컴퓨터 프로그래밍

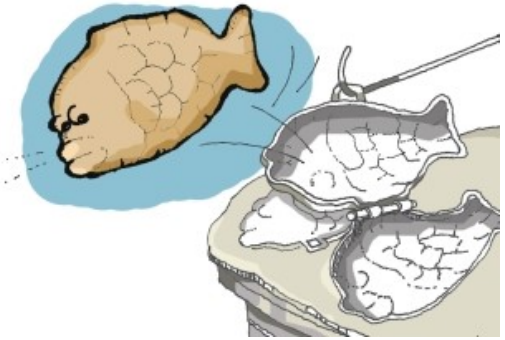
class

객체 지향 프로그래밍

- Object-Oriented Programming
  - 실제 세계가 객체(object)들로 구성되어 있다는 것에 착안하여 소프트웨어를 객체 단위로 작성하는 방법
  - 객체는 '데이터'와 '기능'으로 정의되어 상호작용
  - Java에서 객체는 class로 정의

class

- Java에서 객체를 만들기 위한 틀
- class는 멤버 변수와 멤버 메소드로 구성
  - 멤버 변수는 객체의 특성을 표현
  - 멤버 메소드는 객체의 행위를 표현



- class 디자인 시 생각해볼 점들
  - 내가 표현하고자 하는 객체의 특성은?
  - 내가 다루고자 하는 data의 공통 속성은?
  - 만들어진 객체가 할 일은?

```
class Rectangle {
    int x1;
    int y1;
    int x2;
    int y2;

    int getHeight() {
        return Math.abs(y1-y2);
    }
}
```

객체 생성

- class로부터 실제 객체(instance) 만들기
  - Instantiation (인스턴스화)
  - Rectangle rect = new Rectangle();

- new
  - 객체 생성을 위한 키워드
  - 메모리(heap)에 객체의 저장공간을 생성하고 그 주소를 반환
  - Heap: 자유저장공간

- 참조변수 (reference variable)
  - 객체 저장공간의 주소를 저장하는 변수
    - e.g. Rectangle rect
  - 객체의 데이터가 아니라 객체가 저장된 위치의 주소를 저장

객체 생성

```
class Rectangle {
    int x1, y1, x2, y2;
    int getHeight() {
        return Math.abs(y1-y2);
    }

    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();
    }
}
```

객체 내부 접근

- 객체 내에 존재하는 변수 혹은 메소드는 점 연산자(.)로 접근

- 객체 내에 존재하는 변수로의 접근
  - Rectangle rect = new Rectangle();
    rect.x1 = 10;

- 객체 내에 존재하는 메소드의 호출
  - Rectangle rect = new Rectangle();
    rect.getHeight();

객체 내부 접근

멤버 변수에 접근 →

멤버 메소드 호출 →

```
class Rectangle {
    int x1, y1, x2, y2;
    int getHeight() {
        return Math.abs(y1-y2);
    }

    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        r1.x1 = 3;    r1.y1 = 5;
        r1.x2 = 10;   r1.y2 = 10;
        Rectangle r2 = new Rectangle();
        r2.x1 = 7;    r2.y1 = 7;
        r2.x2 = 14;   r2.y2 = 12;
        System.out.println(r1.getHeight());
    }
}
```

Memory Model

- Java Virtual Machine(JVM)은 운영체제 입장에서 하나의 프로그램
  - JVM에게도 다른 프로그램과 마찬가지 크기의 메모리 공간 할당
  - JVM은 OS로부터 할당 받은 메모리로 자기 자신과 Java 프로그램 실행

- JVM의 메모리 구분 및 관리

메소드 영역 (method area)	바이트 코드, static 변수
스택 영역 (stack area)	지역변수, 매개변수
힙 영역 (heap area)	인스턴스

