

객체지향 프로그래밍

7장

클래스

조용주

ycho@smu.ac.kr

클래스 찾기

- 클래스는 무엇이나 될 수 있지만, 무엇이나 될 수 없음
 - 객체 지향으로 프로그래밍하려면 "클래스"를 구현
 - 자바는 단순한 명령문을 실행하려고 해도 무조건 "클래스"를 구현해야 함
 - JShell에서는 클래스 없이 함수도 만들고 호출하기도 했지만 자바 컴파일러를 사용한다면 클래스부터 만들어야 함
 - 어떤 것들을 클래스로 만들어야 하고, 문제가 주어졌다면 클래스를 어떻게 찾아야 할까?

좋은 클래스란?

- 좋은 클래스를 구현하려면 "좋은" 이라는 단어의 의미를 고려해야 함
 - 다른 프로그램에서도 사용할 수 있음
 - 확장이 쉬움
 - 고쳐 쓰기 쉬움
- 재사용성이 높고 확장 및 수정이 수월한 클래스를 잘 구현하려면 어떻게 해야 할까?
 - 일관성(또는 응집성)과 결합성을 기준으로 사용

기준	설명
일관성	클래스는 목적이 있어야 하고 그 목적에 부합해야 함
결합성	클래스는 다른 클래스에 대한 의존이 적어야 함

일관성 또는 응집성

□ 일관성 또는 응집성

- 클래스를 구성하는 요소들이 뭉쳐있는 정도 및 연관성을 의미
- 클래스의 목적을 이룰 수 있도록 관련된 데이터와 그 데이터를 처리하는 함수들만으로 구성
- 클래스가 어떤 **자료(데이터)**를 가지고 있어야 하는지 **먼저 생각**
- 그런 후에 그 자료를 처리하는 코드를 **멤버 함수**로 구현

결합성 (Coupling)

- 결합은 클래스가 서로 연결되면서 발생
- 클래스를 설계할 때에는 서로간의 결합성이 낮아야 함 → **loosely coupled** 또는 **weakly coupled**
- 결합성을 낮춘다는 것
 - 특정 클래스 내부 코드의 변경이 다른 클래스에 미치는 영향을 적다는 것
 - 내부 코드를 변경해도 사용성에 영향을 미치지 않음
 - 클래스를 단순화

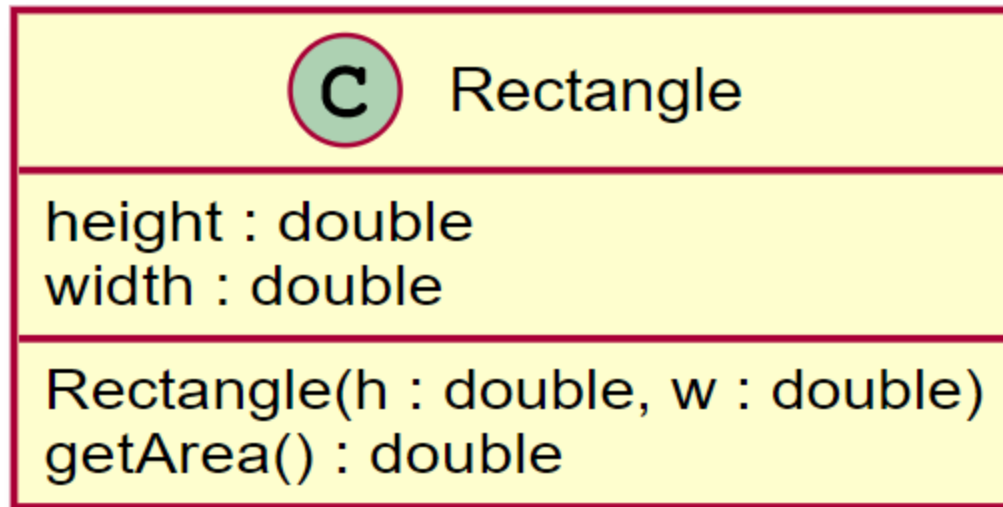
명사 인식

□ 문제에서 클래스 후보를 고르는 방법

- 문제에 있는 명사를 후보로 인식
- 클래스 후보 가운데 서로 포함되는 명사가 있으면 멤버 데이터로 인식
- 클래스가 수행하는 기능보다 데이터를 먼저 고려
- "응집성"에 따라 함께 묶을 수 있는 연관된 데이터들을 | 용해서 클래스를 구성
- 프로그램의 대상이 되는지 관련이 있는지 클래스로 인식하는 기준이 됨

사례 1

- 사각형의 면적을 계산
 - 클래스 후보: 사각형, 면적
- 사각형 클래스



사례 2: "사람"이라는 클래스

□ 사람

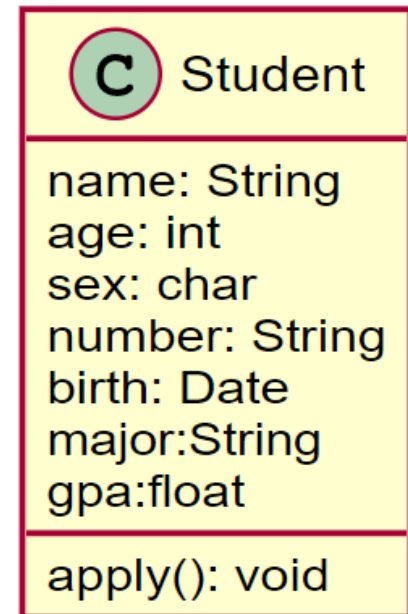
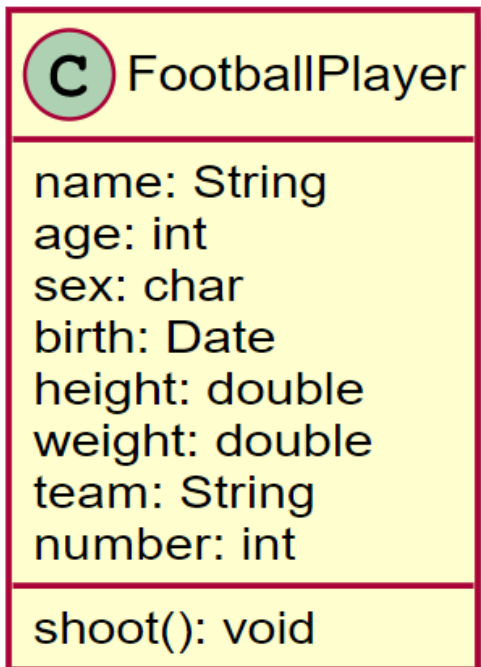
- 이름, 나이, 생년월일, 성별 등

□ 프로 축구 선수

- 키, 몸무게, 구단, 번호

□ 학생

- 학과, 학번, 학점

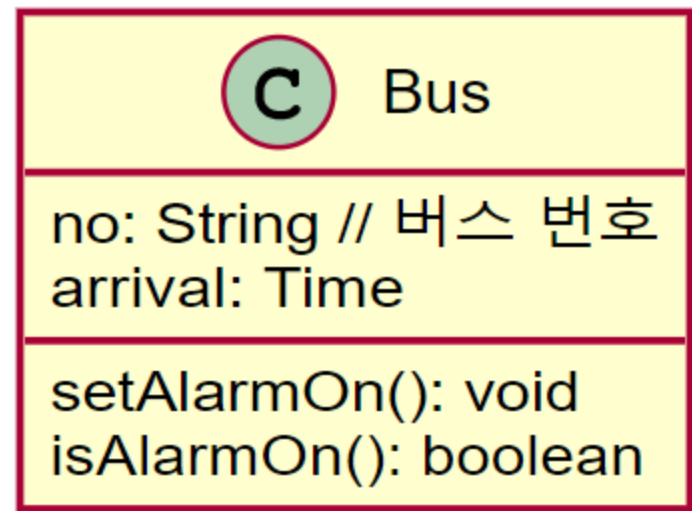


사례 3: "오늘은 버스를 타고 집에 가서 대표팀 축구 경기를 보려고 한다. 버스 시간 알려주기"

□ 문제를 정확하게 정의

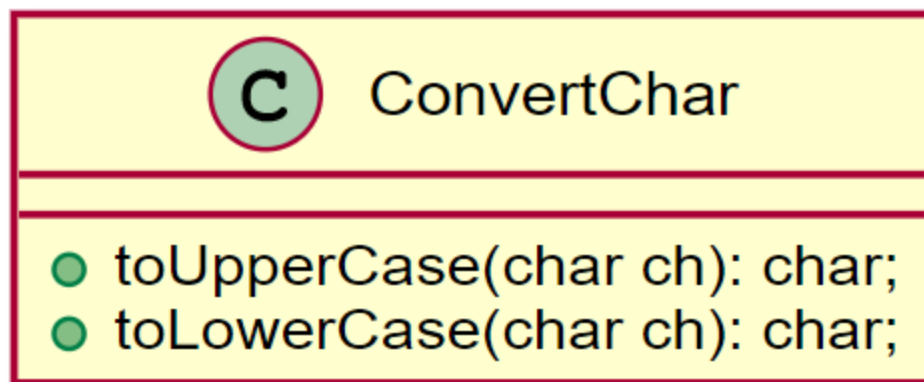
- "축구 경기"가 언제 어느 채널에서 방송되는지를 프로그램할지
- 축구 경기를 보러 가는 사람을 위해 버스 도착 시각을 알려주는 프로그램을 구현할지

