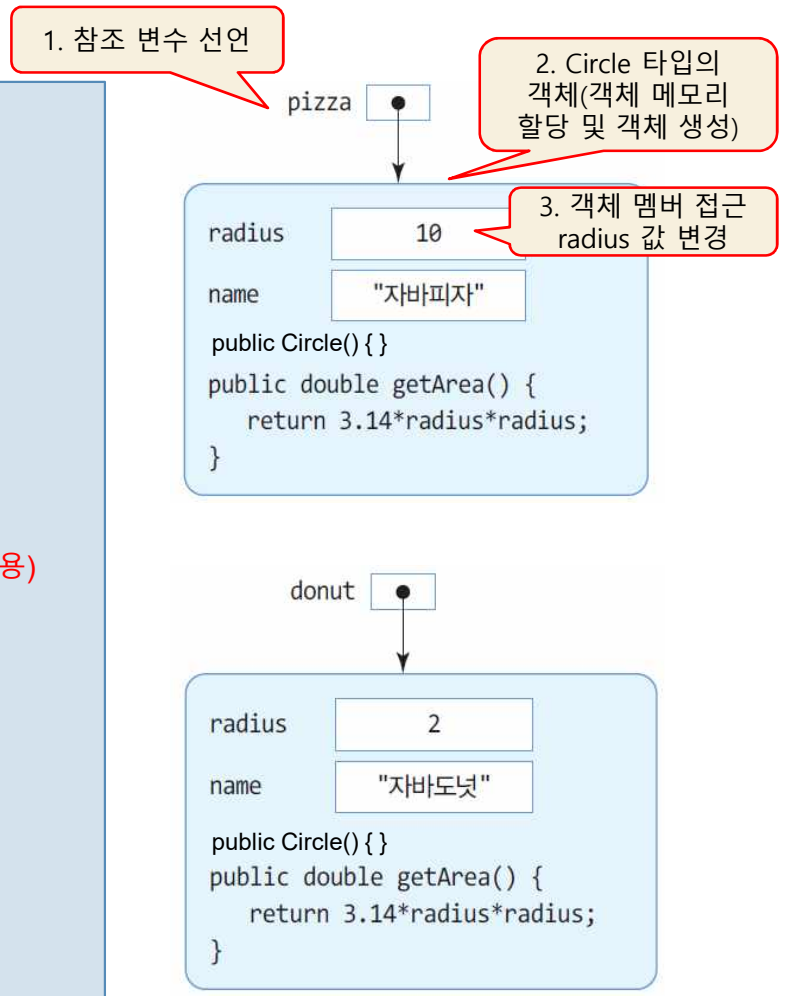
4장 객체 지향

예제4-1 : Circle 클래스의 객체 생성 및 활용

반지름과 이름을 가진 Circle 클래스를 작성하고, Circle 클래스의 객체를 생성하고 객체가 생성된 모습을 표현하시오.

```
public class Circle {
    int radius;              // 원의 반지름 필드
    String name;             // 원의 이름 필드

    public Circle() { }          // 원의 생성자

    public double getArea() {    // 원의 면적 계산 메소드
        return 3.14*radius*radius;
    }

    public static void main(String[] args) {
        Circle pizza;                        //1. 참조변수 선언
        pizza = new Circle();                // 2. 객체 생성(new 연산자 이용)
        pizza.radius = 10;                   // 3. 객체 멤버 접근(점(.)연산자이용)
        pizza.name = "자바피자";             // 피자의 이름 설정
        double area = pizza.getArea();       // 피자의 면적 알아내기
        System.out.println(pizza.name + "의 면적은 " + area);

        Circle donut = new Circle();         // 참조 변수 선언과 객체 생성
        donut.radius = 2;                    // 도넛의 반지름을 2로 설정
        donut.name = "자바도넛";             // 도넛의 이름 설정
        area = donut.getArea();              // 도넛의 면적 알아내기
        System.out.println(donut.name + "의 면적은 " + area);
    }
}
```




```
자바피자의 면적은 314.0
자바도넛의 면적은 12.56
```

