dongguk UNIVERSITY

객체지향프로그래밍

실습강의 9주차

실습강의 소개

- **실습 진행 방법**
  - 간단한 이론 복습 및 해당주차 실습과제 설명
  - 실습 후 보고서와 소스코드를 압축하여 수요일 자정(23:59)까지
    꼭!! 이클래스 제출(**이메일 제출 불가, 반드시 이클래스를 통해 제출**)
  - 실습 과제 제출 기한 엄수(제출 기한 이후로는 0점 처리)
  - **보고서는 출력하여 수업 시작 전에 제출(대면으로 실습 수업 진행 시)**
- **Q & A**
  - **이클래스 및 실습조교 이메일을 통해 질의 응답**
  - **이메일 제목 : [객체지향프로그래밍_홍길동] *본인 과목명과 성명 꼭 작성!**
  - **실습조교 메일 주소 : pmj6823@dgu.ac.kr**
  - **수업용 깃허브 조직 : https://github.com/23-2-Object-Oriented-Programming**

실습강의 소개

- **실습 보고서**
  - 문제 분석, 프로그램 설계 및 알고리즘, 소스코드 및 주석, 결과 및 결과 분석, 소감
- **제출 방법**
  - 보고서, 소스코드, 실행파일을 1개의 파일로 압축하여 e-class "과제" 메뉴를 통해 제출
    - **"이름학번실습주차.zip" 형태로 제출(e.g. :김동국19919876실습7.zip)**
    - 파일명에 공백, 특수 문자 등 사용 금지
  - 대면 수업일 경우 출력한 보고서를 실습 시간에 제출
- **유의 사항**
  - 보고서의 표지에는 학과, 학번, 이름, 담당 교수님, 제출일자 반드시 작성
  - 정해진 기한내 제출
    - 기한 넘기면 0점 처리
    - e-class가 과제 제출 마지막 날 오류로 동작하지 않을 수 있으므로, 최소 1~2일 전에 제출
    - **당일 e-class 오류로 인한 미제출은 불인정**
  - 소스코드, 보고서를 자신이 작성하지 않은 경우 **실습 전체 점수 0점 처리**
  - **Eclipse, 동국대학교 Shastra**를 사용하여 실습 진행

보고서 작성 방법

- **보고서 양식**
  - 문제 분석: 실습 문제에 대한 요구 사항 파악, 해결 방법 등 기술
  - 프로그램 설계 및 알고리즘
    - 해결 방법에 따라 프로그램 설계 및 알고리즘 등 기술
    - e.g.) 문제 해결 과정 및 핵심 알고리즘 기술
  - 소스코드 및 주석
    - 소스코드와 그에 해당하는 주석 첨부
    - 각각의 함수가 수행하는 작업, 매개변수, 반환 값 등을 명시
    - **소스코드 전체 첨부(소스코드 화면 캡처X, 소스코드는 복사/붙여넣기로 첨부)**
  - 결과 및 결과 분석
    - 결과 화면을 캡처하여 첨부, 해당 결과가 도출된 이유와 타당성 분석
  - 소감
    - 실습 문제를 통해 습득할 수 있었던 지식, 느낀 점 등을 기술

업캐스팅

- **업캐스팅(upcasting)**
  - 서브 클래스의 레퍼런스를 슈퍼 클래스 레퍼런스에 대입하는 것
  - 슈퍼 클래스 레퍼런스로 서브 클래스 객체를 가리키게 되는 현상
  - 별도의 지시어 없이 자동으로 형변환이 일어남

```
class Person { }
class Student extends Person { }

Person p;
Student s = new Student();
p = s; // 업캐스팅
```

```
p.grade = "A"; // grade는 Person의
               // 멤버가 아니므로
               // 컴파일 오류
```