□ Bus 클래스를 구현



사례 4: 멤버 변수 없는 클래스

- 멤버 변수가 없는 클래스는 기능만 제공 (유틸리티 클래스)
 - 객체를 생성하지 않고 사용 가능
 - 관련된 메소드를 모아놓음



- 자바에도 **Math** 클래스가 대표적인 유틸리티 클래스
 - 원주율 값 (상수로 제공)
 - 제곱근, `random()` 함수 등이 데이터와 관련 없이 존재함

캡슐화와 사용 권한 제어

□ 캡슐화의 정의와 필요성

■ 클래스의 캡슐화

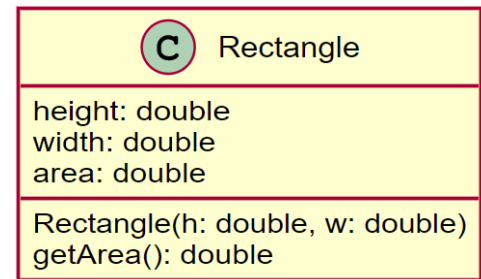
- 관련된 데이터들과 그 데이터를 사용하는 함수들을 함께 클래스에 넣는 것
- 인터페이스를 제공해서 클래스를 사용할 수 있게 함

□ 실생활에서의 캡슐화

- 의약품
- 자동차
- 일반 가전 제품 (냉장고, TV)

캡슐화와 사용 권한 제어

- 캡슐화는 단순히 데이터와 함수들을 묶은 것이 아님
- 캡슐화의 목적은 **사용자가 내부 데이터를 함부로 수정하지 못하도록 만들어서** 클래스가 포함하는 데이터의 **응집성과 무결성**(내부 데이터가 잘못 수정되지 않도록 하는 것)을 **보장**
- 또 다른 목적은 내부 동작 방법을 몰라도 사용 가능
- 사각형 클래스를 다시 생각해볼 것
 - 면적을 매번 계산하는 것은 비효율적 → 변수에 저장
 - 수정된 Rectangle 클래스



캡슐화와 사용 권한 제어

```
// Rectangle.java
class Rectangle {
    double height;
    double width;
    double area;

    Rectangle(double h, double w) {
        height = h;
        width = w;
        area = w * h;
    }

    double getArea() {
        return area;
    }
}
```

캡슐화와 사용 권한 제어

```
// TestRectangle.java
class TestRectangle {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(10, 15);
        double area = rect.getArea();
        System.out.println("area = " + area);
    }
}
```

□ 실행 결과

```
C:\Code\java\07>java TestRectangle
area = 150.0
```

캡슐화와 사용 권한 제어

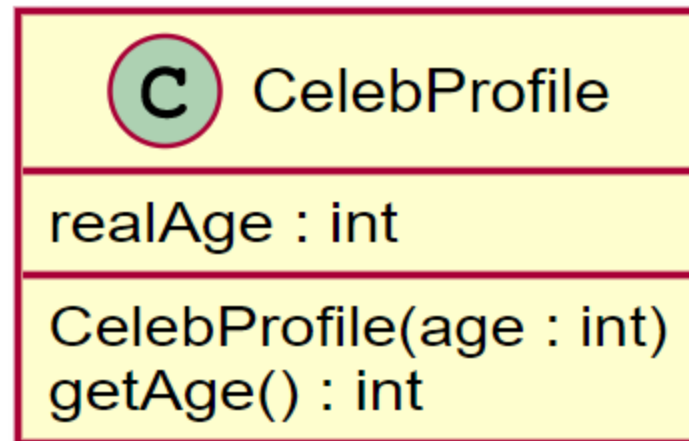
- 만약 사용자가 width 또는 height 변수값을 수정하면?
- 새로 수정한 코드를 컴파일하고 실행시키기

```
// TestRectangle2.java
class TestRectangle2 {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(10, 15);
        double area = rect.getArea();
        System.out.println("area = " + area);
        rect.width = 20;
        area = rect.getArea();
        System.out.println("new area = " + area);
    }
}
```

```
C:\Code\java\07>java TestRectangle2
area = 150.0
new area = 150.0
```

연예인 프로필 클래스 만들기

- 지금은 아니겠지만, 2000년대 초반만 해도 연예인들이 나이를 줄이거나 늘려서 얘기하는 경우들이 있음
- 연예인 프로필을 나타내는 CelebProfile 클래스 구현
 - 나이 정보만 포함하고 외부에 드러내는 나이는 실제 나이보다 2살 아래인 것으로 처리
 - 생성자에 전달되는 나이 또는 getAge()에서 반환하는 대외적으로 보이는 나이
- 클래스 다이어그램



연예인 프로필 클래스 만들기

```
// CelebProfile.java
class CelebProfile {
    int realAge;

    CelebProfile(int age) {
        realAge = age + 2;
    }
    int getAge() {
        return realAge - 2;
    }
}
```

- 이 클래스에서는 멤버 변수 realAge를 절대적으로 보호해야 함
- 외부에서는 접근할 수 없도록 하고, 나이를 확인하고 싶으면 getAge()를 사용하도록 해야 함

접근 제어자와 가시성

□ 캡슐화

- 클래스를 설계할 때부터 사용자가 접근할 수 있는 멤버 변수와 함수들을 명확하게 파악해야 함
- 캡슐화를 지원하기 위해 객체 지향 프로그래밍에서는 클래스의 "틀"이 **데이터를 가지고 있고 사용 권한을 제어**할 수 있어야 함
- 또한 멤버 함수들도 외부에 공개하지 말아야 하는 것들이 있다면 사용하지 못하게 막아야 함
- 클래스를 설계하고 구현하는 프로그래머가 클래스 외부에서는 특정 멤버 변수와 메소드만을 사용할 수 있도록 지정 → 가시성(visibility) 또는 접근성(accessibility)

접근 제어자와 가시성

- 클래스의 변수나 함수들의 접근성을 지정하는 키워드를 접근 지시자(access specifier) 또는 접근 제어자(access modifier)라고 부름

구분	UML 표시	설명	클래스 내부	패키지 내부	서브 클래스	모든 클래스
public	+	누구나	Y	Y	Y	Y
protected	#	자손, 패키지	Y	Y	Y	N
default (package private)	~	패키지	Y	Y	N	N
private	-	나 자신	Y	N	N	N

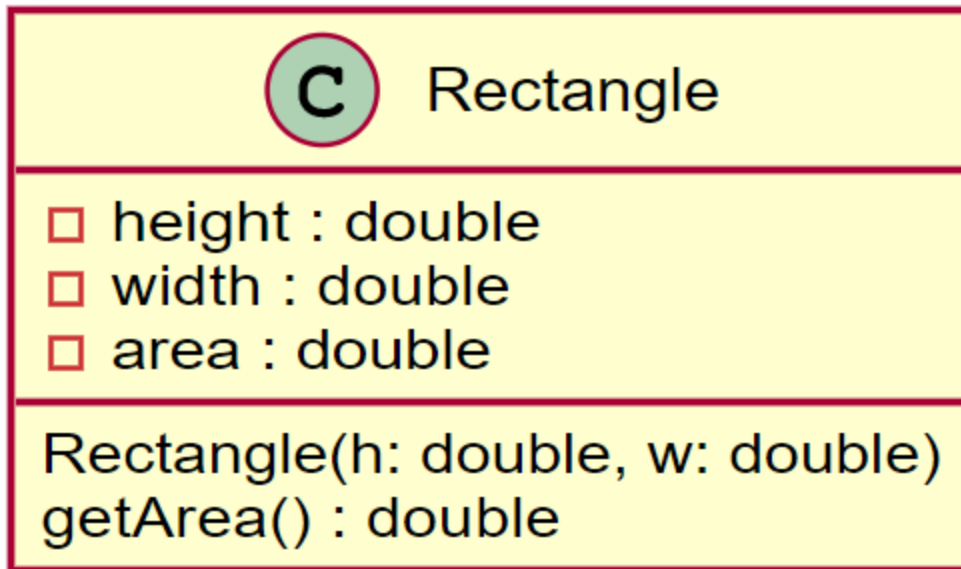
private 접근 제어자

- private으로 지정된 멤버 변수나 함수
 - 클래스 내부에서 자유롭게 사용 가능
 - 클래스 외부에서는 접근(사용) 불가

```
@startuml
class Rectangle {
    -height : double
    -width : double
    -area : double
    Rectangle(h: double, w: double)
    getArea() : double
}
@enduml
```

private 접근 제어자

□ UML 클래스 다이어그램



private 접근 제어자

□ Rectangle 클래스 구현

```
class Rectangle {  
    private double height;  
    private double width;  
    private double area;  
  
    Rectangle(double h, double w) {  
        height = h;  
        width = w;  
        area = w * h;  
    }  
  
    double getArea() {  
        return area;  
    }  
}
```

