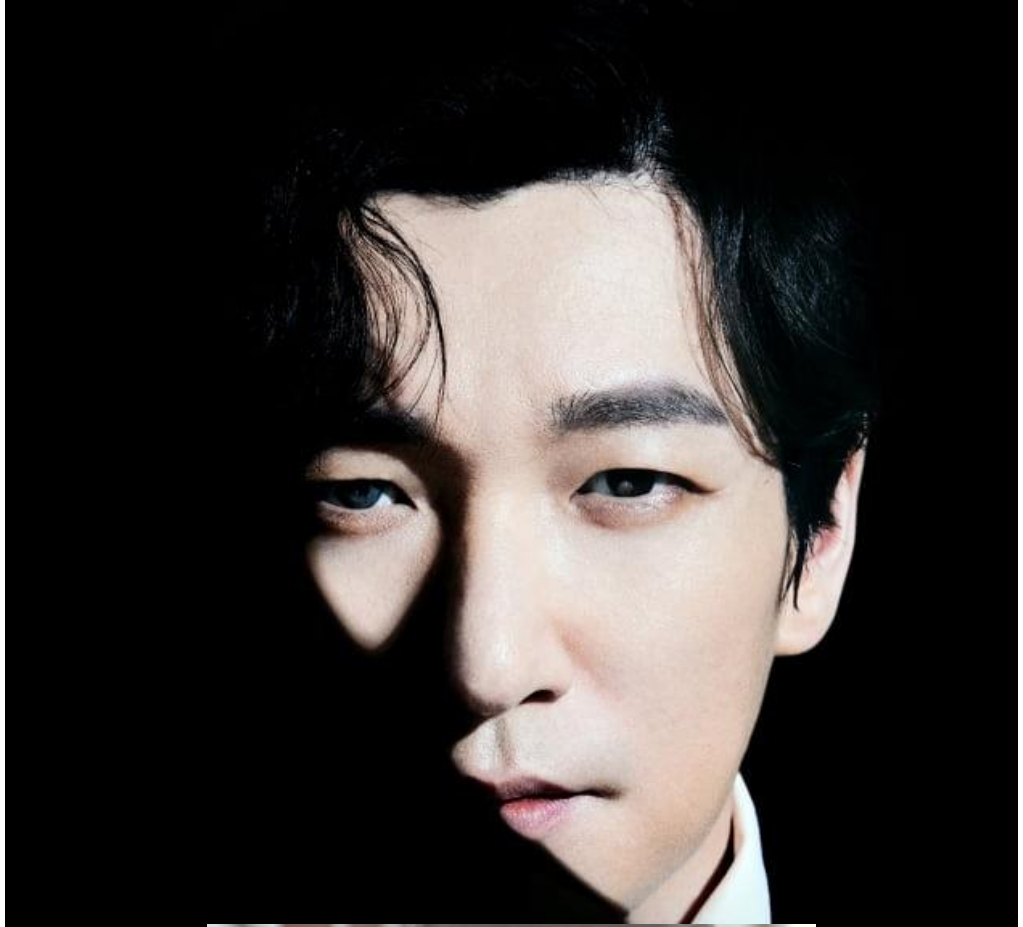


카카오
결제
인스턴스



OOP 와 다형성을 통해서
결제 시스템 전체를
변경할 필요 X

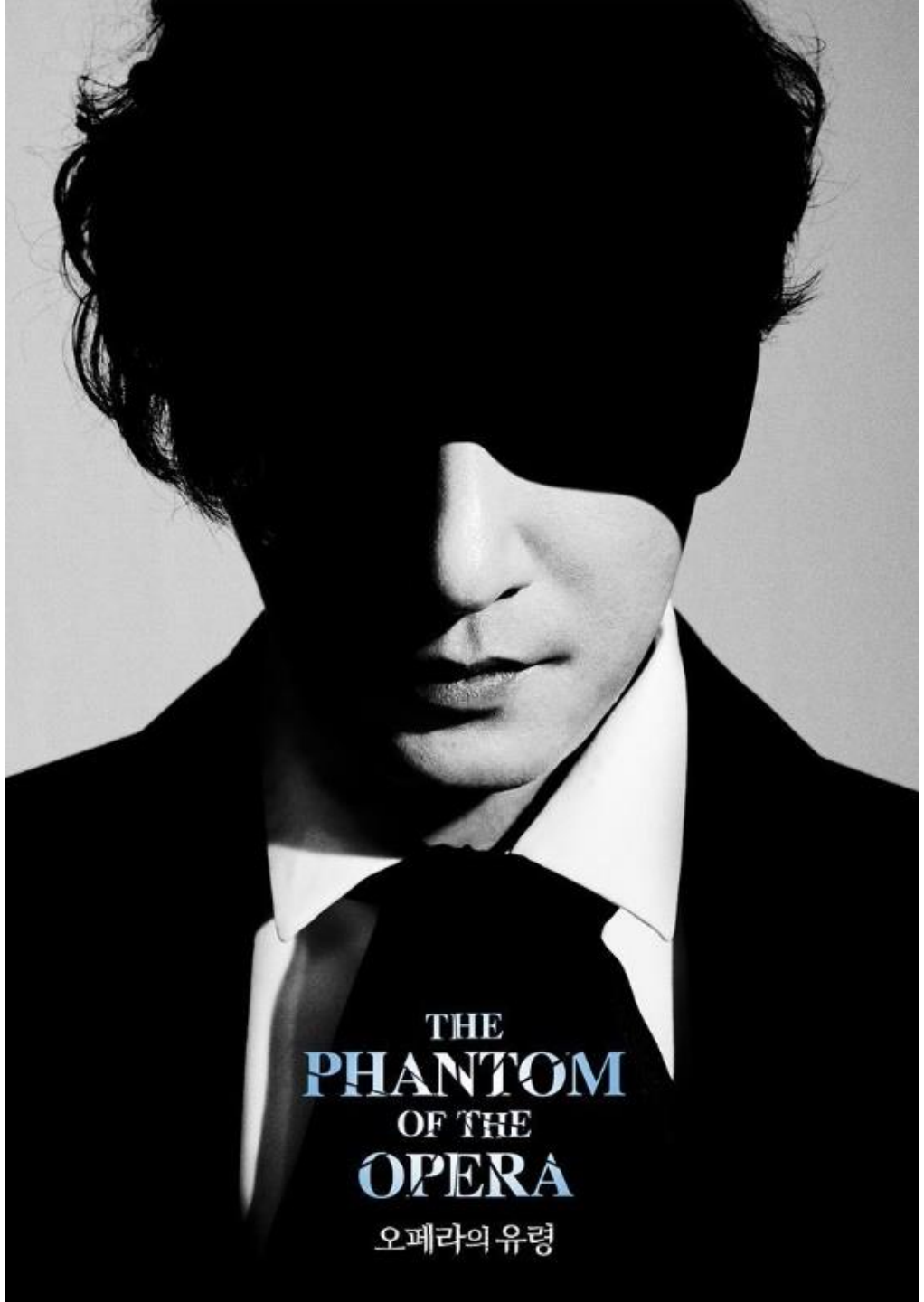
결제 시스템은 그대로 두고
인스턴스만 변경!



조승우라는 배우에 맞춰서
오페라를 설계한다면!?

조승우씨가 아프면!?
조승우씨가 안 한다고 하면?

그리고 조승우씨에게 맞춰진
오페라 자체의 내용도
쉽게 변경이 가능할까요?