Method Area

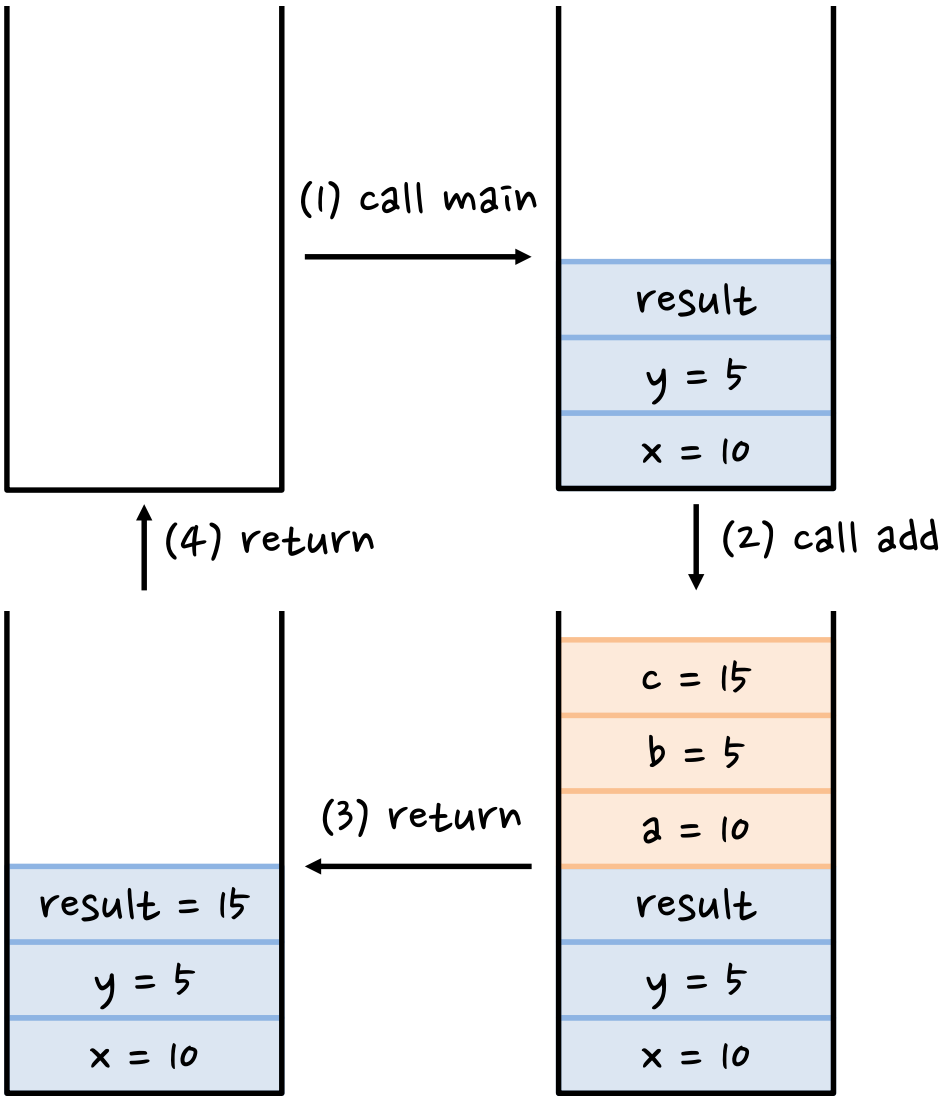
- 메소드 영역에 저장되는 내용
  - 실행에 필요한 바이트 코드
  - static 변수 및 메소드
    - 인스턴스 생성 없이 static 변수 및 메소드에 접근 가능한 이유
    - e.g. Math.PI, Math.pow()

