

2018-1

객체지향 프로그래밍 과제

Homework1

Composed by:

WISE Research Lab Ajou University



AJOU UNIVERSITY

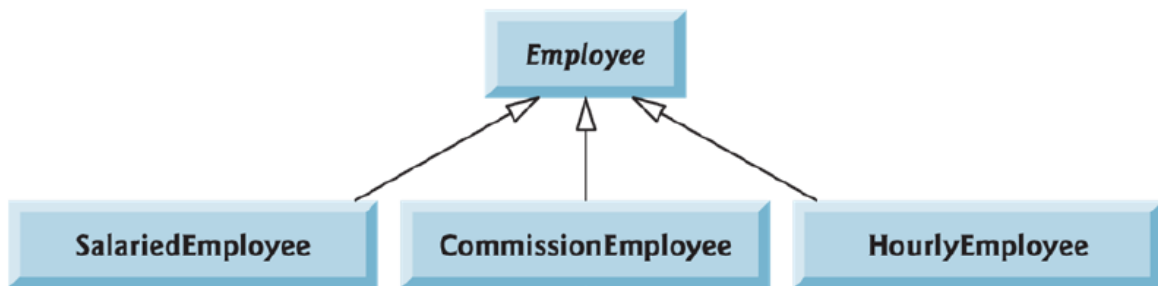
WISE

1 Abstract class

1.1 Abstract class

- 오브젝트의 생성 없이 선언만 필요하다면 abstract class 를 이용할 수 있다.
- Abstract class 의 목적은 적절한 superclass 를 제공하여 이를 상속하는 다른 클래스들이 공통된 디자인을 갖게 하는 것이다.
- **abstract** 키워드를 사용하여 Abstract class 와 method 로 선언할 수 있다.
- Abstract class 는 Inheritance hierarchies 에서 superclass 로만 사용되며 Abstract class 타입의 객체를 생성하여 사용할 수 없다.
- Abstract method 는 선언만이 가능하며, 구현부분을 작성할 수 없다. Abstract method 를 가진 클래스는 반드시 명시적으로 abstract class 로 선언되어야 한다.
- Abstract class 에도 일반적인 Constructor, method 가 있을 수 있으며, 이는 상속되지 않는다.
- Abstract superclass 를 상속받는 subclass 는 abstract method 를 override 해야 한다. (이때 subclass 도 abstract class 인 경우에는 override 를 생략할 수 있다.)

1.2 Abstract class 예제



```
public abstract class Employee {
    private String name;

    public Employee(String name) { super(); this.name = name; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public abstract double earnings();
}
```

```

public class SalariedEmployee extends Employee {
    private double weeklySalary;

    public SalariedEmployee(String name) { super(name); }
    public double getWeeklySalary() { return weeklySalary; }
    public void setWeeklySalary(double weeklySalary) {
        this.weeklySalary = weeklySalary;
    }

    @Override
    public double earnings() {
        return getWeeklySalary();
    }
}

```

```

public class HourlyEmployee extends Employee{
    private double wage;
    private double hours;

    public HourlyEmployee(String name) { super(name); }
    public double getWage() { return wage; }
    public void setWage(double wage) { this.wage = wage; }
    public double getHours() { return hours; }
    public void setHours(double hours) { this.hours = hours; }

    @Override
    public double earnings() {
        if(getHours() <= 40)
            return getWage() * getHours();
        else
            return 40*getWage() + (getHours() - 40)*getWage()*1.5;
    }
}

```

```

public class CommissionEmployee extends Employee{
    private double grossSales;
    private double commissionRate;

    public double getGrossSales() { return grossSales; }
    public void setGrossSales(double grossSales) {
        this.grossSales = grossSales;
    }
    public double getCommissionRate() { return commissionRate; }
    public void setCommissionRate(double commissionRate) {
        this.commissionRate = commissionRate;
    }
    public CommissionEmployee(String name) { super(name); }

    @Override
    public double earnings() {
        return getCommissionRate() * getGrossSales();
    }
}

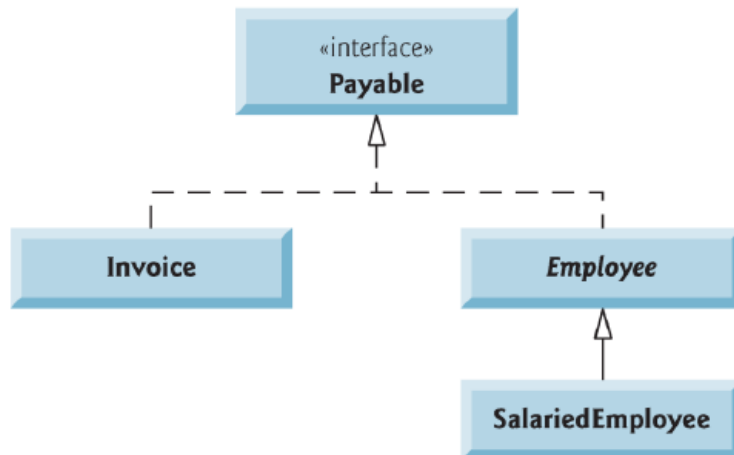
```