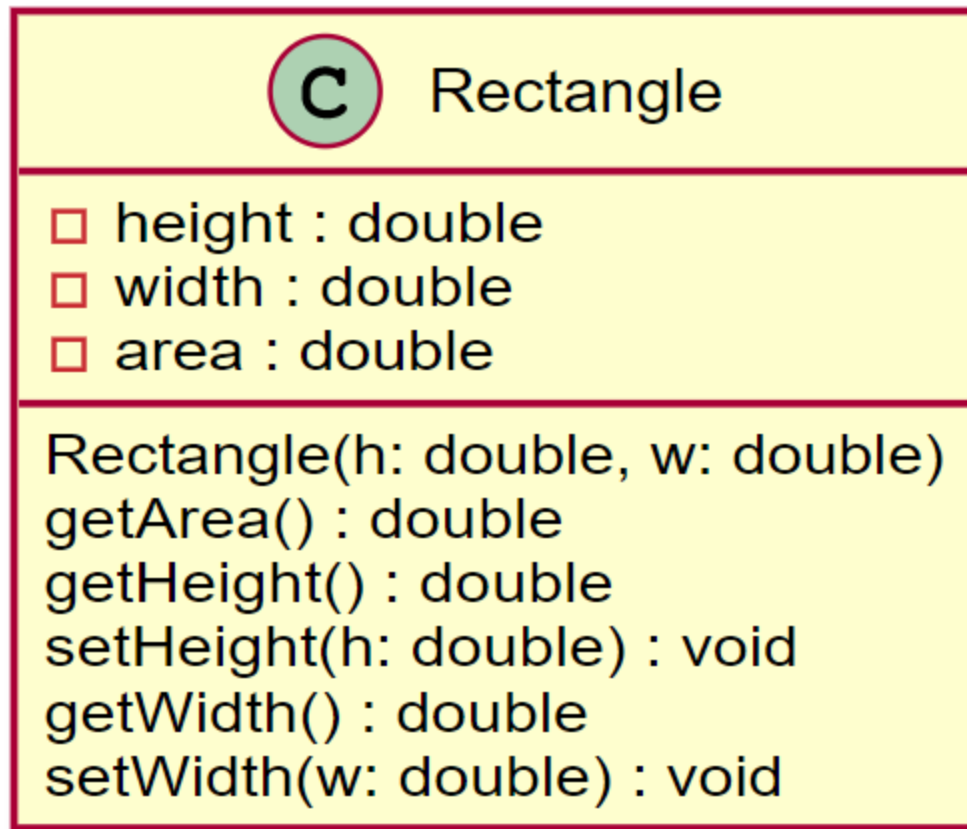
private 접근 제어자

- TestRectangle2.java 다시 컴파일 하면 오류 발생

```
C:\Code\java\07>javac
TestRectangle2.java
TestRectangle2.java:7: error: width
has private access in Rectangle
    rect.width = 20;
        ^
1 error
```

private 접근 제어자

- ❑ 오류를 제거하려면 get 함수나 set 함수 등을 제공
- ❑ 수정된 Rectangle 클래스 다이어그램

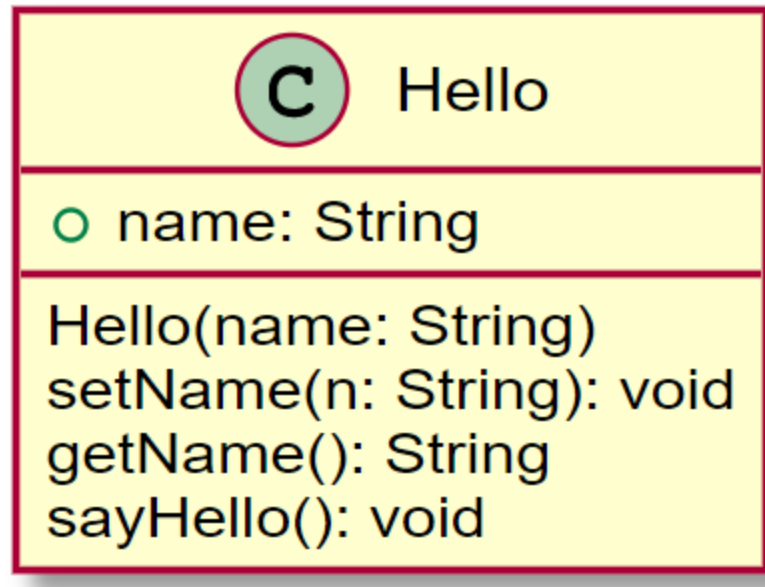


private 접근 제어자

```
class Rectangle {  
    ...    // 기존 Rectangle 코드  
  
    double getHeight() { return height; }  
  
    void setHeight(double h) {  
        height = h;  
        area = width * height;  
    }  
  
    double getWidth() { return width; }  
    void setWidth(double w) {  
        width = w;  
        area = width * height;  
    }  
}
```

public 접근 제어자

- public으로 지정된 멤버 변수나 함수
 - 클래스 내/외부에서 자유롭게 사용 가능
- Hello 클래스의 속성을 public으로 지정



```
// TestPublicHello.java
```

```
class Hello {
```

```
    public String name;
```

```
    Hello(String name) { this.name = name; }
```

```
    void setName(String n) { name = n; }
```

```
    String getName() { return name; }
```

```
    void sayHello() {
```

```
        System.out.println("hello " + name);
```

```
    }
```

```
}
```

```
class TestPublicHello {
```

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

```
        Hello h = new Hello("world");
```

```
        h.sayHello();
```

```
        h.setName("jsl");
```

```
        h.sayHello();
```

```
        h.name = "Yongjoo Cho";
```

```
        h.sayHello();
```

```
    }
```

```
}
```

```
C:\Code\java\07>java TestPublicHello
```

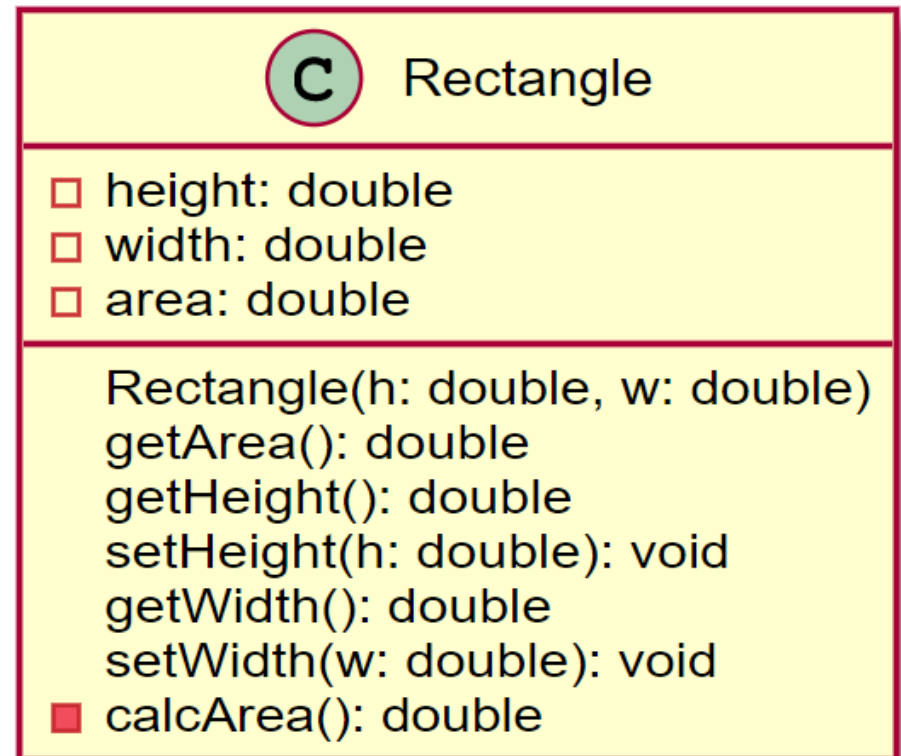
```
hello world
```

```
hello jsl
```

```
hello Yongjoo Cho
```