예제 4-2 : 두 개의 생성자를 가진 Circle 클래스

다음 코드는 2개의 생성자(디폴트 생성자와 생성자 오버를 가진 Circle 클래스이다. 실행 결과는 무엇인가?

```
public class Circle {
    int radius;
    String name;

    public Circle() { // 매개 변수 없는 생성자
        radius = 1; name = ""; // radius의 초기값은 1
    }
    public Circle(int r, String n) { // 매개 변수를 가진 생성자
        radius = r; name = n;
    }
    public double getArea() {
        return 3.14*radius*radius;
    }

    public static void main(String[] args) {
        Circle pizza = new Circle(10, "자바피자"); // Circle 객체 생성, 반지름 10

        double area = pizza.getArea();
        System.out.println(pizza.name + "의 면적은 " + area);

        Circle donut = new Circle(); // Circle 객체 생성, 반지름 1
        donut.name = "도넛피자";
        area = donut.getArea();
        System.out.println(donut.name + "의 면적은 " + area);
    }
}
```

생성자 이름은 클래스 이름과 동일

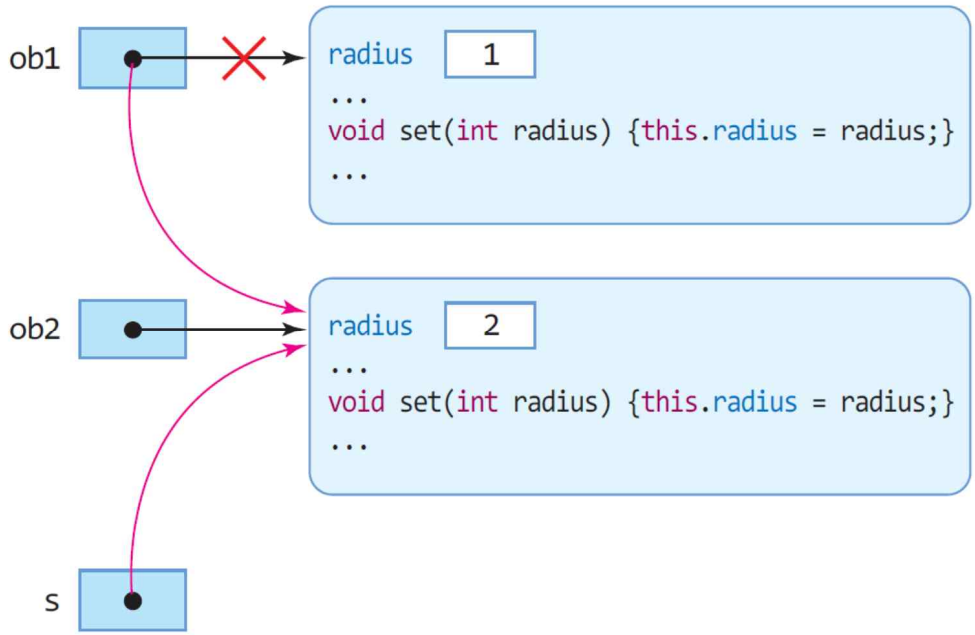
생성자는 리턴 타입 없음

```
자바피자의 면적은 314.0
도넛피자의 면적은 3.14
```

예제 4-3 : 객체의 치환

* 객체의 치환은 객체가 복사되는 것이 아니며 레퍼런스가 복사된다.