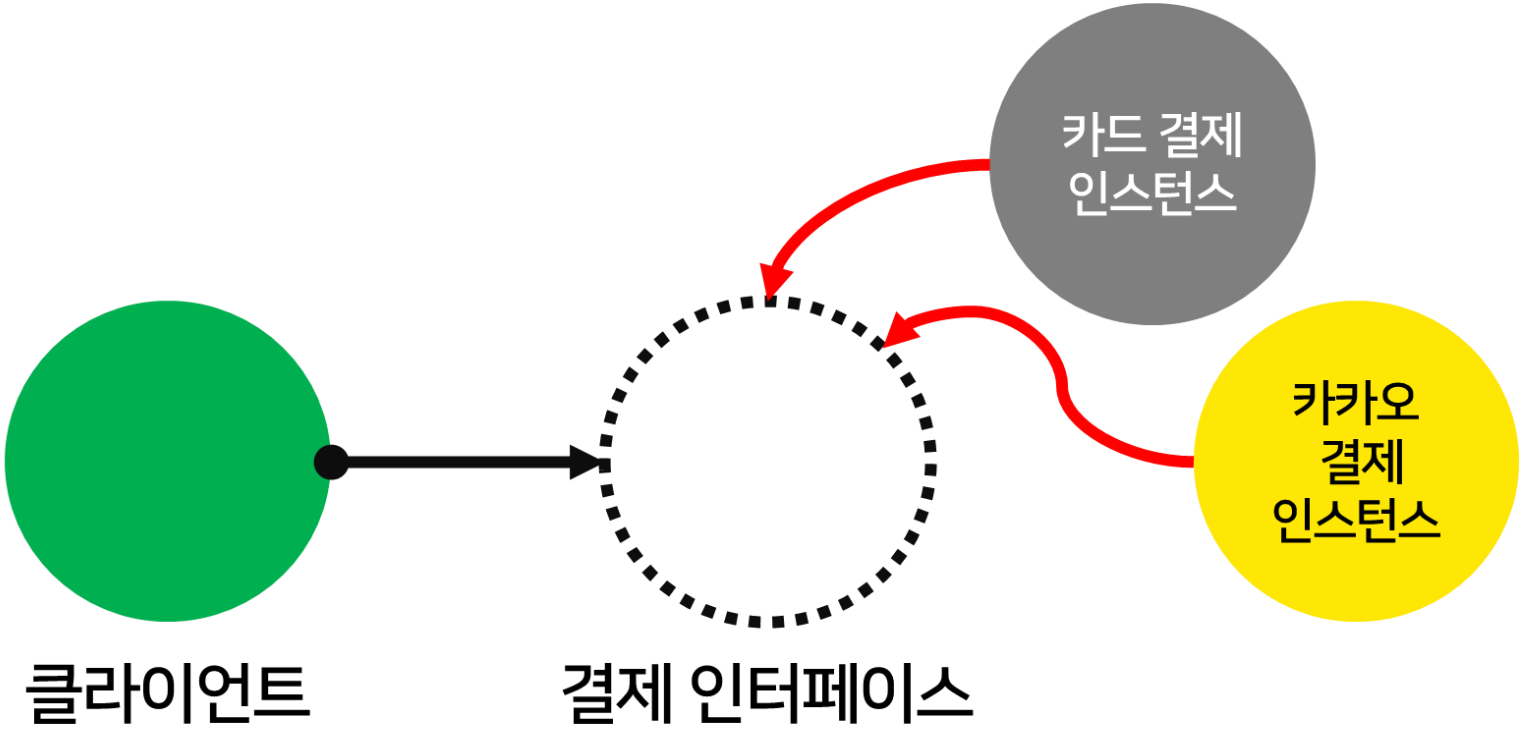
유령이라는 역할에 집중해서
오페라를 설계한다면!?

유령 역할을 잘할 수 있는 배우만
잘 섭외하면 됩니다!!