2 Interface

2.1 Interface

- Interface란 일반적으로 사람/시스템 간 서로 상호작용하는 방법을 표준화시킨 것을 뜻한다.
 - Interface는 *어떤* operation을 하는지에 대해 명시하고 있지만, *어떻게* 해당 operation을 수행하는지에 대해서는 명시하지 않는다.
 - 자바에서의 Interface는 **interface** 키워드로 선언할 수 있으며, interface에 선언된 method들은 암시적으로 public abstract가, field들은 public, static, final이 된다. (즉, constant와 abstract method만 존재할 수 있다.)
 - 또한, Class와는 다르게 Interface member들은 반드시 public이어야 하며 interface 안에는 어떠한 구현 코드도 포함하지 않는다.
 - Interface를 사용하면 서로 상속관계가 없는 클래스들 사이에서도 공통의 method를 구현할 수 있다.
 - Interface의 method를 구현하고자 하는 클래스들은 **implements** 키워드를 사용하여 interface의 method를 override한다.
- ✓ Abstract class와의 차이점: 상속할 기본 구현이 없을 때 abstract class 대신 interface가 자주 사용된다.

2.2 Interface 예제



```
public interface Payable {
    double getPaymentAmount();
}
```

```
public class Invoice implements Payable{
    private int quantity;
    private double pricePerItem;

    public int getQuantity() { return quantity; }
    public void setQuantity(int quantity) { this.quantity = quantity; }
    public double getPricePerItem() { return pricePerItem; }
    public void setPricePerItem(double pricePerItem) {
        this.pricePerItem = pricePerItem;
    }

    @Override
    public double getPaymentAmount() {
        return getQuantity() * getPricePerItem();
    }
}
```

```
public abstract class Employee implements Payable{
    private String name;

    public Employee(String name) { super(); this.name = name; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public abstract double earnings();
}
```

```

public class SalariedEmployee extends Employee {
    private double weeklySalary;

    public SalariedEmployee(String name) { super(name); }
    public double getWeeklySalary() { return weeklySalary; }
    public void setWeeklySalary(double weeklySalary) {
        this.weeklySalary = weeklySalary;
    }

    @Override
    public double earnings() {
        return getWeeklySalary();
    }
    @Override
    public double getPaymentAmount() {
        return getWeeklySalary();
    }
}

```

<참고예제: instanceof 연산자>

instanceof 연산자는 object 가 특정한 타입인지 비교해준다. 주어진 object 에 대해,

- An instance of a class
- An instance of a subclass
- An instance of a class that implements a particular interface 인지 확인할 수 있다.

```

class InstanceofDemo {
    public static void main(String[] args) {

        Parent obj1 = new Parent();
        Parent obj2 = new Child();

        System.out.println("obj1 instanceof Parent: "
            + (obj1 instanceof Parent));
        System.out.println("obj1 instanceof Child: "
            + (obj1 instanceof Child));
        System.out.println("obj1 instanceof MyInterface: "
            + (obj1 instanceof MyInterface));
        System.out.println("obj2 instanceof Parent: "
            + (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: "
            + (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: "
            + (obj2 instanceof MyInterface));
    }
}

class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}

```

과제 개요

체스¹는 가로와 세로가 각각 8 줄씩 64 칸의 격자로 배열된 체스보드에서 두 명의 플레이어가 피스들을 규칙에 따라 움직여 싸우는 보드 게임이다. 본 과제에서는 각 10 줄씩 100 칸의 격자로 배열된 체스보드에서 재정의된 움직임을 가지는 간단한 체스 프로그램을 구현한다. 프로그램의 기능은 다음과 같다.

