**Report02**



|  |  |
| --- | --- |
| **과 목 :** | **소프트웨어프로젝트** |
| **제출 일자 :** | **2019. 4. 15** |
| **담당 교수 :** | **이남규** |
| **학 과 :** | **사회복지학부** |
| **학 번 :** | **20170299** |
| **이 름 :** | **김유빈** |

**<1. 사각형을 나타내는 Rectangle 클래스를 작성하라.>**

포인트) **생성자**

먼저 클래스 생성

**class** Rectangle{

**double** width, height;

String color;

1. 디폴트 값의 사각형을 위한 무인자 생성자

-> 즉, 매개변수 없는 매개변수 초기화 목적의 생성자

Rectangle(){

width = 1; //디폴트 값 = 1

height = 1; // ""

color = "white"; //디폴트 값 = "white"

}

2. 주어진 가로와 세로의 사각형을 위한 생성자

-> 즉, 매개 변수 가로, 세로가 들어오는 생성자

Rectangle(**double** width, **double** height){ //매개변수 width, height

**this**(); -> 필드에 디폴트 값 삽입을 위해 또다른 생성자 호출

**this**.width = width; -> 매개변수 이름 == 필드 이름 일 때 this 사용

**this**.height = height;

}

3. 주어진 하나의 길이(가로와 세로로 사용)의 사각형을 위한 생성자

-> 2번과 비슷하다, 단 매개변수가 하나이다(2번과는 다른 생성자)

Rectangle(**double** line){

**this**();

width = line; -> 매개변수랑 이름이 겹치지 않으므로 this 사용x

height = line;

}

생성자에 의해 width와 height가 초기화 된 객체 생성 완료

그 다음은 함수 호출로 면적(getArea() 함수)과 둘레(getPerimeter() 함수) 구하기

**public** **double** getArea() { //직사각형의 면적 계산

**return** (width \* height);

}

**public** **double** getPerimeter() { //직사각형의 둘레의 길이 계산

**return** 2 \* (width + height);

}

**<객체 배열>**

Rectangle [] r = **new** Rectangle[3];

|  |
| --- |
| r[0] |
| r[1] |
| r[2] |

r

객체 배열에 대한 레퍼런스 변수 r 선언

레퍼런스의 배열 생성

r[0] = **new** Rectangle();

r[1] = **new** Rectangle(width,height);

r[2] = **new** Rectangle(width);

width = 1

height = 1

|  |
| --- |
| r[0] |
| r[1] |
| r[2] |

r

width = 입력한 가로

height = 입력한 세로

width = height = 입력한 길이

각 레퍼런스 배열에 대한 객체 생성

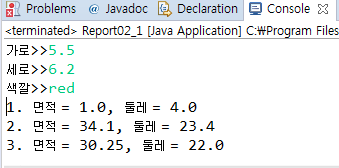
<출력>

for문으로 r[i]로 접근하기

**for**(**int** i=0;i<**r.length**;i++) -> 배열의 길이 = 3만큼 반복

System.out.println((i+1)+".면적 = " + r[i].getArea() + ", 둘레 = " + r[i].getPerimeter());

<실행 결과>



**<2. 더하기(+), 빼기(-), 곱하기(\*) 나누기(/)를 수행하는 각 클래스 Add, Sub, Mul, Div>**

1. default 패키지에 하나 파일에는 Calc, 다른 하나의 파일에는 Add, Sub, Mul, Div 클래스를 포함하도록 작성하라.