Stack Area

- 스택 영역에 저장되는 내용
  - 지역 변수, 매개 변수
    - 프로그램이 실행되는 도중 할당
    - 불필요시 소멸

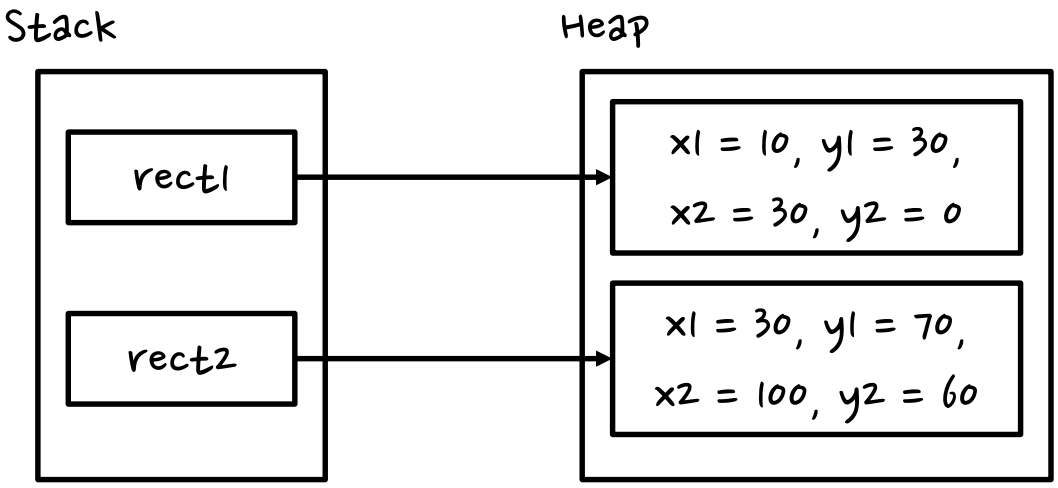
```
public static void main(String[] args) {
    int x = 10, y = 5, result;
    result = add(x, y);
    System.out.println(result);
}

public static int add(int a, int b) {
    int c = a + b;
    return c;
}
```