어떤 배우가 아파도? → 대타 가능
극 자체의 내용도? → 쉽게 변경 가능

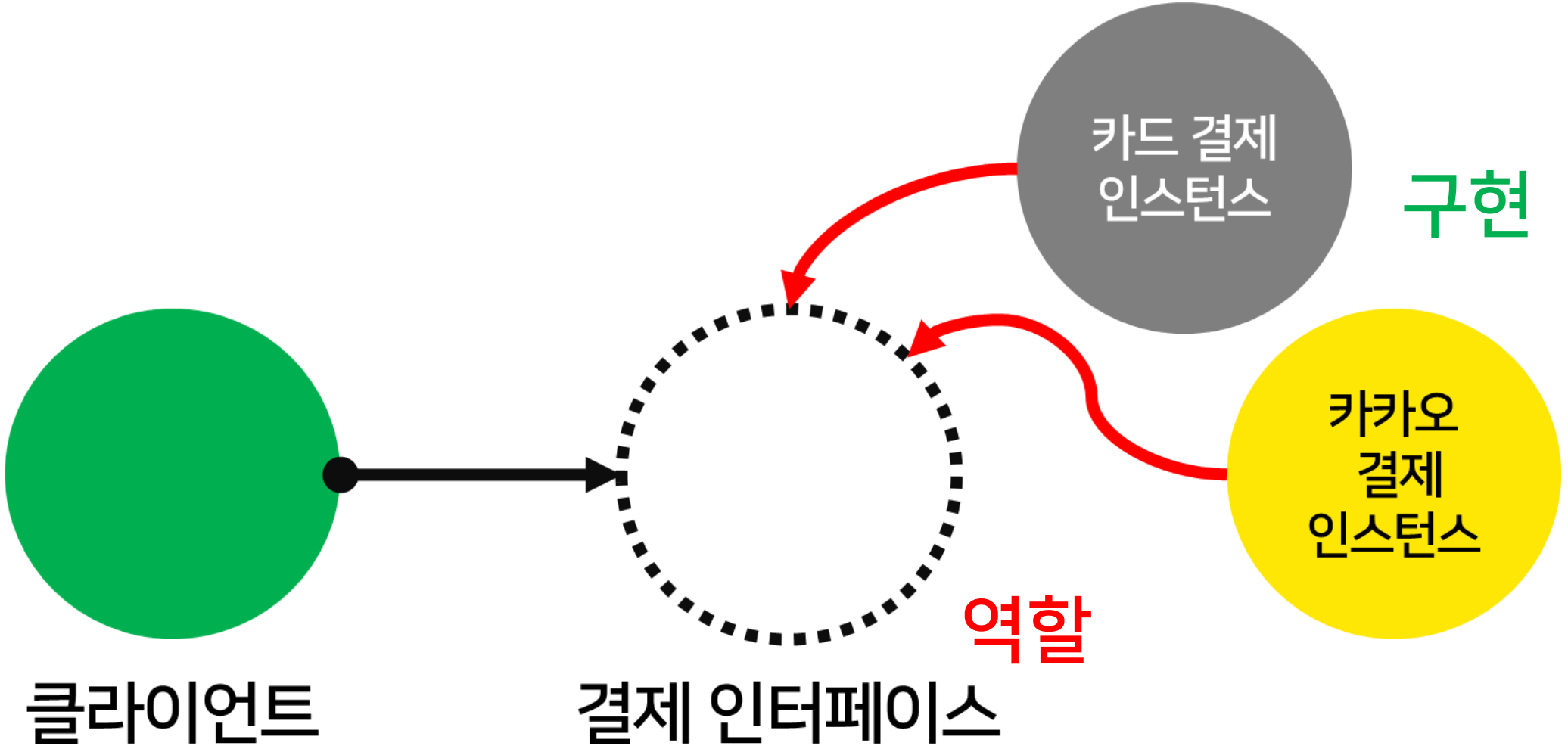
이걸 백엔드와 서비스 세상에 접목시켜 보면!?

왜, OOP 와 다형성을 이렇게 강조하는지
느낌이 오실 겁니다!!



실제 코드로

확인!