멤버 함수, 생성자와 접근 제어자

- 접근 제어자는 멤버 함수나 생성자 등을 제한시키는데 사용 가능
 - 멤버 변수와 똑같은 형태로 접근 제한
 - 예: private으로 지정된 메소드들은 클래스 내부에서만 사용 가능
- Rectangle 클래스에 calcArea() 함수를 추가 (내부적으로만 사용함)
- 클래스 다이어그램



멤버 함수, 생성자와 접근 제어자

```
// TestPublicHello.java
class Rectangle {
    ... // 이전 코드와 같음
    private void calcArea() {
        area = width * height;
    }
}

class TestRectangle3 {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(10, 15);
        double area = rect.getArea();
        System.out.println("면적 = " + area);
        // rect.calcArea(); // 오류
    }
}
```

멤버 함수, 생성자와 접근 제어자

- Hello 클래스를 수정해서 set 함수와 get 함수를 private으로 지정해보고 컴파일

```
// TestHello2.java
class Hello {
    public String name;
    public Hello(String name) { this.name = name; }
    private void setName(String n) { name = n; }
    private String getName() { return name; }
    public void sayHello() {
        System.out.println("hello " + name);
    }
}

class TestHello2 {
    public static void main(String[] args) {
        Hello h = new Hello("world");
    }
}
```

```
        h.sayHello();  
        h.setName("jsl");  
        System.out.println(h.getName());  
        h.name = "Yongjoo Cho";  
        h.sayHello();  
    }  
}
```

□ 컴파일 결과

```
C:\Code\java\07>javac TestHello2.java  
TestHello2.java:22: error: setName(String) has  
private access in Hello  
    h.setName("jsl");  
      ^  
TestHello2.java:23: error: getName() has private  
access in Hello  
    System.out.println(h.getName());  
                        ^  
2 errors
```

멤버 함수, 생성자와 접근 제어자

▣ 생성자만 private으로 바꾸면?

```
// TestHello3.java
class Hello {
    public String name;
    private Hello(String name) { this.name = name; }
    void setName(String n) { name = n; }
    String getName() { return name; }
    public void sayHello() {
        System.out.println("hello " + name);
    }
}

class TestHello3 {
    public static void main(String[] args) {
        Hello h = new Hello("world");
        h.sayHello();
    }
}
```


멤버 함수, 생성자와 접근 제어자

□ 컴파일 오류

```
C:\Code\java\07>javac TestHello3.java
TestHello3.java:20: error: Hello(String) has private
access in Hello
    Hello h = new Hello("world");
                ^
```

1 error

접근 제어자가 없는 멤버 변수, 멤버 함수, 생성자

- 접근 제어자가 지정되지 않으면 default(package private)이라고 가정
- default로 지정된 변수, 함수, 생성자 등은 클래스 내부의 다른 메소드와 같은 패키지(package)에 있는 클래스의 함수에서 접근 가능
 - 단순히 같은 폴더(디렉토리)에 있는 다른 클래스에서 접근 가능
 - JShell에서 작성되는 코드들도 같은 패키지로 인식됨
 - 따라서 접근 제어자를 지정하지 않고, 객체를 생성하고 속성 또는 멤버 함수들을 자유롭게 사용할 수 있음

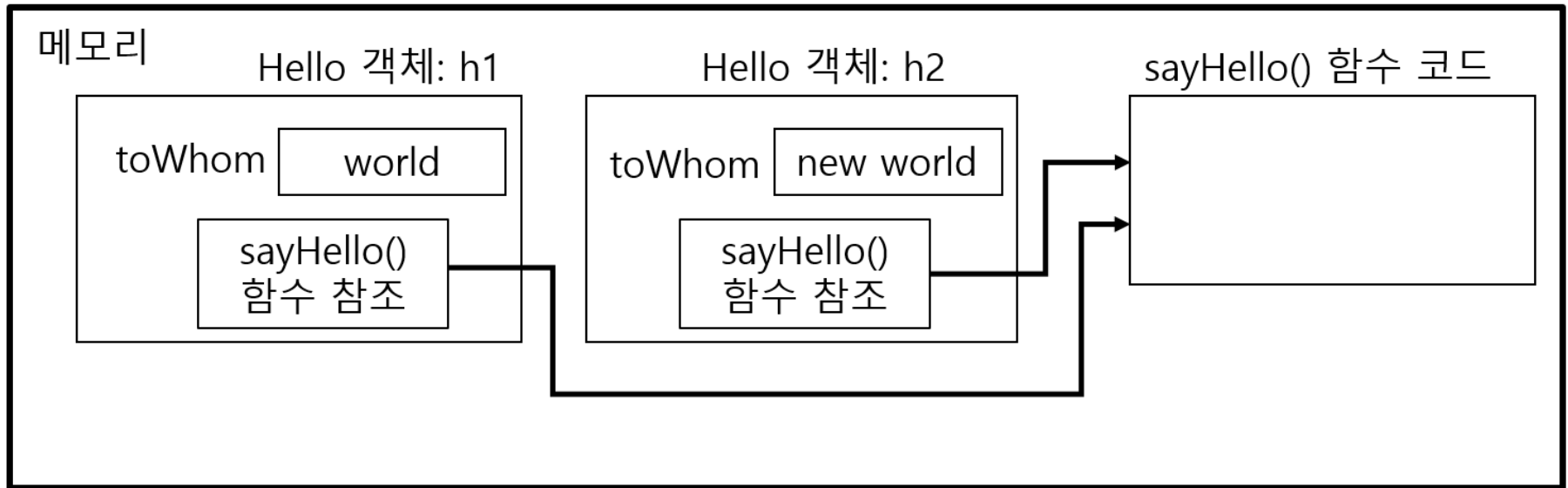
정적 메소드(Static Method)

- ▣ 클래스를 사용하려면 객체를 생성해야 했음

```
class Hello {  
    private String toWhom;  
    public Hello(String whom) {  
        this.toWhom = whom;  
    }  
    public void sayHello() {  
        System.out.println("hello " + toWhom);  
    }  
  
    public static void main(String args[]) {  
        Hello h1 = new Hello("world");  
        h1.sayHello();  
        Hello h2 = new Hello("new world");  
        h2.sayHello();  
    }  
}
```