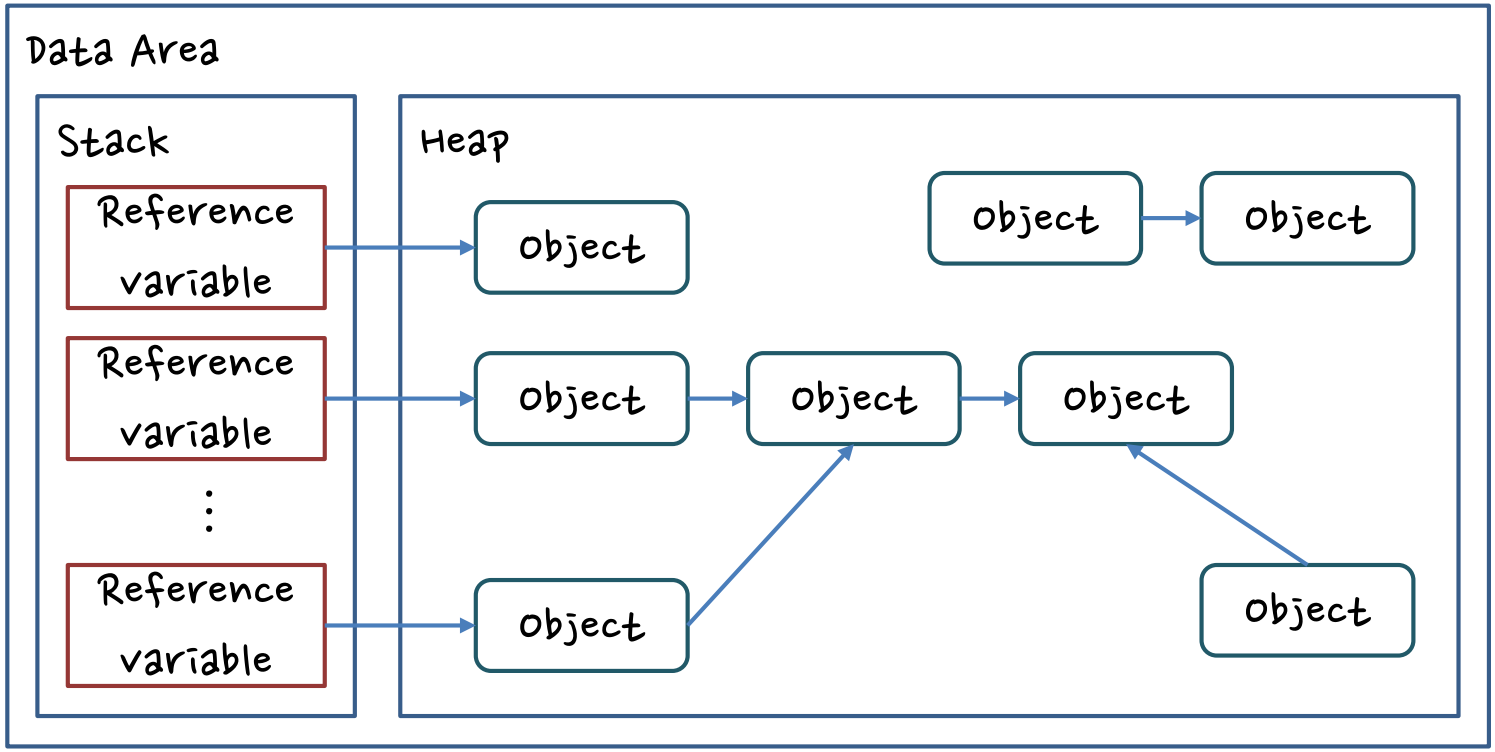
Heap Area

- 힙 영역에 저장되는 내용
  - 생성된 인스턴스
- JVM에 의한 메모리 정리(garbage collection)가 발생하는 공간
  - 인스턴스 생성은 프로그래머가, 소멸은 JVM이 담당
  - 참조변수에 의한 참조를 더 이상 가지지 않는 인스턴스는 소멸 대상



Garbage collection

- Garbage
  - 더 이상 참조되지 않는 인스턴스
- Reachability
  - 어떤 인스턴스에 유효한 참조가 존재하는가?
  - unreachable 인스턴스는 garbage로 간주하여 삭제



Garbage collection

- Garbage collection 시점
  - 소멸 대상이 되었다고 바로 소멸되지는 않는다
  - Garbage collection도 시스템 자원을 이용하므로 빈번한 수행은 부담
  - 성능에 영향을 미치지 않도록 별도 알고리즘으로 수행

Object as Parameter

- 객체를 method의 parameter로 전달

```
class Rectangle {
    int x1, y1, x2, y2;
    int getHeight {
        return Math.abs(y1-y2);
    }
    boolean intersects(Rectangle rect) {
        // x1, y1 ??
        // rect.x1, rect.y1 ??
    }
}
```

```
public static void main(String[] args) {
    Rectangle r1 = new Rectangle();
    r1.x1 = 3;    r1.y1 = 5;
    r1.x2 = 10;   r1.y2 = 10;
    Rectangle r2 = new Rectangle();
    r2.x1 = 7;    r2.y1 = 7;
    r2.x2 = 14;   r2.y2 = 12;
    if (r1.intersects(r2)) {
        // ...
    }
}
}
```

실습

- 사각형을 표현하기 위한 Rectangle class 구현

- 두 사각형이 교차하는지 판단하는 intersects method 구현
  - e.g.
    - 사각형 r1과 사각형 r2가 교차하는가?
    - boolean intersect = r1.intersects(r2);

- 두 사각형이 포함 관계인지 판단하는 contains method 구현
  - e.g.
    - 사각형 r1이 사각형 r2를 둘러싸고 있는가?
    - boolean contain = r1.contains(r2);

실습

- 복소수를 표현하기 위한 complex class 구현
- 복소수 간의 덧셈, 뺄셈, 곱셈을 할 수 있는 method 구현

참조 (Reference)