카드 결제
인스턴스
구현
카카오
결제
인스턴스
역할
클라이언트
결제 인터페이스

구현 파트

```java
public interface Pay {
    void pay(int amount);
}
```

페이 서비스의
구현 파트 인터페이스가 될
Pay 선언

```java
public class KBPay implements Pay{  1 usage    new *
    @Override  1 usage    new *
    public void pay(int amount) {
        System.out.println("KB Pay 시스템과 연결 합니다");
        System.out.println(amount + "원 결제를 시도합니다");
        System.out.println("결제 성공!");
    }
}
```

Pay 인터페이스의 설계에 맞게
클래스를 '구현'

```java
public class KaKaoPay implements Pay{  1 usage    new *
    @Override  1 usage    new *
    public void pay(int amount) {
        System.out.println("KaKao Pay 시스템과 연결 합니다");
        System.out.println(amount + "원 결제를 시도합니다");
        System.out.println("결제 성공!");
    }
}
```

Pay 인터페이스의 설계에 맞게
클래스를 '구현'

역할 파트

```java
public class PaySystem {  2 usages    new *
    private Pay pay;  2 usages

    public void setPay(Pay pay) {  2 usages
        this.pay = pay;
    }

    public void payment(int amount) {  1 usage
        System.out.println("결제를 시작합니다.");
        pay.pay(amount);
    }
}
```

어떤 결제 '구현'이 들어올지 알 수 없으므로 최상위 부모인 Pay 타입으로 멤버 변수를 선언!

'구현'이 결정이 되면 해당 구현을 멤버 변수에 할당하는 set 메서드

```java
public class PaySystem {  2 usages    new *
    private Pay pay;  2 usages

    public void setPay(Pay pay) {  2 usages    new
        this.pay = pay;
    }

    public void payment(int amount) {  1 usage
        System.out.println("결제를 시작합니다.");
        pay.pay(amount);
    }
}
```

결제 기능은 어떤 '구현'이 와도
해당 구현의 결제 구현에 따르면 되므로
매개변수로 결제 금액만 받아서 전달

실제 결제는 구현 파트에 구현된
pay() 메서드가 알아서 실행합니다!

인터페이스를 구현한 상태이므로
메서드는 캐스팅 없이 실행 시켜도
오버라이딩 우선 법칙으로 인해서
문제가 전혀 발생하지 않습니다!

운영 파트

```java
public class PayMain {  new *
    public static void main(String[] args) {  new *
        PaySystem paySystem = new PaySystem();
        Scanner scanner = new Scanner(System.in);
        int option;
        int amount;
```

역할을 담당할 PaySystem 인스턴스

사용자 입력 및 입력을 저장할 변수 선언

```java
System.out.print("결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : ");
option = scanner.nextInt();

System.out.print("결제 금액을 입력하세요 : ");
amount = scanner.nextInt();

if (option == 1) {
    paySystem.setPay(new KBPay());
} else if (option == 2) {
    paySystem.setPay(new KaKaoPay());
}

paySystem.payment(amount);
```