[연습 1] 업캐스팅 사례

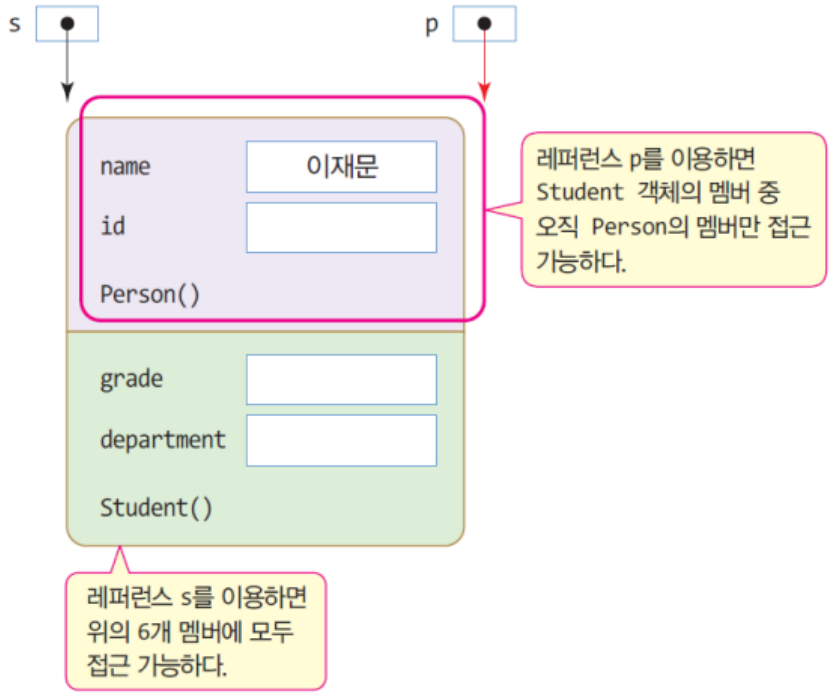
- class UpcastingEx의 main 함수를 실행시킬 경우 발생하는 컴파일 오류를 찾아 내고 이유를 설명하라.

```
public class UpcastingEx {
    public static void main(String[] args) {
        Person p;
        Student s = new Student("이재문");
        p = s; // 업캐스팅 발생
        System.out.println(p.name); // 오류 없음
        p.grade = "A"; // 컴파일 오류
        p.department = "Com"; // 컴파일 오류
    }
}
```

```
class Person {
    String name;
    String id;
    public Person(String name) {
        this.name = name;
    }
}
```

```
class Student extends Person {
    String grade;
    String department;
    public Student(String name) {
        super(name);
    }
}
```

[연습 1] 업캐스팅 사례



다운 캐스팅

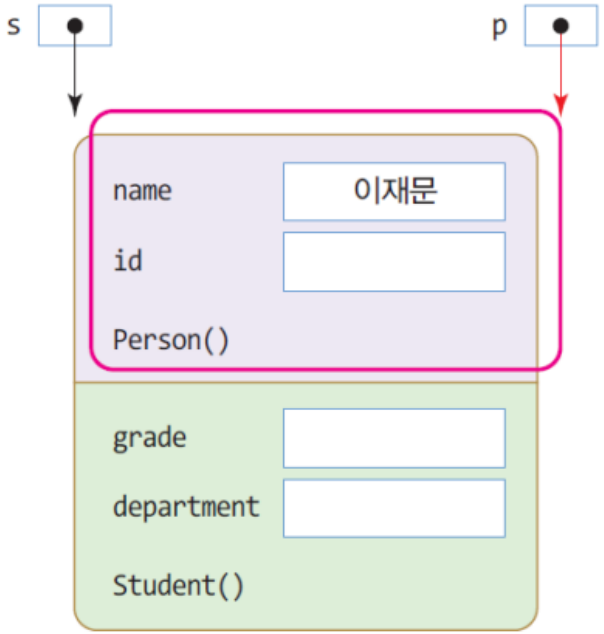
- **다운캐스팅(downcasting)**
  - 슈퍼 클래스 레퍼런스를 서브 클래스 레퍼런스에 대입
  - 업캐스팅된 것을 다시 원래대로 되돌리는 것
  - 반드시 명시적 타입 변환 지정

```
class Person { }
class Student extends Person { }
Person p = new Student("이재문"); // 업캐스팅
Student s = (Student)p; // 다운캐스팅, 강제타입변환
```