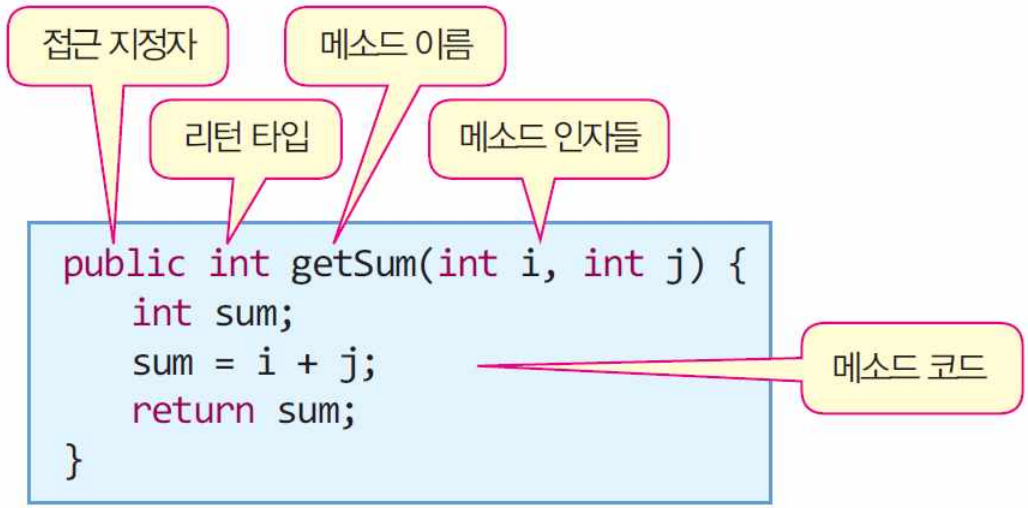
```
public class Circle {
   int radius;
   public Circle(int radius) { this.radius = radius; }
   public void set(int radius) { this.radius = radius; }
   public static void main(String [] args) {
      Circle ob1 = new Circle(1);
      Circle ob2 = new Circle(2);
      Circle s;

      s = ob2;
      ob1 = ob2; // 객체 치환
      System.out.println("ob1.radius=" + ob1.radius);
      System.out.println("ob2.radius=" + ob2.radius);
   }
}
```




```
ob1.radius=2
ob2.radius=2
```

메소드 형식

- 메소드
  - 클래스의 멤버 함수, C/C++의 함수와 동일
  - 자바의 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)
- 메소드 구성 형식
  - 접근 지정자
    - public. private, protected, 디폴트(접근 지정자 생략된 경우)
  - 리턴 타입
    - 메소드가 반환하는 값의 데이터 타입



메서드 인자 전달

- 자바의 인자 전달 방식
  - 경우 1. 기본 타입의 값 전달
    - 값이 복사되어 전달
    - 메소드의 매개변수가 변경되어도 호출한 실인자 값은 변경되지 않음
  - 경우 2. 객체 혹은 배열 전달
    - 객체나 배열의 레퍼런스만 전달
      - 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아님
    - 메소드의 매개변수와 호출한 실인자 객체나 배열 공유

인자 전달 – 기본 타입의 값이 전달되는 경우

- 매개변수가 byte, int, double 등 기본 타입의 값일 때
  - 호출자가 건네는 값이 매개변수에 복사되어 전달. 실인자 값은 변경되지 않음