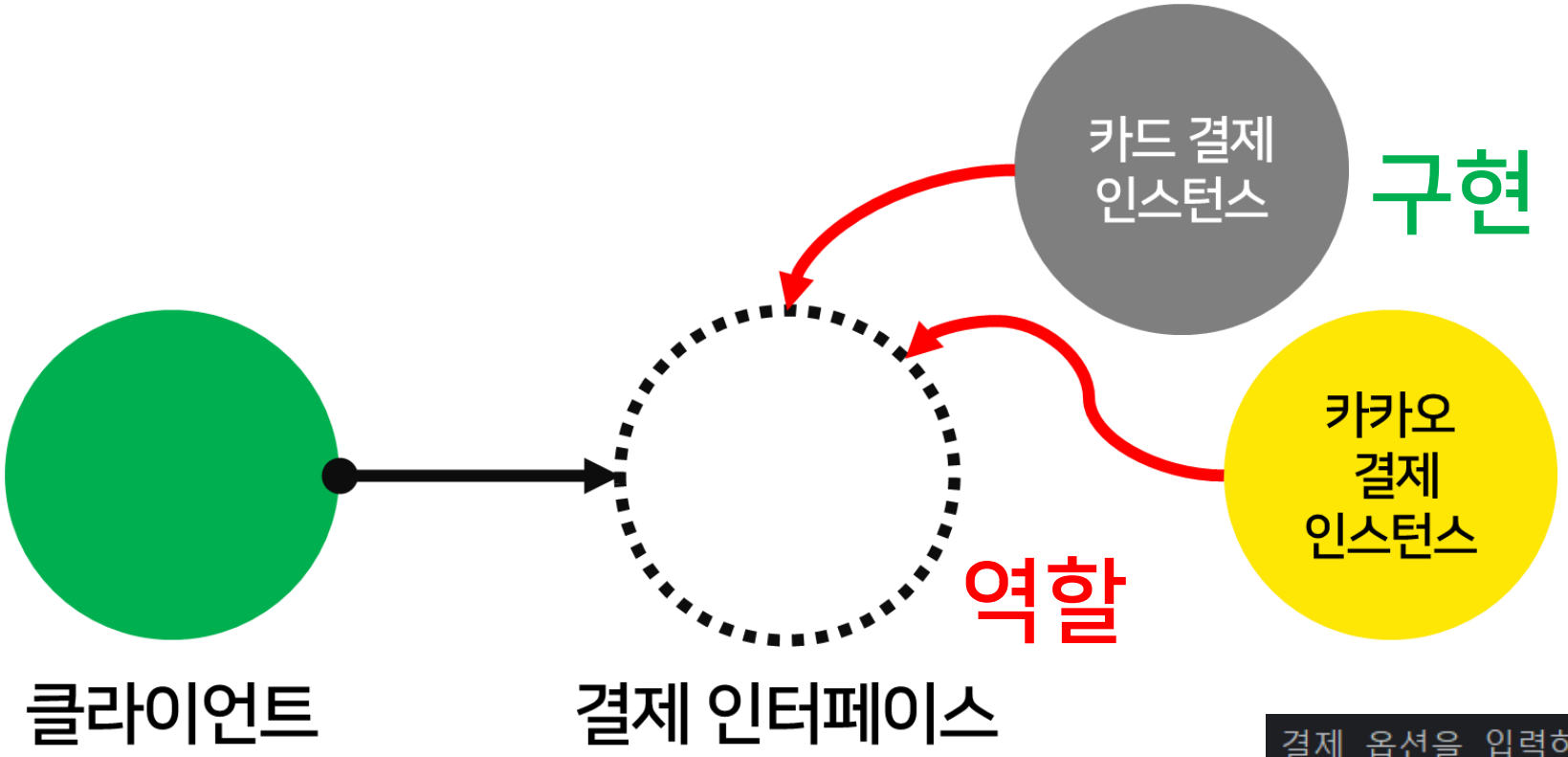
사용자로부터 필요한 정보를 입력 받기

사용자 입력에 따라 pay 의 '구현' 설정

```
System.out.print("결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : ");
option = scanner.nextInt();

System.out.print("결제 금액을 입력하세요 : ");
amount = scanner.nextInt();

if (option == 1) {
    paySystem.setPay(new KBPay());
} else if (option == 2) {
    paySystem.setPay(new KaKaoPay());
}

paySystem.payment(amount);
```

결정 된 구현에서 결제 진행!

```
결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : 1
결제 금액을 입력하세요 : 3000
결제를 시작합니다.
KB Pay 시스템과 연결 합니다
3000원 결제를 시도합니다
결제 성공!
```




```
결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : 1
결제 금액을 입력하세요 : 3000
결제를 시작합니다.
KB Pay 시스템과 연결 합니다
3000원 결제를 시도합니다
결제 성공!
```

```
결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : 2
결제 금액을 입력하세요 : 1000
결제를 시작합니다.
KaKao Pay 시스템과 연결 합니다
1000원 결제를 시도합니다
결제 성공!
```

실습, 네이버 페이 추가하기!