1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료

체스 말은 킹(King), 퀸(Queen), 룯(Rook), 나이트(Knight), 비숍(Bishop), 폰(Pawn)으로 구성되며, 각 종류마다 말의 초기 개수, 움직일 수 있는 방향과 범위, 기능들이 모두 다르다. 킹과 퀸은 각 1 기, 룯 2 기, 나이트 2 기, 비숍 2 기, 폰 12 기로 이루어져 있다. 본 과제에서는 표기의 편의성을 위해 킹은 K, 퀸은 Q, 룯은 R, 나이트는 N, 비숍은 B, 폰은 P 로 표시한다. 말의 이동과 체스 판의 표기법은 상세설명을 참고하여 구현한다.

프로그램 작성시 클래스 설계는 자유롭게 하되, 다음에 제시된 조건을 반드시 지켜 작성한다.

- 최소 하나 이상의 **abstract class** 를 만들어 사용한다.
- 반드시 **상속을 활용**한다.
- 모든 클래스의 attribute 는 **private** 또는 **protected** 로 선언하며, getter/setter 를 이용한다.
- 어떠한 경우에도 프로그램이 비정상적으로 종료되지 않도록 **예외 처리**를 한다.

¹ 체스, 위키피디아, <https://ko.wikipedia.org/wiki/%EC%B2%B4%EC%8A%A4>

상세 설명

1. 체스 말 생성

- 설명

10X10 사이즈의 체스 판을 생성한다. 체스 말들의 위치를 체스 규칙에 맞춰 초기화 한다. 생성시 말들의 위치는 다음 그림과 같이 흰색 말은 1, 2 열에 검은색 말은 9,10 열에 두는 것으로 한다.



P	R	N	B	K	Q	B	N	R	P
P	P	P	P	P	P	P	P	P	P
P	P	P	P	P	P	P	P	P	P
P	R	N	B	K	Q	B	N	R	P

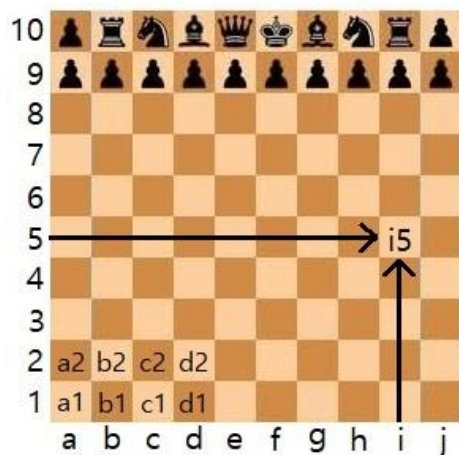
2. 체스 말 랜덤 생성

- 설명

10X10 사이즈의 체스 판을 생성한다. 체스 말들의 개수는 게임의 규칙에 맞춰 생성하되, 말의 위치를 랜덤하게 생성한다. 단, 말끼리 위치가 겹치지 않도록 한다.

3. 체스 말 삭제

- 설명



사용자로부터 위치를 입력 받아 해당 위치에 있는 체스 말을 삭제한다. 위치의 입력은 그림과 같이 A1 부터 J10 까지 그림과 같은 형태로 받는다. 대문자와 소문자는 같은 문자로 인식되어야 하며, 범위가 벗어났을 경우와 입력된 위치에 말이 없을 경우에 대한 예외처리가 있어야 한다.

4. 체스판 보기

- 설명

체스판의 현황을 다음 예시와 같이 출력한다.

P1	R1	N1	B1	K1	Q1	B1	N1	R1	P1
P1	P1	P1	P1	P1	P1	P1	P1	P1	P1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
P0	P0	P0	P0	P0	P0	P0	P0	P0	P0
P0	R0	N0	B0	K0	Q0	B0	N0	R0	P0

0 : 밝은 칸, 1 : 어두운 칸
R : 룯(Rook) + 0(흰 말), 1(검은 말)
N : 나이트(Knight)
B : 비숍(Bishop)
K : 킹(King)
Q : 퀸(Queen)
P : 폰(Pawn)

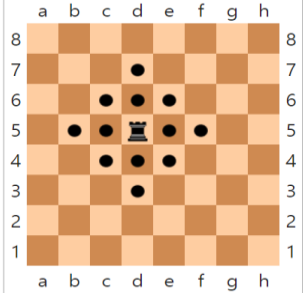
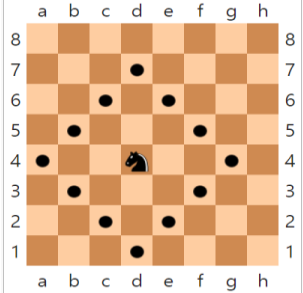
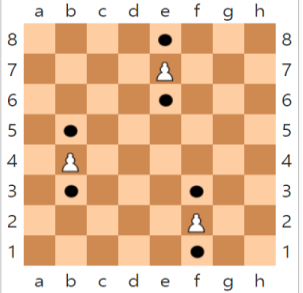
5. 체스 말 이동