[연습 2] 다운캐스팅 사례

- 아래 코드를 작성하고 실행하라.

```
public class DowncastingEx {
    public static void main(String[] args) {
        Person p = new Student("이재문"); // 업캐스팅
        Student s;
        s = (Student) p; // 다운캐스팅
        System.out.println(s.name); // 오류 없음
        s.grade = "A"; // 오류 없음
    }
}
```



instanceof 연산자

- **instanceof 연산자**
  - 레퍼런스가 가리키는 객체의 타입을 식별
  - instanceof 연산자 사용 사례

```
Person p = new Professor();
if(p instanceof Person) // true
if(p instanceof Student) // false. Student를 상속받지 않기 때문
if(p instanceof Researcher) // true
if(p instanceof Professor) // true
```

오버라이딩과 오버로딩

- **오버라이딩**
  - 서브 클래스에서 슈퍼 클래스의 메소드를 중복 작성하는 것
  - 슈퍼 클래스의 메소드가 무력화되고, 항상 서브 클래스에 오버라이딩한 메소드가 실행됨
    - 동적 바인딩
  - 슈퍼 클래스의 메소드의 원형(메소드 이름, 인자 타입 및 개수, 리턴 타입) 동일하게 작성
  - 다형성을 실현하는 도구 중 하나
- **오버로딩**
  - 같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성
  - 메소드 이름은 반드시 동일하게 작성하며 매개 변수의 타입 및 개수가 달라야함
    - 정적 바인딩
  - 메소드 호출 명령어의 매개 변수를 보고 컴파일 시에 정적으로 바인딩됨.