- 참조 변수에는 주소가 저장된다는데, 그 의미는?

```
complex c1 = new complex();
c1.real = 5;
c1.imaginary = 3;
complex c2 = c1;
c2.real = 1;
c2.imaginary = 10;
```

vs.

```
complex c1 = new complex();
c1.real = 5;
c1.imaginary = 3;
complex c2 = new complex();
c2.real = 1;
c2.imaginary = 10;
```

참조 (Reference)

- 참조 변수를 parameter로 전달하는 경우

```
complex c1 = new complex();
c1.real = 5;
c1.imaginary = 3;
addOne(c1);
-------------------------
void addOne(complex c) {
    c.real++;
}
```

vs.

```
int a = 5;
addOne(a);
-------------------
void addOne(int i) {
    i++;
}
```

생성자 (constructor)

- 객체를 생성하면서 멤버 변수를 원하는 값으로 초기화 하고 싶은 경우

```
complex c1 = new complex();
c1.real = 5;
c1.imaginary = 3;
```



```
complex c1 = new complex(5, 3);
```

- 객체 생성시 딱 한 번 호출되는 '생성자' method를 만들어라

- 생성자 특징
  - class 이름과 동일한 이름의 method
  - Return type 없음, 따라서 반환도 없음

생성자 (constructor)

- Example

```
complex(int r, int i) {
    real = r;
    imaginary = i;
}
```

1+2i 복소수 객체를 생성하려면

```
complex c = new complex(1, 2);
```

— 생성자를 사용한 객체 생성 문장은 객체를 생성함과 동시에 생성자 method를 실행한다는 의미로 이해

기본 생성자 (Default constructor)

- 생성자를 정의하지 않으면 Java compiler는 기본 생성자를 정의
- 기본 생성자는 아무런 parameter를 받지 않으며 하는 일도 없다
- 사용자 정의 생성자를 만들면 기본 생성자는 만들어지지 않는다

- complex c = new complex(); 가 error 없이 실행되는 것도 기본 생성자 덕분

실습

- 가로, 세로, 높이 값을 가지는 Box class 구현
- Box class의 생성자 구현
  - 가로, 세로, 높이를 모두 매개변수로 받거나,
  - 아무 것도 받지 않을 수 있다
    - 아무 것도 받지 않을 경우 가로, 세로, 높이는 모두 1로 한다
- boolean canPack(Box b) method 구현
  - Box b를 canPack method를 호출한 box 객체 안에 넣을 수 있는지 확인하는 method
    - 만약 b1.canPack(b2);를 실행하여 true가 return되면 b1안에 b2를 넣을 수 있음

Object as class Member variable

- Rectangle class 개선
- 멤버 변수 x1, y1, x2, y2를 숫자 쌍으로 표현한다면?

```
class Point {
    int x, y;
    Point(int x_, int y_) {
        x = x_;
        y = y_;
    }
}
```

```
class Rectangle {
    Point p1, p2;
    int getHeight {
        return Math.abs(p1.y-p2.y);
    }

    public static void main(String[] args) {
        Rectangle rect = new Rectangle();
        rect.p1 = new Point(2, 3);
        int p1x = rect.p1.x;
    }
}
```

Access control

- 내가 만든 class를 사용하는 사람이 멤버 변수에 접근할 필요가 있는가?
  - Rectangle class가 오른쪽과 같을 때 멤버 변수 y1 or y2를 마음대로 수정할 수 있다면 height 정보는 믿을 수 있는 것인가?