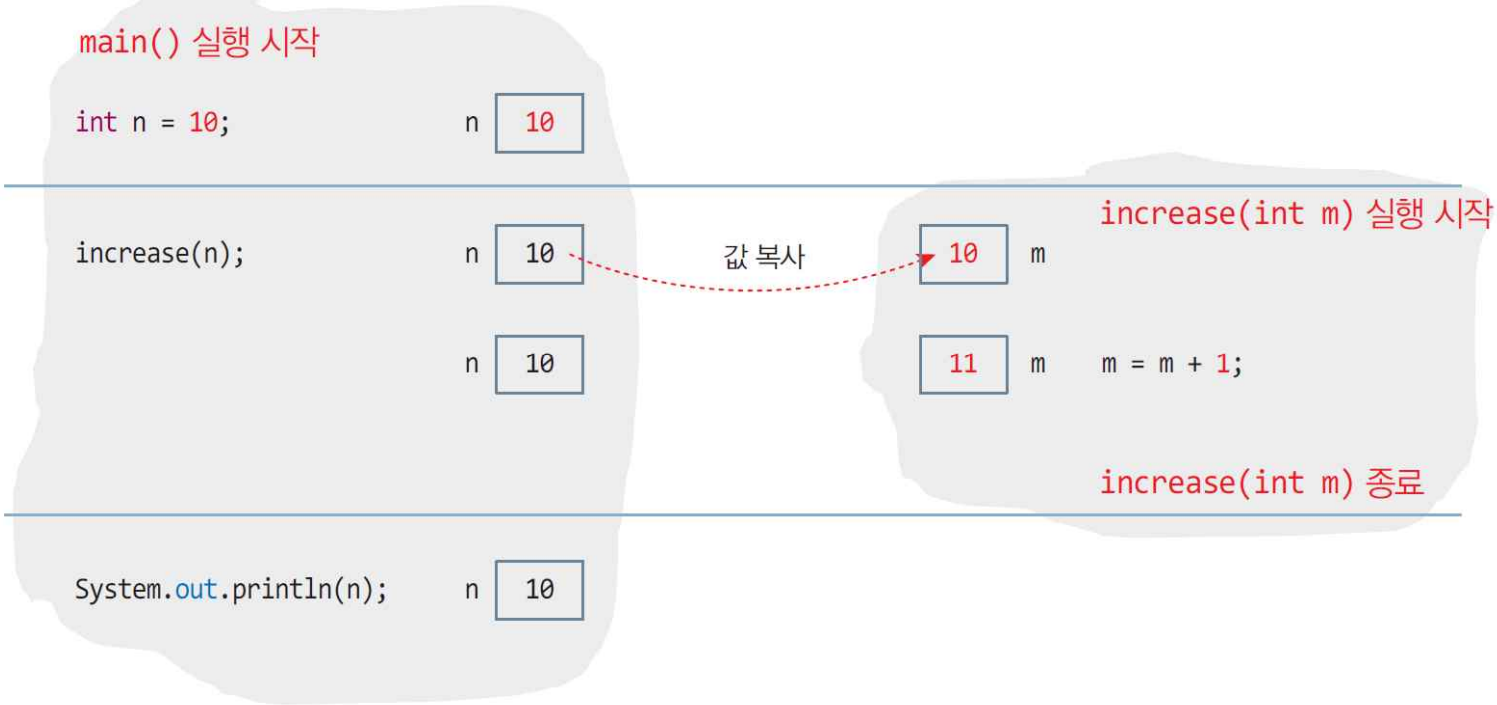
```
public class ValuePassing {
   public static void main(String args[]) {
      int n = 10;

      increase(n);

      System.out.println(n);
   }
```

호출

```
   static void increase(int m) {
      m = m + 1;
   }

}
```

실행 결과

```
10
```



인자 전달 – 객체가 전달되는 경우

- 객체의 레퍼런스만 전달
  - 매개 변수가 실인자 객체 공유

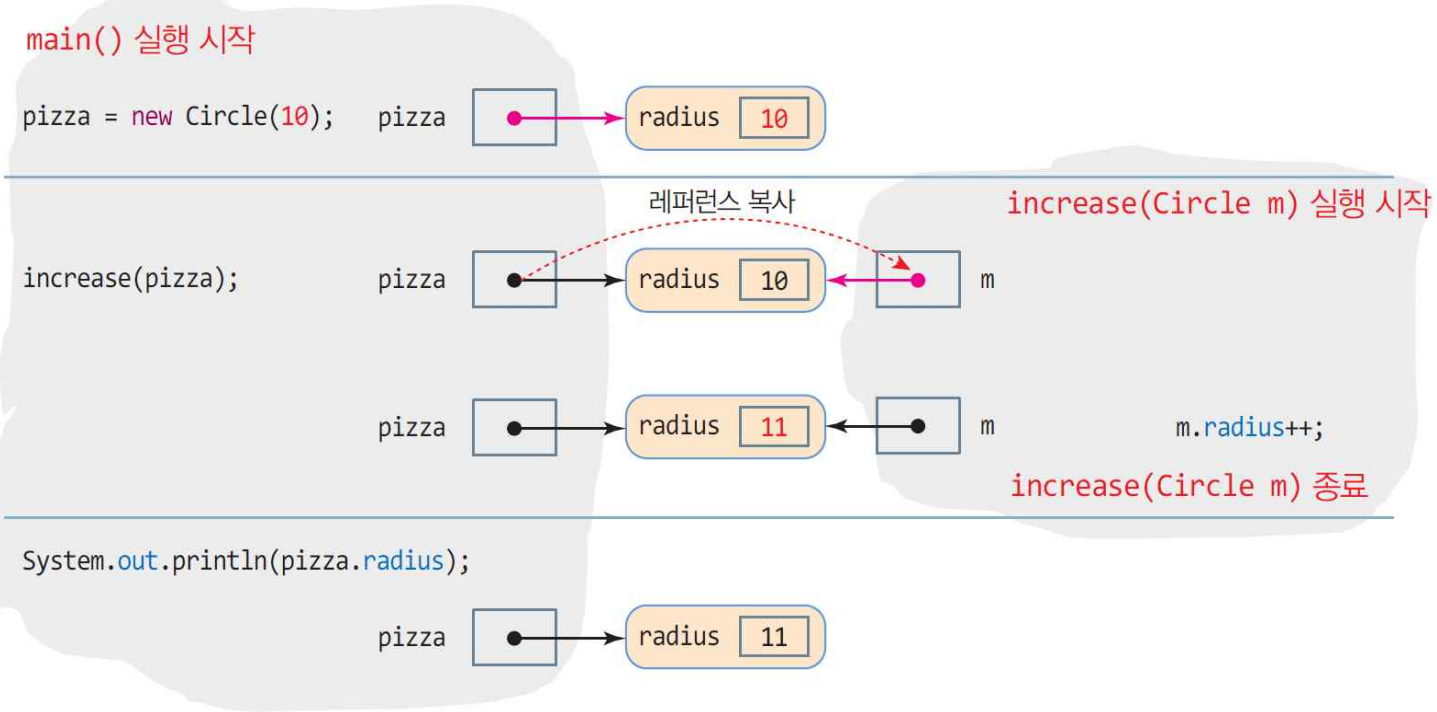
```
public class ReferencePassing {
   public static void main (String args[]) {
      Circle pizza = new Circle(10);

      increase(pizza);

      System.out.println(pizza.radius);
   }
```