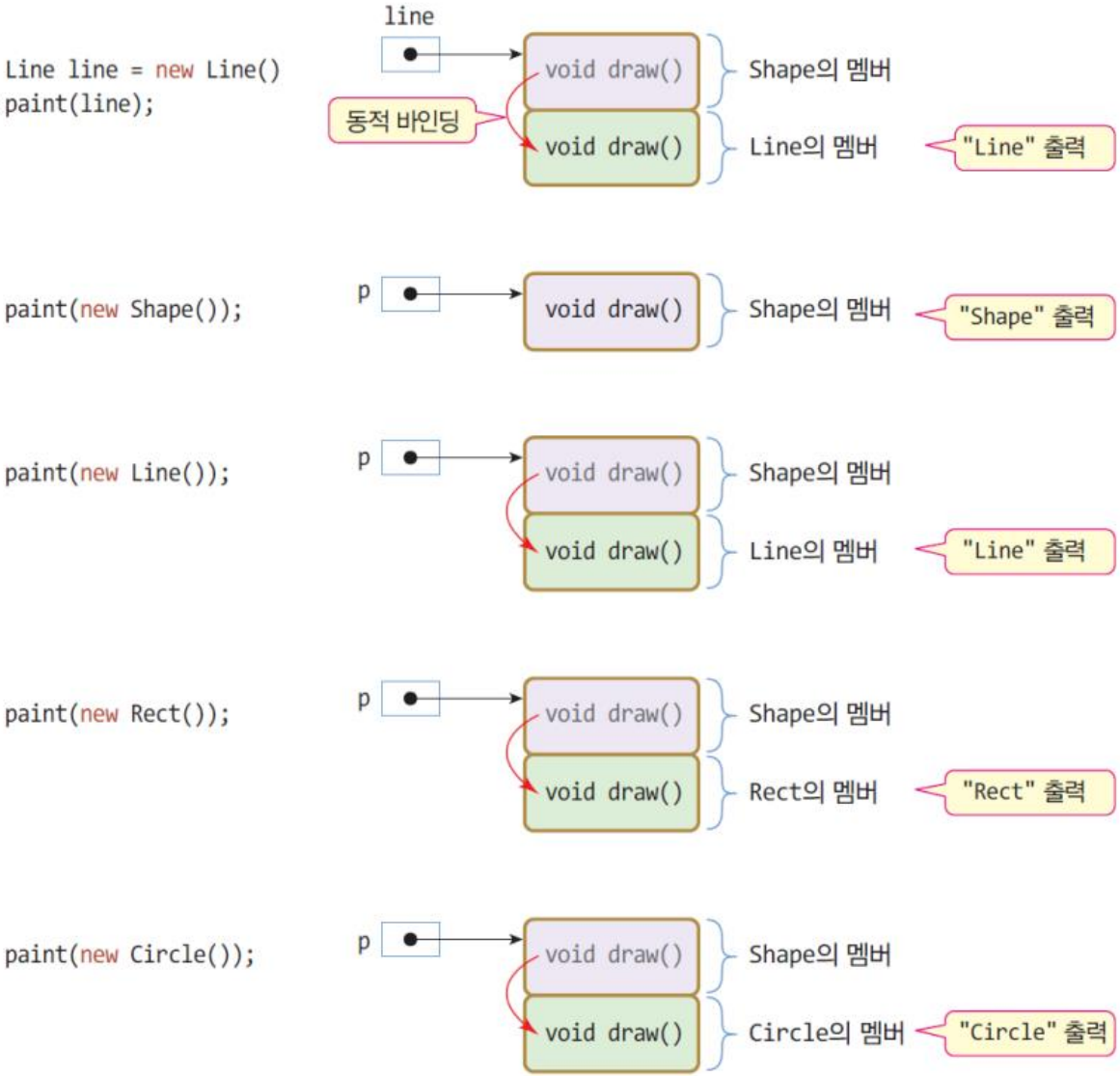
[연습 3] 오버라이딩 예시

- 아래 코드를 작성하고 main에서 각 메소드들이 어떻게 동적 바인딩되는지 설명하라.

```
class Shape { // 도형의 슈퍼 클래스
    public void draw() {
        System.out.println("Shape");
    }
}
class Line extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Line");
    }
}
class Rect extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Rect");
    }
}
class Circle extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Circle");
    }
}
```

```
public class MethodOverridingEx {
        static void paint(Shape p) {
       // p가 가리키는 객체에 오버라이딩된 메소드 호출
        p.draw();
    }
    public static void main(String[] args) {
        Line line = new Line();
        // Line의 draw() 실행. "Line" 출력
        paint(line);
        // Shape의 draw() 실행. "Shape" 출력
        paint(new Shape());
        // 오버라이딩된 메소드 Line의 draw() 실행
        paint(new Line());
        // 오버라이딩된 메소드 Rect의 draw() 실행
        paint(new Rect());
        // 오버라이딩된 메소드 Circle의 draw() 실행
        paint(new Circle());
    }
}
```

[연습 3] 오버라이딩 예시



오버로딩과 오버라이딩

비교 요소	메소드 오버로딩	메소드 오버라이딩
선언	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 재작성
관계	동일한 클래스 내 혹은 상속 관계	상속 관계
목적	이름이 같은 여러 개의 메소드를 중복 선언하여 사용의 편리성 향상	슈퍼 클래스에 구현된 메소드를 무시하고 서브 클래스에서 새로운 기능의 메소드를 재정의하고자 함
조건	메소드 이름은 반드시 동일함. 메소드의 인자의 개수나 인자의 타입이 달라야 성립	메소드의 이름, 인자의 타입, 인자의 개수, 인자의 리턴 타입 등이 모두 동일하여야 성립
바인딩	정적 바인딩. 컴파일 시에 중복된 메소드 중 호출되는 메소드 결정	동적 바인딩. 실행 시간에 오버라이딩된 메소드 찾아 호출

추상 클래스

- **추상 메소드(abstract method)**
  - absctract로 선언된 메소드, 정의는 없고 원형만 선언

```
abstract public String getName(); // 추상 메소드
abstract public String fail() return "Good Bye"; // 추상 메소드 아님. 컴파일 오류
```