아래의 main함수의 틀은 똑같다

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

**int** num1 = s.nextInt(); //피연산자1(정수형)

String cal = s.next(); //연산자

**int** num2 = s.nextInt(); //피연산자2(정수형)

**if**(cal.equals("+")) { //연산자가 덧셈일 때

Add temp = **new** Add(); -> Add클래스가 디폴트이므로 같은 패키지 내에서면 접근 가능 //객체 생성

temp.setValue(num1, num2);

System.***out***.println(temp.calculate()); //덧셈 수행

}

Add클래스를 똑 같은 패키지안에 생성하여

**class** OtherCalc{ ->Add, Sub, Mul, Div 클래스에서 공통된 메소드와 필드를 포함하는 부모 클래스

**int** a, b;

**void** setValue(**int** a, **int** b) {

**this**.a = a;

**this**.b = b;

}

}

**class** Add **extends** OtherCalc{ ->OtherCalc클래스의 자식 클래스 Add

**int** calculate() { //덧셈 수행

**return** a + b;

}

}

1. pkg1 패키지에 Calc 클래스, pkg2 패키지에 Add, Sub, Mul, Div 클래스를 가지도록 작성하라

**public** **class** Calc 를 pkg1에 생성 (틀은 1번과 같다)

단, **import** pkg2.Add;

**import** pkg2.Sub;

**import** pkg2.Mul;

**import** pkg2.Div; -> 외부 패키지에 대한 접근

**public** **class** Add, Sub, Div, Mul 은 pkg2에 따로 생성

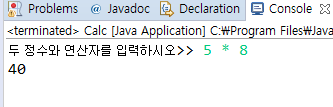
다른 패키지 접근이 되어야 하므로 public으로 접근 생성

각 클래스 내에 **public** **void** setValue(**int** a, **int** b)

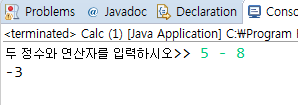
**public** **int** calculate()도 다른 패키지(pkg1) 접근이 되어야 하므로 public으로 접근 생성

<실행결과>

1.



2.



**<3.다음은 키와 값을 하나의 아이템으로 저장하고 검색 수정이 가능한 추상 클래스>**

1. String get(String key) : key 값으로 'value'를 검색

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | KEY |  |  |  |  |  |  |  |

keyArray

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

valueArray

String result = **null**; //결과 변수 초기화

**for**(keyArray배열의 길이만큼 반복)

**if**(keyArray[i].equals(key)) { //해당 key를 찾으면

result = valueArray [i]; //해당 VALUE-> 결과변수

**break**;

}

**return** result;//같은 번지의 value 반환

2. **void** put(String key, String value) : key와 value를 쌍으로 '저장'

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | KEY |  |  |  |  |  |  |  |

keyArray

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | VALUE |  |  |  |  |  |  |  |

valueArray

**for**(keyArray배열의 길이만큼 반복) {

**if**(keyArray[i] == key) { //만약 같은 키가 이미 있다면

valueArray [i] = value; //새로운 값 대입

**break**;

}

**else** **if**(keyArray[i] == **null**) {

keyArray[i] = key; //해당 키가 새로운 키라면 비어 있는 배열에 삽입

valueArray [i] = value; //해당 값도 삽입

**break**;

}

}

3. String delete(String key) : key 값을 가진 아이템(value와 함께)을 '삭제'. 삭제된 value 값 리턴

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | KEY |  |  |  |  |  |  |  |

keyArray

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | VALUE |  |  |  |  |  |  |  |

valueArray

String result = **null**;

**for**(**int** i=0;i < keyArray.length;i++)

**if**(keyArray[i].equals(key)) { //해당 키 값과 같다면

valueArray[i] = **null**; //해당 키의 값을 삭제하고

result = valueArray[i]; //그 값을 결과에 집어넣기

**break**;

}

**return** result;

4. **int** length() : 현재 저장된 아이템의 개수 리턴

**int** count = 0; -> 아이템 저장 개수

**for**(**int** i=0;i < keyArray.length;i++)

**if**(keyArray[i] != **null**) //널문자가 아닐때 마다

count++; //갯수 더하기

**return** count;

마지막으로

**class** Dictionary **extends** PairMap 는 상속받도록 하여 자식 클래스

Dictionary(**int** num)라는 생성자 함수{//매개 변수 == 배열 크기

keyArray = **new** String[num];

valueArray = **new** String[num];

}

<실행결과>