호출

```
   static void increase(Circle m) {
      m.radius++;
   }

}
```

실행 결과

```
11
```



메소드 오버로딩

- 메소드 오버로딩(Overloading)
  - 이름이 같은 메소드 작성, 다음 2개의 조건
    - 매개변수의 개수나 타입이 서로 다르고
    - 이름이 동일한 메소드들
  - 리턴 타입은 오버로딩과 관련 없음
    - 오버로딩의 성공 여부를 따질 때 리턴 타입은 고려하지 않음

```
// 메소드 오버로딩이 성공한 사례

class MethodOverloading {
   public int getSum(int i, int j) {
      return i + j;
   }
   public int getSum(int i, int j, int k) {
      return i + j + k;
   }
}
```

```
// 메소드 오버로딩이 실패한 사례

class MethodOverloadingFail {
   public int getSum(int i, int j) {
      return i + j;
   }
   public double getSum(int i, int j) {
      return (double)(i + j);
   }
}
```

두 개의 getSum() 메소드는 매개변수의 개수, 타입이 모두 같기 때문에 메소드 오버로딩 실패

non-static 멤버와 static 멤버의 차이

	non-static 멤버	static 멤버

선언	class Sample {
	  int n;
	  void g() {…}
	}
		class Sample {
		  static int m;
		  static void f() {…}
		}