정적 메소드

▣ 객체 생성했을 때 메모리 구조



정적 메소드

▣ ConvertChar 클래스

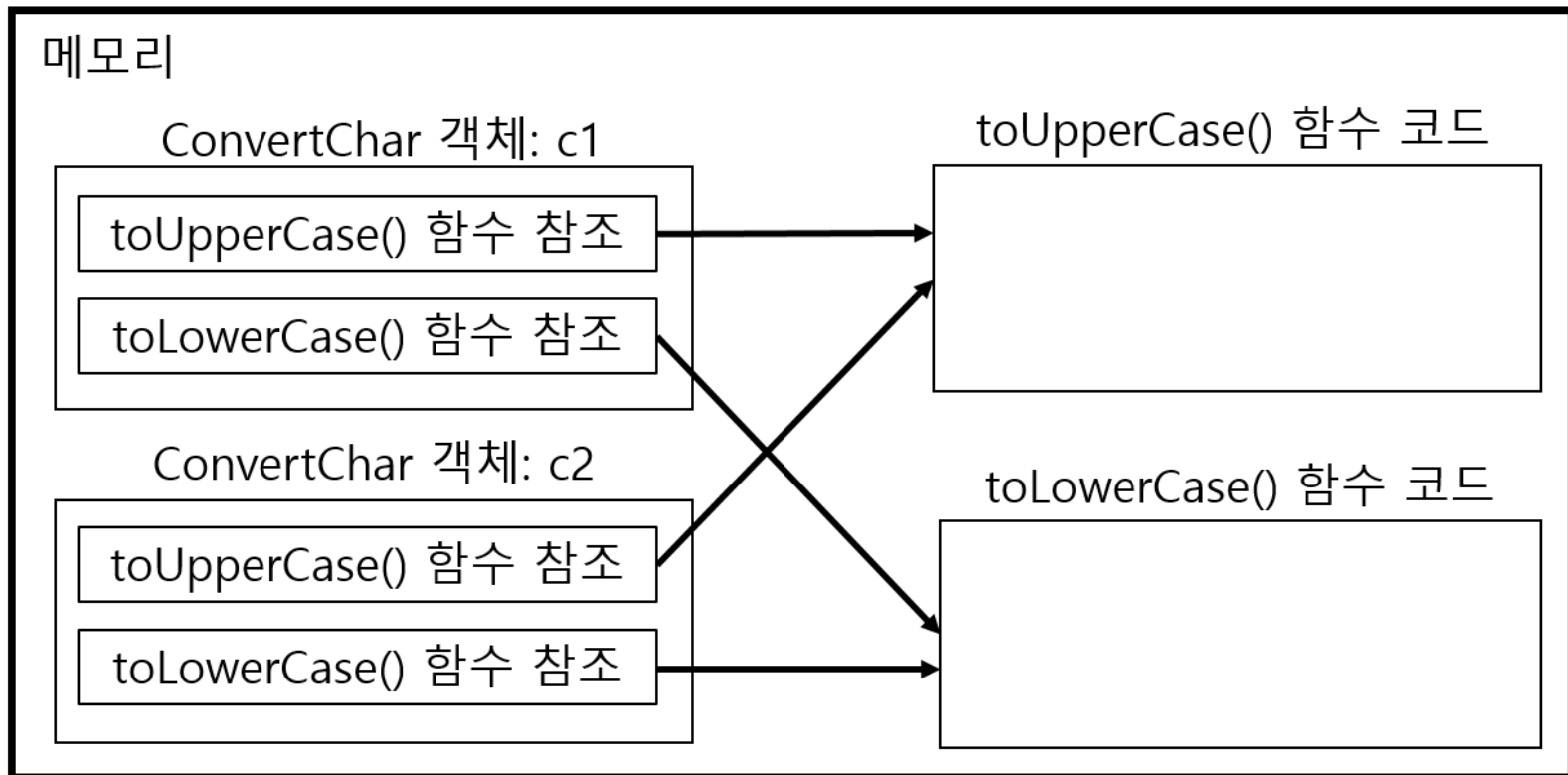
```
// ConvertChar.java
class ConvertChar {
    char toUpperCase(char ch) {
        if (ch >= 'a' && ch <= 'z')
            return (char) ('A' + (int) ch - (int) 'a');
        return ch;
    }

    char toLowerCase(char ch) {
        if (ch >= 'A' && ch <= 'Z')
            return (char) ('a' + (int) ch - (int) 'A');
        return ch;
    }
}
```

```

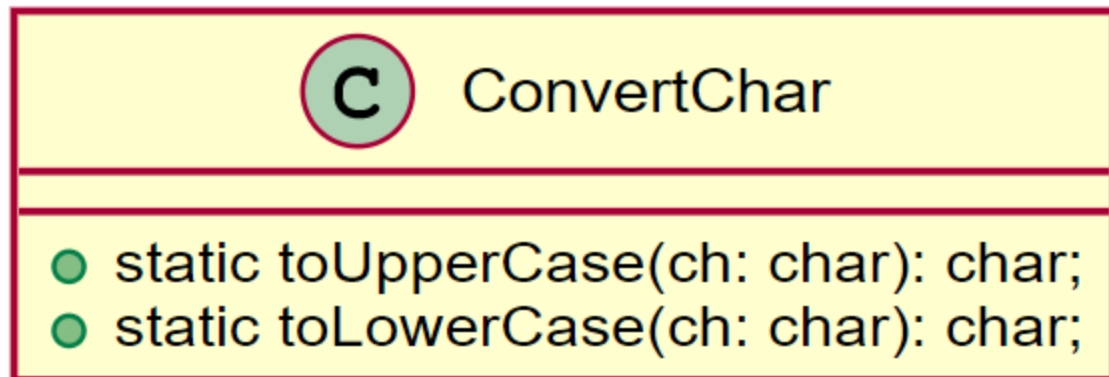
public static void main(String[] args) {
    ConvertChar c1 = new ConvertChar();
    System.out.println(c1.toUpperCase('c'));
    System.out.println(c1.toLowerCase('C'));
    ConvertChar c2 = new ConvertChar();
    System.out.println(c2.toUpperCase('h'));
    System.out.println(c2.toLowerCase('H'));
}
}

```



정적 메소드

- 자바는 객체를 생성하지 않고 클래스의 메소드를 직접 호출할 수 있는 정적 메소드 기능 제공
- 정적 메소드를 호출하는 것은 클래스_이름.메소드_이름 또는 객체_이름.메소드_이름 형태로 사용
- ConvertChar 클래스에 정적 메소드를 추가한 클래스 다이어그램



정적 메소드

```
// ConvertChar.java
class ConvertChar {
    public static char toUpperCase(char ch) {
        if (ch >= 'a' && ch <= 'z')
            return (char) ('A' + (int) ch - (int) 'a');
        return ch;
    }

    public static char toLowerCase(char ch) {
        if (ch >= 'A' && ch <= 'Z')
            return (char) ('a' + (int) ch - (int) 'A');
        return ch;
    }
}
```



```
// TestConvertChar.java
class TestConvertChar {
    public static void main(String[] args) {
        System.out.println(ConvertChar.toUpperCase('c'));
        System.out.println(ConvertChar.toLowerCase('C'));
        System.out.println(ConvertChar.toUpperCase('h'));
        System.out.println(ConvertChar.toLowerCase('H'));
    }
}
```

▣ 객체를 생성해서 일반 메소드처럼 호출 가능

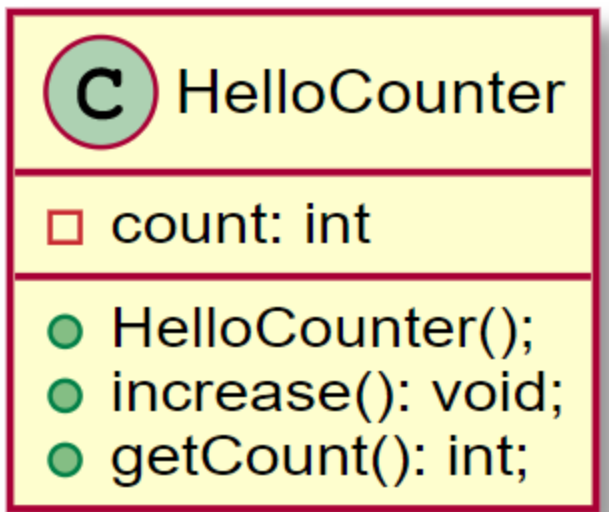
```
// TestConvertChar2.java
class TestConvertChar2 {
    public static void main(String[] args) {
        ConvertChar c1 = new ConvertChar();
        System.out.println(c1.toUpperCase('c'));
        System.out.println(c1.toLowerCase('C'));
        System.out.println(c1.toUpperCase('h'));
        System.out.println(c1.toLowerCase('H'));
    }
}
```

정적 멤버 함수

- 정적 메소드만 있는 클래스들은 C/C++ 혹은 다른 언어의 전역 함수를 대체하기 위해 만들어짐
- 전역 함수란 프로그램 어디서나 사용할 수 있는 함수
- 자바의 일반 함수들은 클래스에 존재하기 때문에 객체가 생성되기 전에는 사용할 수 없음
- 정적 메소드는 객체 생성 없이 사용할 수 있어 전역 함수 대체 가능
- `Math.random()`은 이미 사용해봄
- 이 밖에 `Math` 클래스에는 많은 정적 메소드를 제공

정적 멤버 변수(static member variable)

- Hello 클래스 객체가 생성될 때마다 개수를 기록하고 화면에 출력하는 프로그램을 작성
- HelloCounter 클래스를 구현
- 클래스 다이어그램



```
class HelloCounter {
    private int count;
    public HelloCounter() {
        count = 0;
    }
    public void increase() {
        count++;
    }
    public int getCount() {
        return count;
    }
}
```

정적 멤버 변수(static member variable)