- **추상 클래스(abstract class)**
  - absctract로 선언된 클래스
  - 인스턴스 생성이 불가하며, 상속을 통해 서브 클래스를 구현해야함
  - 생성을 위한 것이 아닌 상속을 위한 슈퍼클래스로 활용하는 용도
    - 다형성 실현

```
// 추상 메소드를 가진 추상 클래스
abstract class Shape {
    public Shape() {
        ...
    }
    public void edit() {
        ...
    }
    abstract public void draw(); // 추상 메소드
}
```

```
// 추상 메소드 없는 추상 클래스
abstract class JComponent {
    String name;
    public void load(String name) {
        this.name = name;
    }
}
```

[연습 4] 추상 클래스 예시

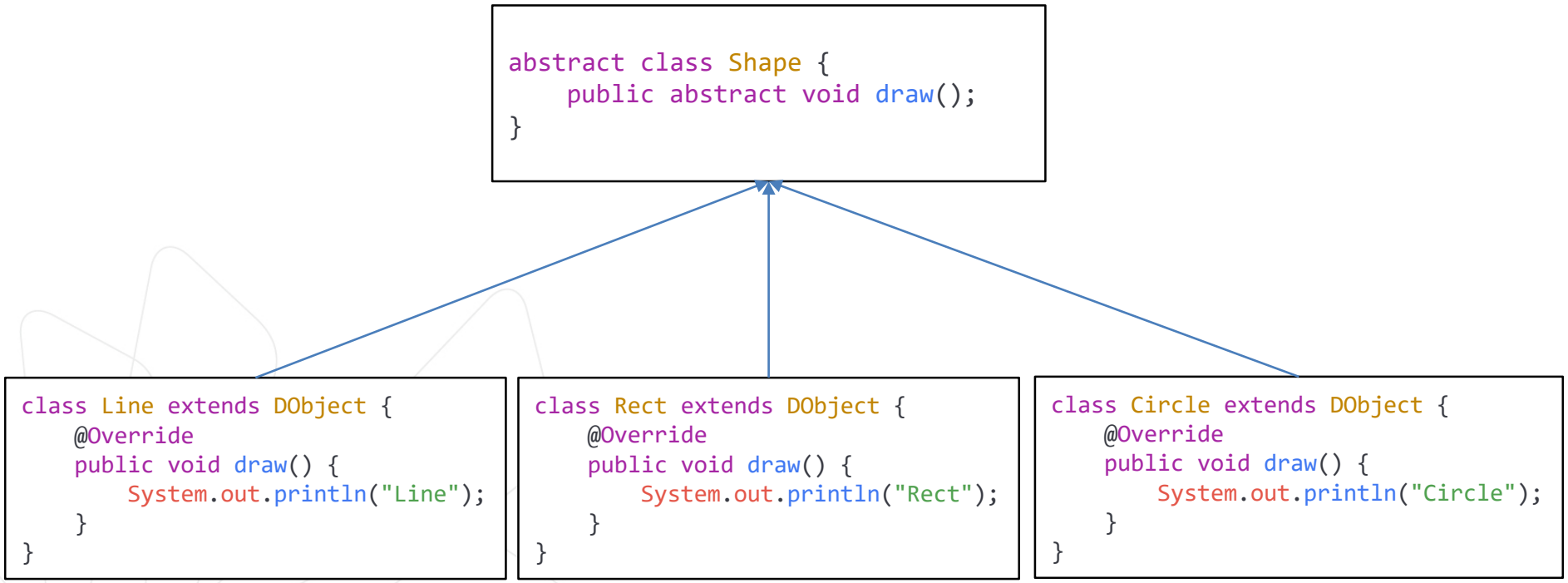
- 추상 클래스 Calcuator를 상속받는 GoodCalc 클래스를 구현하라.

```
abstract class Calculator {
    public abstract int add(int a, int b);
    public abstract int subtract(int a, int b);
    public abstract double average(int[] a);
}
```

```
public class GoodCalc extends Calculator {
    @Override
    public int add(int a, int b) { // 추상 메소드 구현
        return a + b;
    }
    @Override
    public int subtract(int a, int b) { // 추상 메소드 구현
        return a - b;
    }
    @Override
    public double average(int[] a) { // 추상 메소드 구현
        double sum = 0;
        for (int i = 0; i < a.length; i++)
            sum += a[i];
        return sum / a.length;
    }
    public static void main(String[] args) {
        GoodCalc c = new GoodCalc();
        System.out.println(c.add(2, 3));
        System.out.println(c.subtract(2, 3));
        System.out.println(c.average(new int[] {2, 3, 4}));
    }
}
```