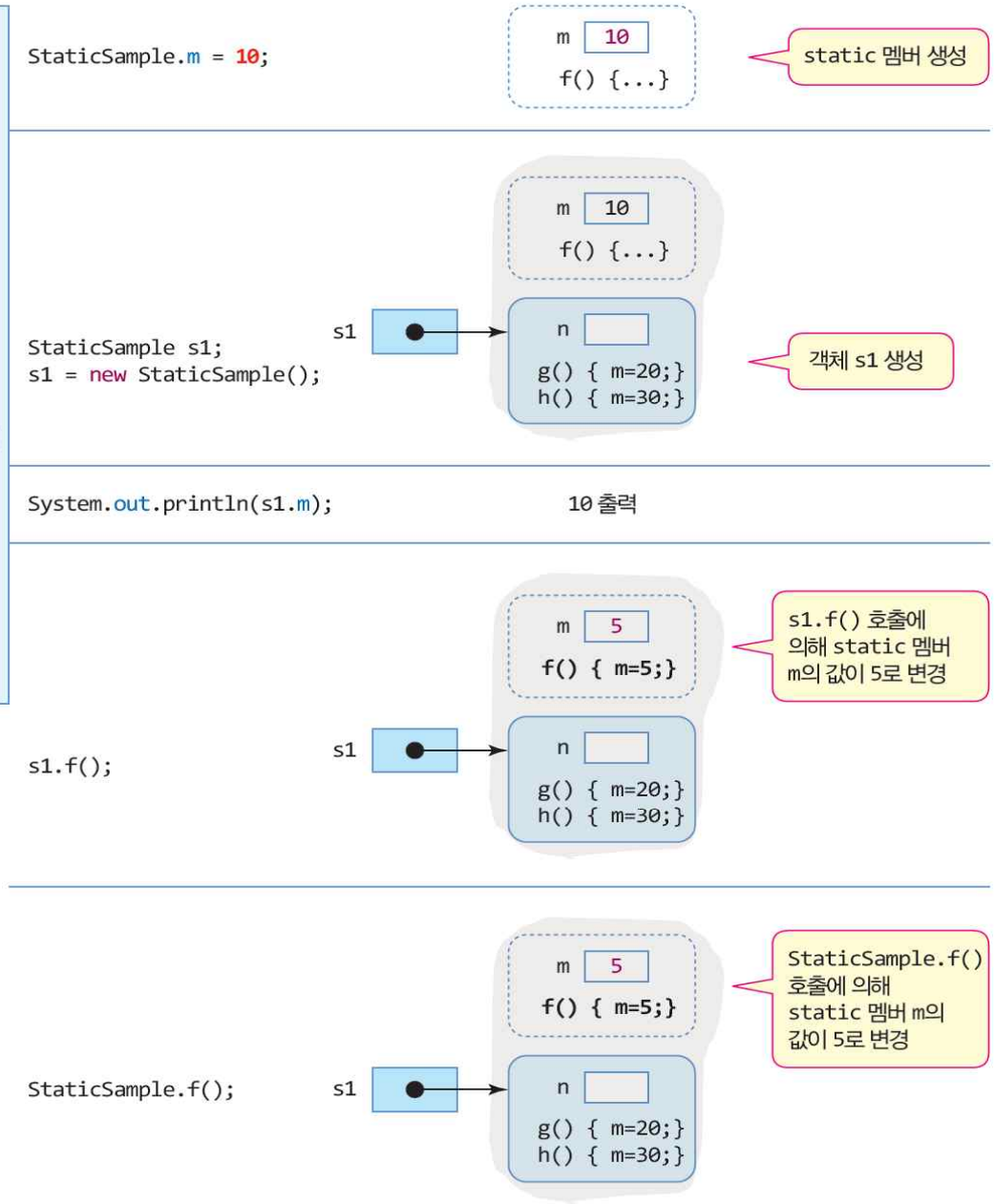
공간적 특성	멤버는 객체마다 별도 존재 • 인스턴스 멤버라고 부름	멤버는 클래스당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간(클래스 코드가 적재되는 메모리)에 생성 • 클래스 멤버라고 부름
시간적 특성	객체 생성 시에 멤버 생성됨 • 객체가 생길 때 멤버도 생성 • 객체 생성 후 멤버 사용 가능 • 객체가 사라지면 멤버도 사라짐	클래스 로딩 시에 멤버 생성 • 객체가 생기기 전에 이미 생성 • 객체가 생기기 전에도 사용 가능 • 객체가 사라져도 멤버는 사라지지 않음 • 멤버는 프로그램이 종료될 때 사라짐
공유의 특성	공유되지 않음 • 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

예제 4-4 : static 멤버를 클래스 이름으로 접근

```
class StaticSample {
    public int n;
    public void g() {
        m = 20;
    }
    public void h() {
        m = 30;
    }
    public static int m;
    public static void f() {
        m = 5;
    }
}

public class Ex {
    public static void main(String[] args) {
        StaticSample.m = 10;

        StaticSample s1;
        s1 = new StaticSample();
        System.out.println(s1.m);
        s1.f();
        StaticSample.f();
    }
}
```

실행 결과

```
10
```



[quiz4_1_식별자] Rectangle 클래스를 작성하시오.

너비와 높이를 입력 받아 사각형의 합을 출력하는 프로그램을 작성하라. 너비(width)와 높이(height) 필드, 그리고 면적 값을 제공하는 getArea() 메소드를 가진 Rectangle 클래스를 작성하고 활용하시오.