▣ 수정된 Hello 클래스

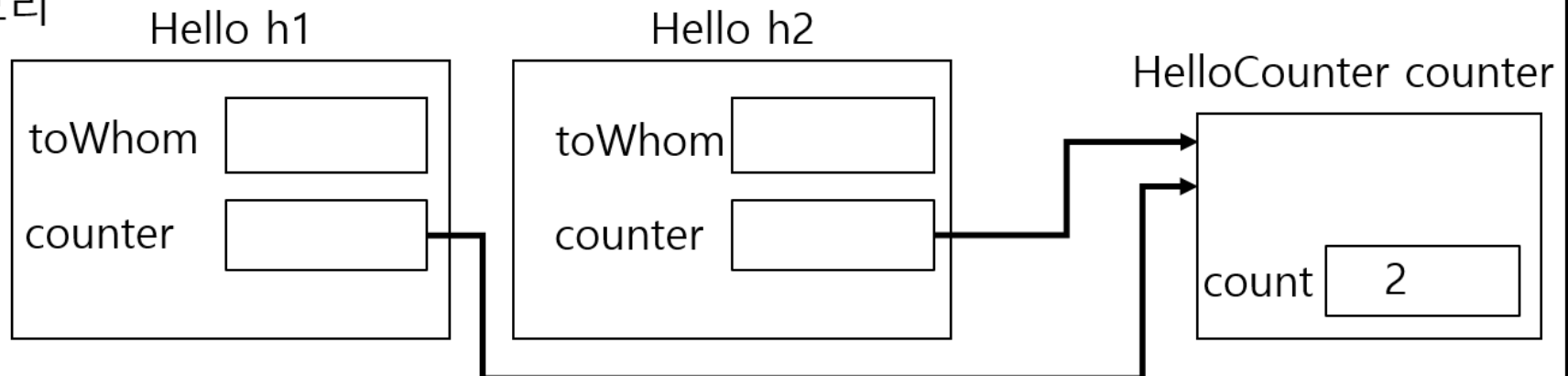
```
class Hello {  
    private String toWhom;  
    private HelloCounter counter;  
  
    Hello(String whom, HelloCounter counter) {  
        this.toWhom = whom;  
        this.counter = counter;  
        counter.increase();  
    }  
    void sayHello() {  
        System.out.println("hello " + toWhom);  
    }  
    int getCount() {  
        return counter.getCount();  
    }  
}
```

```

class TestHelloCounter {
    public static void main(String[] args) {
        HelloCounter counter = new HelloCounter();
        Hello h1 = new Hello("world", counter);
        System.out.println("counter: " + h1.getCount());
        Hello h2 = new Hello("new world", counter);
        System.out.println("counter1: " + h1.getCount());
        System.out.println("counter2: " + h2.getCount());
        h1.sayHello();
        h2.sayHello();
    }
}

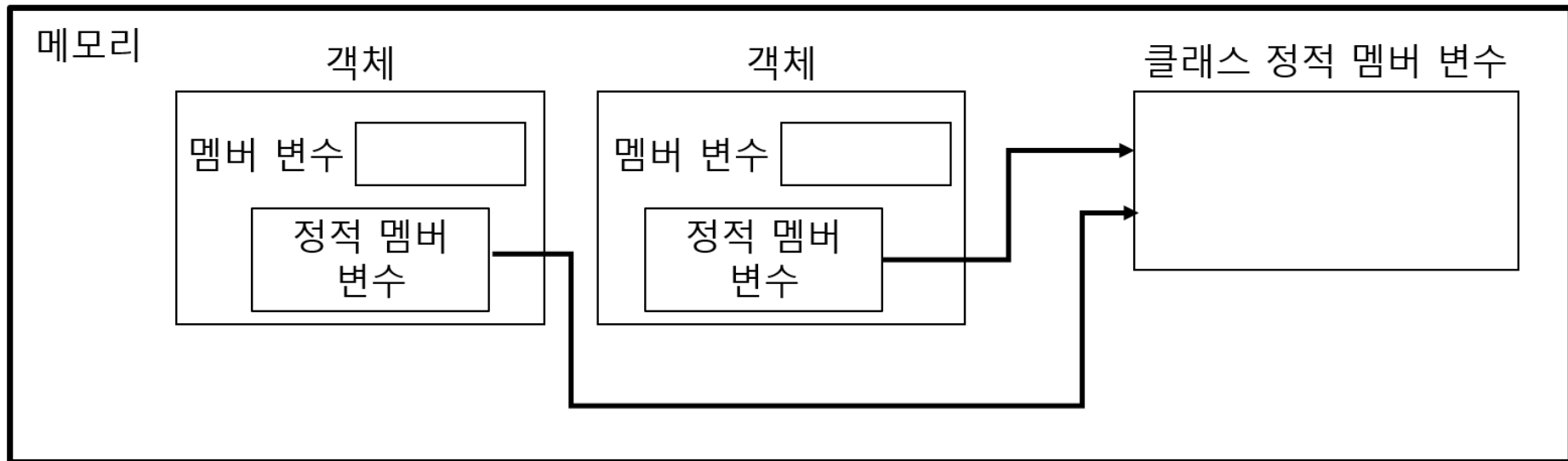
```

메모리



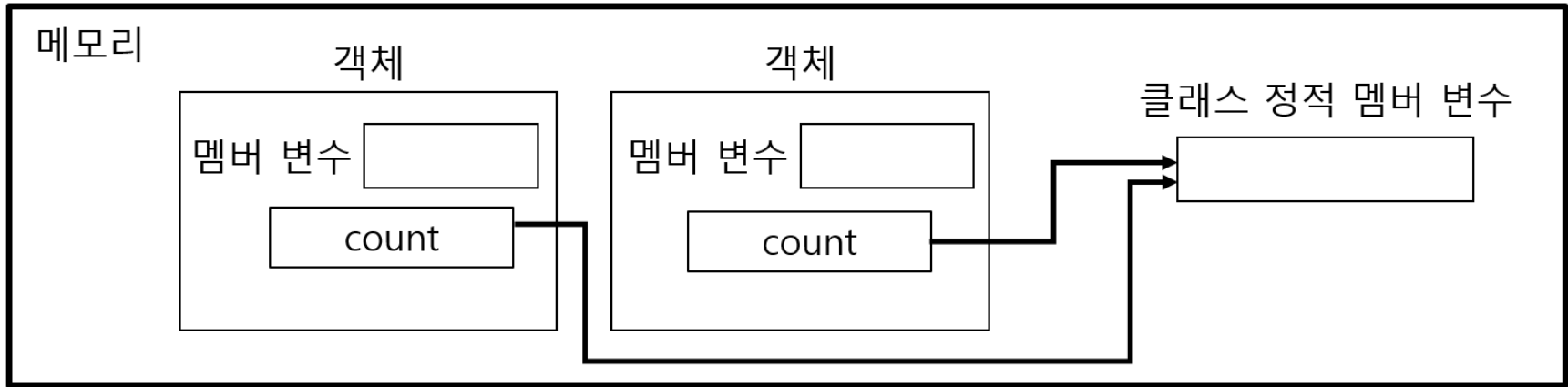
정적 멤버 변수(static member variable)

- 클래스에 선언되는 정적 멤버 변수는 해당 클래스로부터 생성되는 모든 객체가 공유하는 변수
 - 한 객체에서 정적 멤버 변수 값을 변경하면, 다른 객체들도 변경된 값을 사용하게 됨
 - 객체를 생성하지 않아도 사용할 수 있음

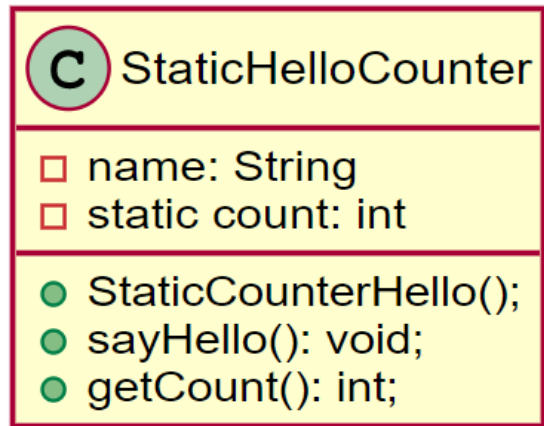


정적 멤버 변수(static member variable)

- 정적 멤버 변수를 이용해서 다시 구현
 - HelloCounter 대신 정적 멤버 변수를 사용
 - 생성자에서 변수값을 조작해서 생성 횟수를 기록



□ 클래스 다이어그램



```
// TestStaticHelloCounter.java
```

```
class StaticHelloCounter {  
    private String name;  
    private static int count = 0;  
  
    StaticHelloCounter(String name) {  
        this.name = name;  
        count++;  
    }  
    void sayHello() {  
        System.out.println("hello " + name);  
    }  
    int getCount() { return count; }  
}
```

```
class TestStaticHelloCounter {  
    public static void main(String[] args) {  
        StaticHelloCounter h1 =  
            new StaticHelloCounter("world");  
        System.out.println("counter: " + h1.getCount());  
    }  
}
```


정적 멤버 변수(static member variable)

```
StaticHelloCounter h2 =  
    new StaticHelloCounter("new world");  
System.out.println("count 1: " + h1.getCount());  
System.out.println("count 2: " + h2.getCount());  
h1.sayHello();  
h2.sayHello();  
}  
}
```

- 클래스 내부에서 정적 멤버 변수는 일반 멤버 변수와 사용법이 동일하지만, 모든 객체에 공유됨에 주의

정적 멤버 변수와 접근 제어자

- 정적 멤버 변수도 접근 제어자로 접근성 지정 가능
- 접근 제어자로 외부에서 정적 멤버 변수를 사용할 수 있도록 지정
 - 클래스_이름.정적_멤버_변수_이름
 - 객체_이름.정적_멤버_변수_이름

```
// TestPublicStaticHelloCounter.java
class PublicHelloCounter {
    private String name;
    public static int count = 0;

    PublicHelloCounter(String name) {
        this.name = name;
        PublicHelloCounter.count++;
        /* this.count++도 사용 가능 */
    }
}
```

정적 멤버 변수와 접근 제어자

```
void sayHello() {  
    System.out.println("hello " + name);  
}  
}  
  
class TestPublicStaticHelloCounter {  
    public static void main(String[] args) {  
        PublicHelloCounter h1 =  
            new PublicHelloCounter("world");  
        System.out.println("count: " + h1.count);  
        PublicHelloCounter h2 =  
            new PublicHelloCounter("new world");  
        System.out.println("count 1: " +  
            PublicHelloCounter.count);  
        System.out.println("count 2: " + h2.count);  
    }  
}
```