- 현재의 코드에 네이버페이 서비스를 추가해 보세요!
- Pay 인터페이스를 구현하여 NaverPay 구현을 만들어 주세요
- Pay 인터페이스의 pay() 메서드를 오버라이드하여, 해당 메서드가 실행되면 “Naver Pay 시스템과 연결합니다“ → “xxx 원 결제를 시도합니다“ → “결제 성공!” 메시지가 뜨면 됩니다!

실습, 네이버 페이 추가하기!

- 운영 클래스에서 3번 옵션을 입력하면 네이버 페이에서 결제가 발생하도록 코드를 수정해 주세요!

코드

refactoring

```
System.out.print("결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : ");
option = scanner.nextInt();
System.out.print("결제 금액을 입력하세요 : ");
amount = scanner.nextInt();
```

입력을 받는 부분도
PaySystem 부분이므로
메서드로 만들어 주기

결제 옵션을 설정하는 파트와
결제 금액을 입력 받는 파트를
나누어서 메서드로 만들기

```java
public void selectPay() {  1 usage   new *
    System.out.print("결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : ");
    int option = scanner.nextInt();
    findPay(option);
}
```

결제 옵션을 입력받아
결제 인스턴스를 결정하는
findPay() 로 넘겨주는
selectPay() 메서드

```java
public class PaySystem {  2 usages   Tetz *
    private static final int KB_PAY = 1;  1 usage
    private static final int KAKAO_PAY = 2;  1 usage
```

코드 가독성을 위해
옵션 코드는 상수로 지정

```java
public void findPay(int option) {  1 usag
    if (option == KB_PAY) {
        this.setPay(new KBPay());
    } else if (option == KAKAO_PAY) {
        this.setPay(new KaKaoPay());
    }
}
```

전달 받은 option 의 값에 따라
결제 인스턴스를 결정하는 메서드

```java
public void selectAmount() {  1 usage   new *
    System.out.print("결제 금액을 입력하세요 : ");
    int amount = scanner.nextInt();
    payment(amount);
}

public void payment(int amount) {  1 usage   Tetz
    System.out.println("결제를 시작합니다.");
    pay.pay(amount);
}
```

결제 금액을 입력받아
payment() 로 넘기는 메서드

전달 받은 결제 금액 만큼을
결정 된 결제 인스턴스에서 진행하는
메서드

```java
public class PayMain {  Tetz *
    public static void main(String[] args) {
        PaySystem paySystem = new PaySystem();


        paySystem.selectPay();
        paySystem.selectAmount();
    }
}
```

```java
public class PayMain {  new *
    public static void main(String[] args) {  new *
        PaySystem paySystem = new PaySystem();
        Scanner scanner = new Scanner(System.in);
        int option;
        int amount;

        System.out.print("결제 옵션을 입력하세요 (1. KB 페이 / 2. 카카오페이) : ");
        option = scanner.nextInt();

        System.out.print("결제 금액을 입력하세요 : ");
        amount = scanner.nextInt();

        if (option == 1) {
            paySystem.setPay(new KBPay());
        } else if (option == 2) {
            paySystem.setPay(new KaKaoPay());
        }

        paySystem.payment(amount);
```

Refactoring 전후
코드 차이 비교!

```java
public class PayMain {  Tetz *
    public static void main(String[] args) {
        PaySystem paySystem = new PaySystem();

        paySystem.selectPay();
        paySystem.selectAmount();
    }
}
```

실습, 드라이버 시스템 만들기

- 위의 개념을 이용해서 드라이버 시스템을 만들어 봅시다!
- Car 라는 인터페이스는 openDoor() / drive() 추상 메서드를 제공합니다
- Car 인터페이스를 구현하여 K5Car / Grand(그랜져) / G70 클래스를 만들어 주시면 됩니다!
- openDoor 메서드는 차종에 따라
  - K5, 차 문을 엽니다 / 그랜져, 차 문을 엽니다 / G70, 차 문을 엽니다
  - 위와 같은 내용을 출력 합니다