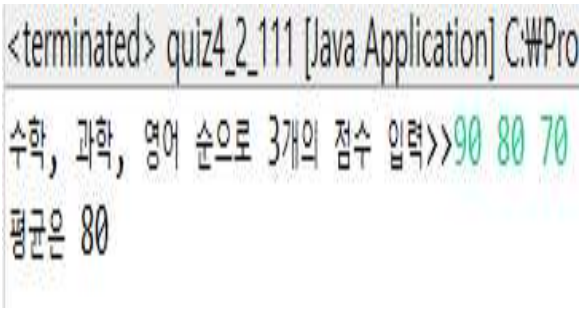
[실행결과]
>> 4 5
사각형의 면적은 20


```
package w2_111;
import java.util.Scanner;
class Rectangle {
		int ???;
		int ???;
		public int getArea() {
				???	;
		}
}
public class quiz4_1_111 {
		public static void main(String[] args) {
				???		;// 객체 생성
				Scanner scanner = new Scanner(System.in);
				System.out.print(">> ");
				rect.width = scanner.???();
				rect.height = scanner.???();
				System.out.println("사각형의 면적은 " + ???.???());
				scanner.close();
		}

}
```

[quiz4_2_식별자] Grade 클래스를 작성하시오

3과목의 평균 성적을 구하는 Grade 클래스를 정의하시오.

- 3개의 과목은 int 타입의 math, science, English 필드에 저장하고 각 필드의 접근 지정자는 private으로 선언
- 생성자 오버로딩으로 입력 받은 3개의 과목을 각 필드에 저장하는 생성자를 정의
- 세 과목의 평균을 구해 리턴하는 average()메서드를 정의
- main 메서드에서 키보드로 3과목을 입력 받아 객체 생성 시 각 필드에 입력 값을 저장하고 평균 값을 출력



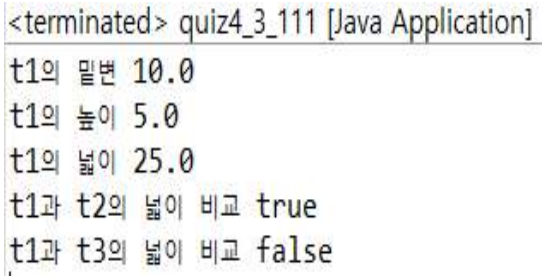

```
package w2_111;
import java.util.Scanner;
class Grade {


                //???  여러 문장으로 클래스의 멤버를 작성



}
public class quiz4_2_111 {
            public static void main(String[] args) {


            //???  여러 문장으로 클래스의 멤버를 작성



            }
   }
```

[quiz4_3_식별자] Triangle 클래스를 작성하시오

삼각형을 나타내는 Triangle 클래스를 작성하시오.

- 삼각형의 속성으로는 실수값의 **밑변(base)와 높이(height)**을 동일 클래스에서만 접근하도록(캡슐화) 하고
- 이 속성(필드)을 접근할 수 있는 **접근자**와 **생성자**도 정의하고 **넓이를 구하는 메서드(findArea())**도 정의하고
- 또한 2개의 삼각형 넓이가 동일한지 비교하는 **isSameArea( ) 메서드도** 정의하시오.

[실행결과]
t1의 밑변 10.0
t1의 높이 5.0
t1의 넓이 25.0
t1과 t2의 넓이 비교 true
t1과 t3의 넓이 비교 false




```
package w2_111;
class Triangle {


        //???  여러 문장으로 클래스의 멤버를 작성



}
public class quiz4_3_111 {
                public static void main(String[] args) {
                                Triangle t1 = new Triangle();
                                Triangle t2 = new Triangle(5.0, 10.0);
                                Triangle t3 = new Triangle(8.0, 8.0);
                System.out.println("t1의 밑변 " + ??? );
                System.out.println("t1의 높이 " + ??? );
                System.out.println("t1의 넓이 " + t1.findArea());
                System.out.println("t1과 t2의 넓이 비교 " + t1.isSameArea(t2));
                System.out.println("t1과 t3의 넓이 비교 " + t1.isSameArea(t3));
                }
  }
```