추상 클래스의 목적

- **구현이 아닌 상속을 위함**
  - 중복되는 속성과 기능을 추상 클래스로 정의함
  - 서브 클래스에서는 각 클래스에 맞게 구현
    - 다형성 실현



실습과제 안내 [1/2]

- **실습과제 1**
  - 동아리에는 일반 동아리원과 집행부에 속한 학생이 있다. 일반 동아리원을 나타내는 Member 클래스와 집행부에 속한 학생을 의미하는 Crew 클래스를 구현하라. 단, Crew 클래스는 Member 클래스를 상속받는다. Member 클래스의 멤버 필드로는 학번과 이름이 있고, toString() 메소드를 오버라이딩하여 Member 정보를 출력한다. Crew 클래스의 멤버 필드는 담당 분야(ex. president, secretary 등)이 있고 toString() 메소드를 오버라이딩하여 Crew 정보를 Member 정보와 함께 출력한다.
  - Java의 Object 클래스가 가진 메소드 2개 이상을 조사하고 소감에 작성하라.

- **처리조건**
  - extends
  - 접근 지정자
  - 입출력
  - override

- **실행결과**

```
일반 동아리원 수를 입력하시오: 2
집행부 수를 입력하시오: 2

---Input Crew Information---
Student ID: 1011
Name: CREW1
Department: PRESIDENT

Student ID: 1213
Name: CREW2
Department: SECRETARY


---Input Member Information---
Student ID: 1234
Name: ST1

Student ID: 5678
Name: ST2
```

```
---Crew Information---
Student ID: 1011
Name: CREW1
Department: PRESIDENT

Student ID: 1213
Name: CREW2
Department: SECRETARY


---Member Information---
Student ID: 1234
Name: ST1

Student ID: 5678
Name: ST2
```

실습과제 안내 [2/2]

- **실습과제 2**
  - 어느 온라인 쇼핑몰에서 상품마다 고정된 가격을 할인해주는 정책, 상품 가격에 비례해 할인을 해주는 정책이 있다. 이를 구현하기 위해 DiscountPolicy라는 추상 클래스를 만들었다. 이를 상속 받은 FixedDiscountPolicy, PercentDiscountPolicy 두 클래스를 구현해야 한다.
  - discountValue 변수는 할인되는 양을 의미하며 FixedDiscountPolicy에서 getPrice 메소드는 discountValue만큼 할인된 값을 반환한다. PercentDiscountPolicy에서 getPrice 메소드는 discountValue %만큼 할인된 값을 반환한다.

- **처리조건**
  - 멤버 접근 지정자
  - extends
  - 업캐스팅
  - 오버라이딩
  - 입출력

- **실행결과 – 다음장 참고**

```
abstract class DiscountPolicy {
    private int discountValue;
    public DiscountPolicy(int discountValue){
        this.discountValue = discountValue;
    }
    abstract double getPrice(int itemPrice);
}
...
public class FixedDiscountPolicy extends DiscountPolicy {
    ...
}
public class PercentDiscountPolicy extends DiscountPolicy
{
    ...
}
```

실습과제 안내 [2/2]

- 실습과제 2 실행 결과

```
------------------------------------------------

상품의 가격을 입력하세요: 31000
discountValue 설정 값을 입력하세요: 1000
할인 정책을 선택하세요(1. 고정 금액 할인, 2. 퍼센트 할인): 1
할인된 상품 가격: 30000원

------------------------------------------------

상품의 가격을 입력하세요: 30000
discountValue 설정 값을 입력하세요: 10
할인 정책을 선택하세요(1. 고정 금액 할인, 2. 퍼센트 할인): 2
할인된 상품 가격: 27000원

------------------------------------------------
```