실습, 드라이버 시스템 만들기

- drive() 메서드는 차종에 따라
  - K5 주행 시작 → K5 주행 종료 / 그랜져 주행 시작 → 그랜져 주행 종료 / G70 주행 시작 → G70 주행 종료
  - 위와 같은 내용을 출력 합니다

실습, 드라이버 시스템 만들기

- 프로그램이 시작 되면
  - 운전하고 싶은 차를 선택하세요. (1. K5 / 2. 그랜져 / 3. G70) :
  - 라는 메시지가 뜨며 사용자의 입력에 따라 운전하는 차의 종류가 달라집니다!
- 역할을 담당하는 Driver 클래스는 아래와 같은 코드를 가지고 있으며, 아래와 같은 코드의 주석 부분을 완성하여 주세요.

```java
public class Driver {  2 usages    new *
    private Car car;  1 usage
    Scanner scanner = new Scanner(System.in);  no usages

    public void setCar(Car car) {  no usages    new *
        // 코드를 완성하세요
    }

    public void selectCar() {  1 usage    new *
        // 코드를 완성 하세요
        // 해당 메서드 마지막에 drive() 메서드가 실행 됩니다
        drive();
    }

    public void drive() {  1 usage    new *
        car.drive();
    }
}
```

실습, 드라이버 시스템 만들기

- 운영 클래스인 CarMain 의 코드는 아래와 같습니다.

```
public class CarMain {  new *
    public static void main(String[] args) {
        Driver driver = new Driver();


        driver.selectCar();
    }
}
```

실습, 드라이버 시스템 만들기

- 실행 결과는 아래와 같으면 됩니다!

```
운전하고 싶은 차를 선택하세요. (1. K5 / 2. 그랜져 / 3. G70) : 1
K5가 주행을 시작 합니다
K5 주행 종료
```

```
운전하고 싶은 차를 선택하세요. (1. K5 / 2. 그랜져 / 3. G70) : 2
그랜져가 주행을 시작 합니다
그랜져 주행 종료
```

```
운전하고 싶은 차를 선택하세요. (1. K5 / 2. 그랜져 / 3. G70) : 3
G70 주행을 시작 합니다
G70 주행 종료
```

실습, 드라이버 시스템 만들기 – 도전 버전!

- 위의 실습 코드에서 아래의 내용을 추가해 주세요!
- K5Car 는 private int oil 이라는 멤버 변수를 가집니다
  - K5Car 생성되는 시점에 생성자를 통해서 oil 값을 초기화 합니다
- K5Car 는 EFFICIENCY 상수를 가지고 있으며 값은 2 입니다
- K5 를 선택하면, 처음에 최초 oil 의 양을 사용자를 통해 입력 받습니다!

실습, 드라이버 시스템 만들기 – 도전 버전!

- K5Car 에서 drive 메서드가 실행되면 연비(EFFICIENCY)당 1km 를 주행할 수 있습니다.
- 또한 1km 를 주행할 때마다 현재까지 주행한 거리를 “K5 가 x km 를 주행했습니다“ 를 출력해 주세요.
- 주행이 종료되면 “K5 가 최종 주행한 거리는 x km 입니다“ 를 출력 하면서 프로그램을 종료합니다.
- 연비 미만의 oil 이 남은 경우 바로 주행을 종료하면 됩니다.

실습, 드라이버 시스템 만들기 – 도전 버전!

- 실행 화면은 다음과 같으면 됩니다!

```
운전하고 싶은 차를 선택하세요. (1. K5 / 2. 그랜져 / 3. G70) : 1
K5 에 주유할 기름의 양을 입력하세요 : 5
K5가 주행을 시작 합니다
K5 가 1km 를 주행했습니다.
K5 가 2km 를 주행했습니다.
K5 가 최종 주행한 거리는 2km 입니다
K5 주행 종료
```