정적 멤버 변수와 일반 멤버 변수

특성	정적 멤버 변수	일반 멤버 변수
메모리 공간	모든 객체가 공유할 수 있는 독립적인 공간	객체 별로 할당된 메모리 공간
생성 시기	프로그램이 시작될 때 생성됨	객체가 생성될 때 만들어짐
접근 가능	<ul style="list-style-type: none">- 객체 생성과 상관없이 접근 및 사용 가능- 객체가 생성되지 않아도 변수는 사용 가능- 함수가 종료되면서 객체 변수가 사라져도 접근 가능	<ul style="list-style-type: none">- 객체 생성 후 접근 및 사용 가능- 객체가 생성되지 않으면 변수 사용 불가- 함수가 종료되면서 객체 변수가 사라지면 해당 객체의 멤버 변수 접근 불가

정적 멤버 변수와 일반 멤버 변수

특성	정적 멤버 변수	일반 멤버 변수
외부 접근	클래스_이름.변수_이름, 객체_이름.변수_이름	객체_이름.변수_이름
내부 접근	클래스_이름.변수_이름, this.변수_이름, 변수_이름	this.변수_이름, 변수_이름

정적 메소드와 정적 멤버 변수

- 정적 메소드는 정적 멤버 변수만 접근 가능
- 앞에서 만든 count 변수값을 확인하려면 객체를 생성해야만 함
- 해결 방법
 - count 변수를 public으로 지정
 - getCount() 함수를 static으로 지정

```
class PrivateCounterHello {  
    private String name;  
    private static int count = 0;  
  
    PrivateCounterHello(String name) {  
        this.name = name;  
        count++;  
    }  
}
```

```
void sayHello() {  
    System.out.println("hello " + name);  
}  
public static int getCount() {  
    return count;  
}  
}
```

```
class TestPrivateStaticHelloCounter {  
    public static void main(String[] args) {  
        System.out.println("count: " +  
                            PrivateCounterHello.getCount());  
        PrivateCounterHello h1 = new  
            PrivateCounterHello("world");  
        System.out.println("count: " +  
                            PrivateCounterHello.getCount());  
        PrivateCounterHello h2 = new  
            PrivateCounterHello("new world");  
        System.out.println("count 1: " + h1.getCount());  
        System.out.println("count 2: " + h2.getCount());  
    }  
}
```

다시 보는 main() 함수와 명령행 인자

- 자바에서 컴파일된 클래스를 실행시키고 싶으면 main() 함수가 있어야 하고, 자바 프로그램은 main() 함수 내에 있는 코드를 실행시키는 것
- main() 함수의 헤더

```
public static void main(String[] args)
```
- Main 클래스 실행

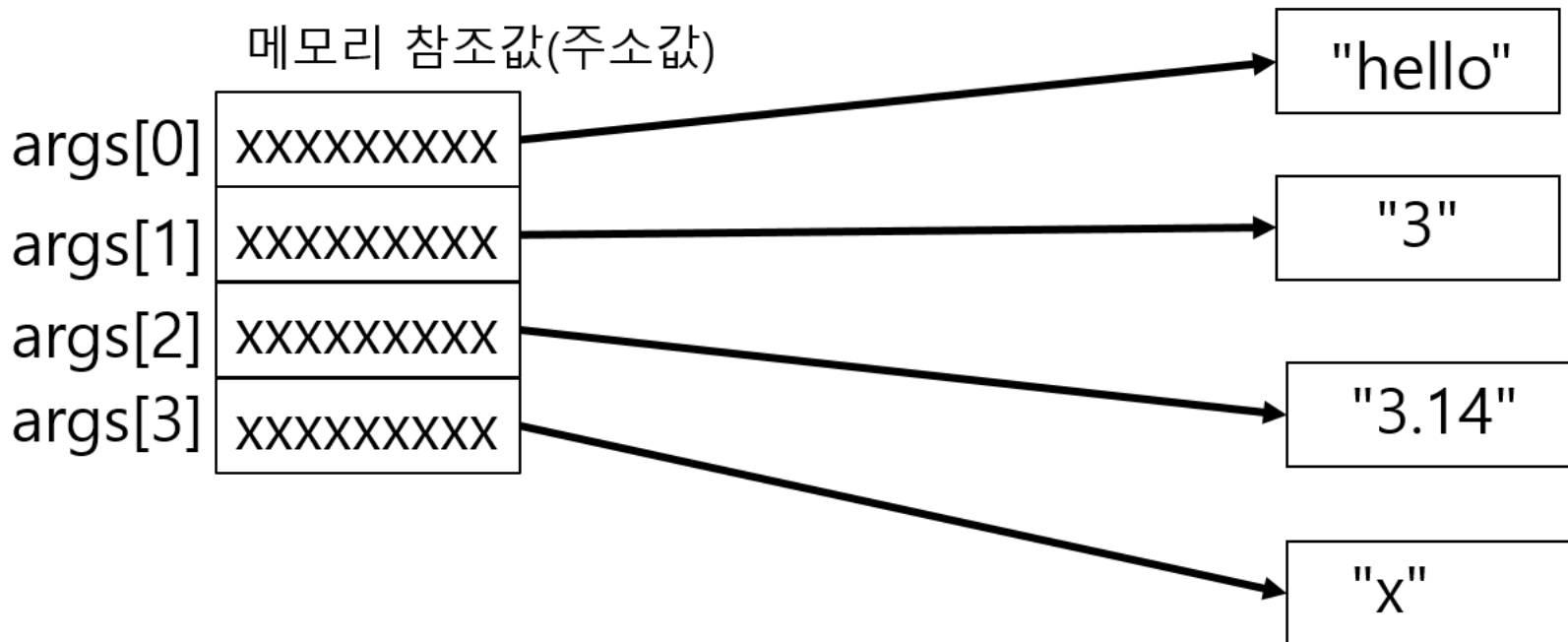
```
java Main
```
- 자바 시스템에서 호출

```
Main.main(args);
```


다시 보는 main() 함수와 명령행 인자

- main() 함수에 전달되는 문자열 배열이 명령행 인자 (command line arguments)
- 예

```
java Main "hello" 3 3.14 x
```



다시 보는 main() 함수와 명령행 인자

- 명령행 인자에 따옴표를 넣으려면 이스케이프 시퀀스 사용
- 인자에 공백 문자가 들어가면 따옴표를 사용
- 인자를 출력하는 프로그램

```
// CmdLineArgs.java
public class CmdLineArgs {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.printf("args[%d] = %s\n", i, args[i]);
        }
    }
}
```