- 설명

사용자로부터 움직일 말의 위치와 움직이고 싶은 위치를 입력 받는다. 해당 말이 다른 색의 말의 위치로 이동할 경우, 상대 체스 말을 제거하고 해당 말을 그 위치로 이동시킨다. 말이 이동할 수 없는 위치인 경우 에러 메시지를 출력한다. 다음은 각 유닛 별 행마법에 대한 설명으로, 한 차례를 기준으로 한다. 기존의 체스의 규칙과는 다르게 **본 과제에서 임의로 지정한 움직임을 구현**한다.

킹은 상대 진영을 향한 3 방향(직선, 좌우 대각선)으로는 여러 칸을 움직일 수 있으며, 나머지 방향으로 한 칸씩만 이동이 가능하다. **퀸**은 기존 체스의 룯과 같다. **비숍**은 대각선으로 (사용자의 선택에 따라) 한 칸 혹은 두 칸씩 이동이 가능하다, 상하좌우로는 한 칸씩 이동이 가능하며, **룩**은 대각선으로는 한 칸, 상하좌우로는 (사용자의 선택에 따라) 한 칸 혹은 두 칸씩 이동이 가능하다. **나이트**는 기존의 룰에 상하좌우 두 칸 이동이 추가되었다. **폰**은 상하 방향으로 한 칸 이동이 가능하다. 마지막으로 나이트를 제외한 나머지 말은 진행방향에 다른 말이 있을 경우 건너뛰어 이동할 수 없다.

<p>검은 말 진영</p> <p>흰색 말의 공격 진행 방향</p> <p>흰색 말 진영</p>		
킹의 이동	퀸의 이동	비숍의 이동

		
룩의 이동	나이트의 이동	폰의 이동

6. 종료

- 설명
프로그램을 종료한다.

출력 예시

1. 프로그램 시작화면

```
<Ajou Chess Board>

원하는 메뉴를 선택하세요.
1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료
>>
```

2. 체스 말 (랜덤) 생성

```
<Ajou Chess Board>

원하는 메뉴를 선택하세요.
1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료
>> 1
체스판 생성이 완료되었습니다.

=====
```

3. 체스 말 삭제

- 해당 위치에 말이 있을 경우

원하는 메뉴를 선택하세요.

1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료

>> 3

삭제를 원하는 체스 말의 위치를 입력하세요. A1

체스 말이 삭제되었습니다.

=====

- 해당 위치에 말이 없을 경우

원하는 메뉴를 선택하세요.

1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료

>> 3

삭제를 원하는 체스 말의 위치를 입력하세요. A1

해당 위치에 체스 말이 없습니다.

=====

4. 체스판 보기

원하는 메뉴를 선택하세요.

1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료

>> 4

```
P1 R1 N1 B1 K1 Q1 B1 N1 R1 P1
P1 P1 P1 P1 P1 P1 P1 P1 P1 P1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
P0 P0 P0 P0 P0 P0 P0 P0 P0 P0
P0 R0 N0 B0 K0 Q0 B0 N0 R0 P0
```

=====

5. 체스 말 이동

- 이동이 가능한 경우

```
원하는 메뉴를 선택하세요.
1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료
>> 5
움직일 말의 위치를 입력하세요. A2
말이 이동할 위치를 입력하세요. A3
말을 A3(으)로 이동 시켰습니다.

=====
```

- 이동이 불가능한 경우

```
원하는 메뉴를 선택하세요.
1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료
>> 5
움직일 말의 위치를 입력하세요. A1
말이 이동할 위치를 입력하세요. A2
해당 위치로 이동할 수 없습니다.

=====
```

6. 종료

```
원하는 메뉴를 선택하세요.
1. 체스 말 생성
2. 체스 말 랜덤 생성
3. 체스 말 삭제
4. 체스판 보기
5. 체스 말 이동
6. 종료
>> 6
프로그램을 종료합니다.
```

✓ **실습과제 제출기한**

- 1) 4 월 22 일 일요일 자정까지 - 100%
- 2) 그 이후 제출 - 0%

✓ **QA**

- 1) QA 시간: 4 월 18 일 15~18 시, 19 시~21 시, 팔달관 913-2
(방문 전 미리 메일주시기 바랍니다.)
- 2) 이메일 문의: ch0104@ajou.ac.kr (조현 조교)