- 정보 은닉 (data hiding)
  - 멤버 변수는 가능한 숨기고, 멤버 메소드를 통해 멤버 변수를 제어
  - 멤버 메소드는 멤버 변수의 정합성(consistency)을 유지하도록 설계

```
class Rectangle {
    int x1;
    int y1;
    int x2;
    int y2;
    int height;

    int getHeight() {
        return height;
    }
    Rectangle(int x_1, int y_1, ···) {
        x1 = x_1;
        ···
        height = Math.abs(y1-y2);
    }
}
```

접근 제어 지시자 (Access control Indicators)

- private
  - class 내부에서만 접근 가능
- public
  - 누구나 접근 가능

```
class Rectangle {
    private int x1;
    private int y1;
    private int x2;
    private int y2;
    private int height;

    public int getHeight() {
        return height;
    }
    public Rectangle(int x_1, int y_1, ···) {
        x1 = x_1;
        ···
        height = Math.abs(y1-y2);
    }
}
```

접근 제어 지시자 (Access control Indicators)

- default
  - 접근 제어 지시자를 사용하지 않은 경우
  - 동일 package 내에서 접근 허용
- protected
  - 상속 관계인 경우 접근 가능
  - default의 접근 범위 + 상속 관계 접근

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	o	x	x	x
default	o	o	x	x
protected	o	o	o	x
public	o	o	o	o

클래스 접근 제어 지시자

- public class
  - 어디서나 객체 생성 가능
  - 하나의 소스 파일(.java)에 하나의 클래스만 public으로 선언 가능
  - public 클래스의 이름과 소스 파일의 이름이 동일할 것

- default class
  - 동일한 패키지 내에 정의된 클래스에 의해서만 객체 생성 가능

- private class
  - 멤버 클래스
  - 클래스 내부에 정의되며 해당 클래스에서만 객체 생성 가능

실습

- 다음과 같은 Rectangle class를 만들자
- 새로운 RectangleMain class를 만들어 접근 제어 지시자를 바꿔가며 멤버 변수 혹은 멤버 메소드에 접근할 수 있는지 알아보자
- 만들어진 사각형 객체의 좌표를 바꾸려면 어떻게 해야할까?

```
class Rectangle {
    private int x1;
    private int y1;
    private int x2;
    private int y2;
    private int height;

    public int getHeight() {
        return height;
    }
    public Rectangle(int x_1, int y_1, ···) {
        x1 = x_1;
        ···
        height = Math.abs(y1-y2);
    }
}
```

```
class RectangleMain {
    public static void main(String[] args) {
        Rectangle rect
            = new Rectangle(1, 2, 3, 4);
        // rect.x1?
        // rect.getHeight()?
    }
}
```

static

- 멤버 변수나 멤버 메소드를 static으로 선언하는 경우
  - 한 class의 모든 객체가 공유
  - 객체를 생성할 필요 없이 초기화 및 접근 가능

```
public final class java.lang.Math {
  public static final double E = 2.718281828459045;

  public static final double PI = 3.141592653589793;

  public static double pow(double a, double b)
                                :
}
```

```
Math.PI
Math.pow(2, 2)
```

System.out.println

- System
  - java.lang package 내 class 이름
  - import java.lang.*; 는 자동 삽입
- out
  - System class 내 static 변수
    - System class의 객체를 생성하지 않고 사용 가능하므로
  - 참조 변수
    - println이라는 method를 가지므로

```
public class java.lang.System {
  public static final PrintStream out;
                    :
}
```

public static void main

- main method는 아무 class에 속해도 무관
  — 어느 .class 파일을 실행시키는가만 달라짐
- main method는 객체 생성 없이 호출되어야 하므로 static
- 외부(JVM)에서 호출되므로 public

Package & Import (optional)

- 서로 다른 일을 하지만 이름이 같은 class가 존재한다면?

```
class circle {
    double rad;
    public circle(double r) {
        rad = r;
    }
    public double getPerimeter() {
        return (rad*2)*Math.PI;
    }
}
```

```
class circle {
    double rad;
    public circle(double r) {
        rad = r;
    }
    public double getArea() {
        return (rad*rad)*Math.PI;
    }
}
```

```
circle c1 = new circle(1,5);
```

?

Package & Import (optional)

- class를 서로 다른 package에 저장

```
package area;
class circle {
    double rad;
    final double PI;
    public circle(double r) {
        rad = r;
        PI = 3.14;
    }
    public double getArea() {
        return (rad*rad)*PI;
    }
}
```

```
area.circle c1 = new area.circle(1,5);
```

Package & Import (optional)

- 난 area에 있는 circle만 쓸건데 매번 area.circle이라고 쓰기 귀찮다
- import area.circle;
- 이후 circle이라고 쓰면 area.circle로 인식

Summary

- class
  - Java에서 객체를 만들기 위한 틀
  - 멤버 변수와 멤버 메소드로 구성
- Instantiation
  - 클래스로부터 실제 객체를 생성 (new 키워드)
  - 생성된 객체는 heap 영역에 저장되며 참조변수에 주소를 저장하여 사용
- Garbage collection
  - Java가 스스로 heap 영역의 데이터를 소멸시키는 절차
  - 더 이상 참조되지 않는 데이터는 자동 소멸
- constructor
  - 객체를 생성할 때 초기화를 위해 수행되는 method

Summary

- Access control
  - OOP의 data hiding
  - Data consistency 유지
  - 공개(public)된 메소드를 통해 숨겨진(private) 데이터 제어
  - 멤버에 대한 접근 제어
    - private, public, protected, default
  - 클래스 자체에 대한 접근 제어
    - private, public, default

실습

- 다음의 기능을 하는 Employee class를 구현
  - 직원은 이름과 월급을 private 변수로 가진다
  - 특정 직원의 월급을 조절할 수 있다

  - 채용된 직원의 수를 알 수 있다
  - 월급이 인상된 직원의 수를 알 수 있다
  - 월급이 삭감된 직원의 수를 알 수 있다

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Employee e1 = new Employee("박근혜", 10000);
    Employee e2 = new Employee("문재인", 10000);
    Employee e3 = new Employee("안철수", 10000);

    e1.changeSalary(5000);
    e2.changeSalary(15000);

    System.out.println("현재 직원 수: "+Employee.getCount());
    System.out.println("월급이 인상된 직원 수: "+Employee.salaryRaised);
    System.out.println("월급이 삭감된 직원 수: "+Employee.salaryReduced);
}
```

```
현재 직원 수: 3
월급이 인상된 직원 수: 1
월급이 삭감된 직원 수: 1
```

과제

- 은행 계좌(Account) 클래스와 예금주(Holder) 클래스를 만들어 입출금 및 잔액 조회가 가능한 프로그램을 작성
  - 초기 설정
    - 예금주는 일정 금액을 소유
    - 은행 계좌는 현재 예금액과 계좌 비밀번호로 구성
  - 동작
    - 예금주는 소유액을 은행계좌에 입금 가능
      - 소유액이 입금액 이상인 경우 입금 불가
    - 예금주는 예금액을 은행계좌로부터 출금 가능
      - 출금액이 예금액 이상인 경우 출금 불가
    - 예금주가 은행 계좌에서 출금하는 경우 출금액과 비밀번호가 필요
      - 비밀번호가 틀릴 경우 출금 불가
    - 예금주가 은행 계좌를 조회하는 경우 비밀번호가 필요
      - 비밀번호가 틀릴 경우 조회 불가

실행 예

```
public class BankMain {

    public static void main(String[] args) {
        Account account = new Account(10000, 1234);
        Holder holder = new Holder(10000);

        holder.withdraw(account, 12000, 1234);
        holder.deposit(account, 15000);
        holder.deposit(account, 5000);
        holder.withdraw(account, 12000, 1235);
        holder.withdraw(account, 12000, 1234);

        System.out.println(holder.getBalance(account, 2234));
        System.out.println(holder.getBalance(account, 1234));
    }

}
```

```
잔액이 부족합니다.

소유액이 부족합니다.
입금되었습니다.
잔액은 15000원 입니다.

비밀번호가 잘못되었습니다.

출금되었습니다.
잔액은 3000원 입니다.

비밀번호가 잘못되었습니다.

잔액은 3000원 입니다.
```