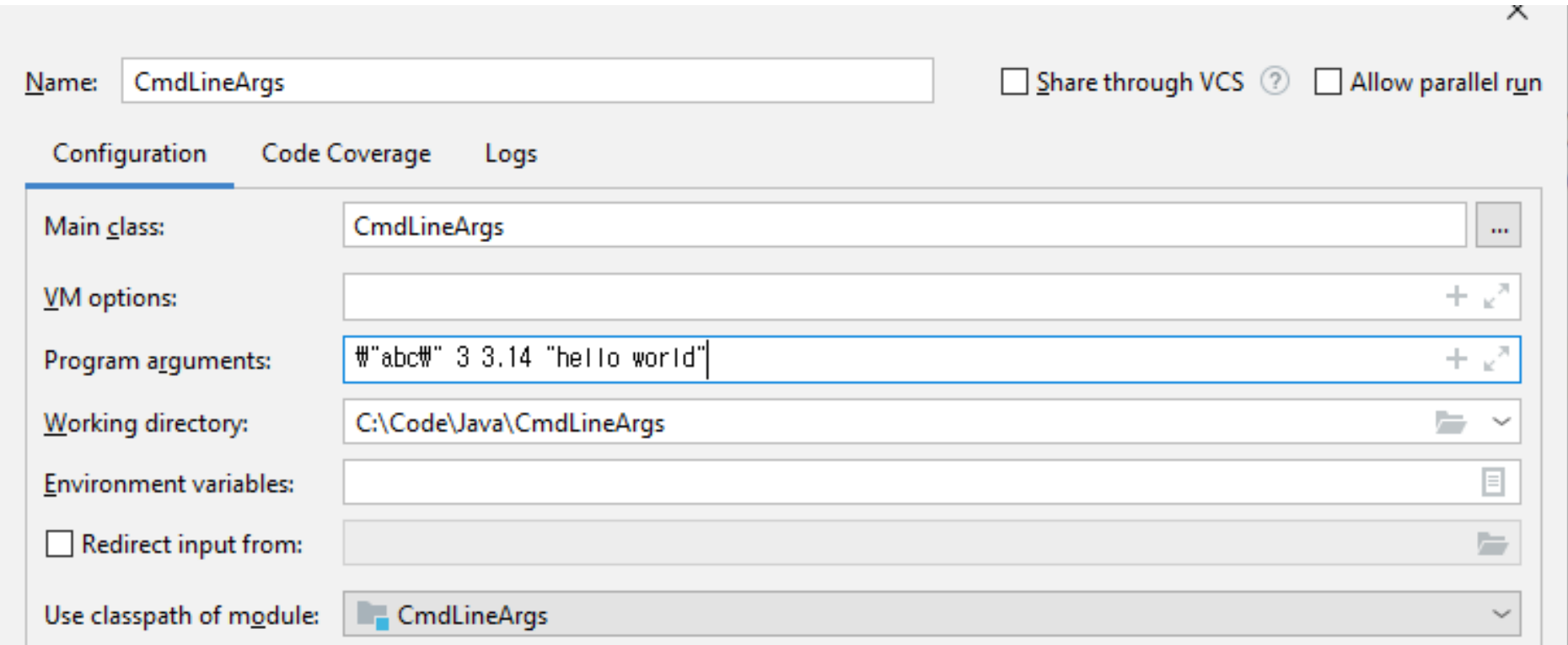
다시 보는 main() 함수와 명령행 인자

▣ 실행 결과

```
C:\Code\java\07>java CmdLineArgs \"abc\" 3 3.14 \"hello world\"  
args[0] = \"abc\"  
args[1] = 3  
args[2] = 3.14  
args[3] = hello world
```

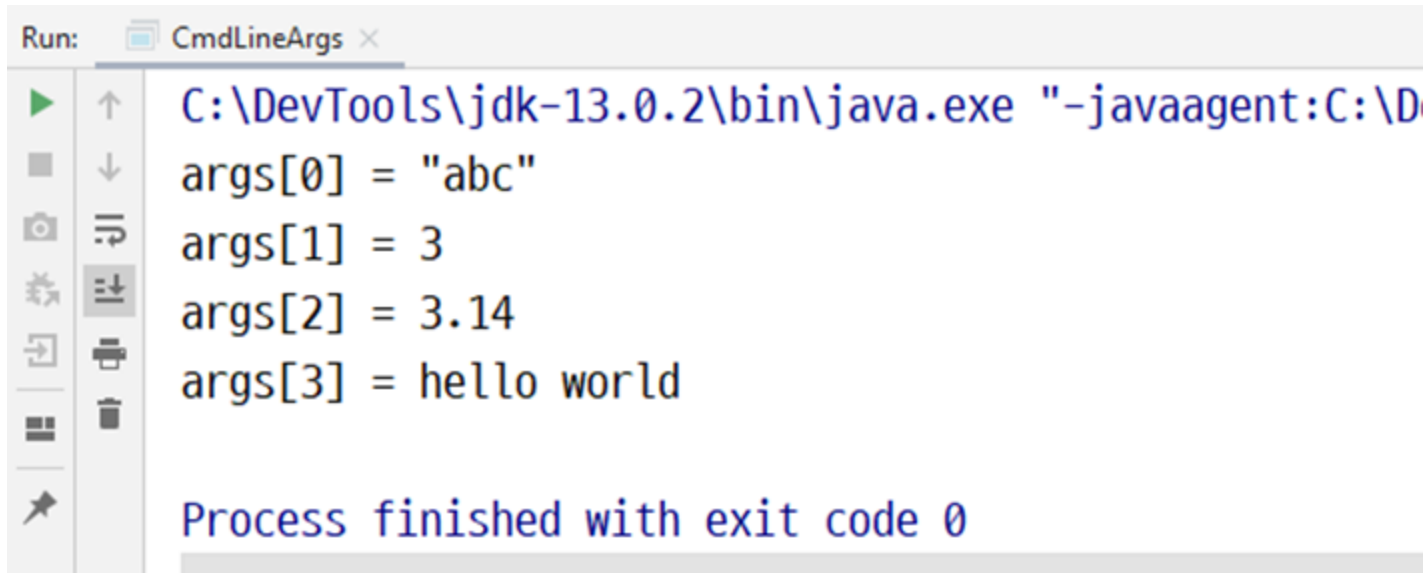
다시 보는 main() 함수와 명령행 인자

- 인텔리제이 IDEA에서 명령행 인자 지정
 - 메뉴에서 Run|Edit configurations...를 실행 한 후 "Program arguments" 부분에 내용 입력



다시 보는 main() 함수와 명령행 인자

□ 실행 화면



The screenshot shows a Java IDE's Run console window. The title bar reads 'Run: CmdLineArgs x'. On the left is a vertical toolbar with icons for running, stepping through, and other debugging actions. The main area displays the command executed: `C:\DevTools\jdk-13.0.2\bin\java.exe "-javaagent:C:\D`. Below the command, the output shows the contents of the `args` array: `args[0] = "abc"`, `args[1] = 3`, `args[2] = 3.14`, and `args[3] = hello world`. At the bottom, it states 'Process finished with exit code 0'.

```
Run: CmdLineArgs x  
C:\DevTools\jdk-13.0.2\bin\java.exe "-javaagent:C:\D  
args[0] = "abc"  
args[1] = 3  
args[2] = 3.14  
args[3] = hello world  
  
Process finished with exit code 0
```

final 멤버 변수와 상수

- final 멤버 변수는 값이 한 번 정해지면 더 이상 바꿀 수 없는 변수

```
final 자료형 변수_이름 [= 초기값]; // [] 부분 생략 가능
```

- 변수가 선언될 때 초기화 되지 않으면 생성자에서 초기화
- 생성자가 여러 개 있으면 모든 생성자에서 초기화

```
class FinalMemberVariable {  
    final int a = 3;  
    final int b;  
    private final int c;  
    final private int d;  
}
```

```
FinalMemberVariable() {  
    b = 5;  
    c = 3;  
    d = 4;  
}
```

```
FinalMemberVariable(int b) {  
    this.b = b;  
    c = 3;  
    // d = 4; // d를 초기화 시키지 않으면 오류 발생  
}
```

```
void setD(int d) {  
    //this.d = d; // 일반 메소드에서 final 변수 변경 불가  
}
```

```
void print() {  
    System.out.printf("a = %d, b = %d, c = %d, d =  
%d\n", a, b, c, d);  
}  
}
```

final 멤버 변수와 상수

- 상수 - 변수가 처음 선언될 때 초기값이 주어지고 프로그램의 실행 중에 그 값을 변경할 수 없는 경우
- final 변수와 비슷하지만 조금 다름
 - final 변수는 생성자에서 값을 변경할 수 있음
- 상수는 프로그램을 만들면서 프로그램 실행 중에 전대 변경해서는 안 되는 값을 지정
 - 수학에서 원의 넓이 등을 계산할 때 쓰는 PI(3.14159...)
 - 일주일을 표현하는 7
 - 일년의 일 수를 표현하는 365
- 상수를 final 멤버 변수로 만드는 것은 비효율적
 - static을 붙여 모든 객체에 공유시킴


```
class Circle {
    static final double PI = 3.14159; // 초기화
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }
    double calcArea() {
        return radius * radius * PI;
    }
    double calcPerimeter() {
        return 2 * radius * PI;
    }
}

class CircleMain {
    public static void main(String[] args) {
        Circle c = new Circle(5);
        System.out.printf("Area = %f\n", c.calcArea());
        System.out.printf("Perimeter = %f\n",
                           c.calcPerimeter());
    }
}
```

final 멤버 변수와 상수

- 접근 제어자(private, protected, public), final, static 등의 키워드 순서는 상관없음
- 자주 사용되는 방법은 PSF 형태
- P: private, protected, public
- S: static, abstract
- F: final

클래스 연관

- 결합성이란 클래스 사이의 상호 관계 정도
- 클래스를 일관성 있게 만들면, 관련되어 있지 않는 "개념"들을 분리해서 새로운 클래스에 구성
 - 클래스간 연관 관계가 생성될 수 있음
- 클래스 또는 객체 사이의 관계
 - 연관(association)
 - 두 클래스 사이가 연결됨
 - 클래스 간에 참조가 사용
 - UML에서 실선과 화살표를 이용
 - 의존(dependency)
 - 연관과 비슷하지만 참조를 사용하는 시기가 짧음
 - UML에서는 점선 또는 점선으로 구성된 화살표

클래스 연관

□ 집합(aggregation)

- 집합과 구성은 전체와 부분을 나타내는 관계
- 두 개는 비슷하지만 전체와 부분의 생명 주기가 일치하는 지로 구분
- 집합은 전체와 부분의 생명 주기가 다름
- UML에서는 실선으로 표시하되 전체를 나타내는 쪽에 다이아몬드 모양이 붙음
 - 마름모 모양은 비어있는 형태

□ 구성(composition)

- 전체와 부분의 생명 주기가 일치
- UML에서는 실선과 마름모 모양으로 표시
- 다이아몬드 모양이 채워져 있음

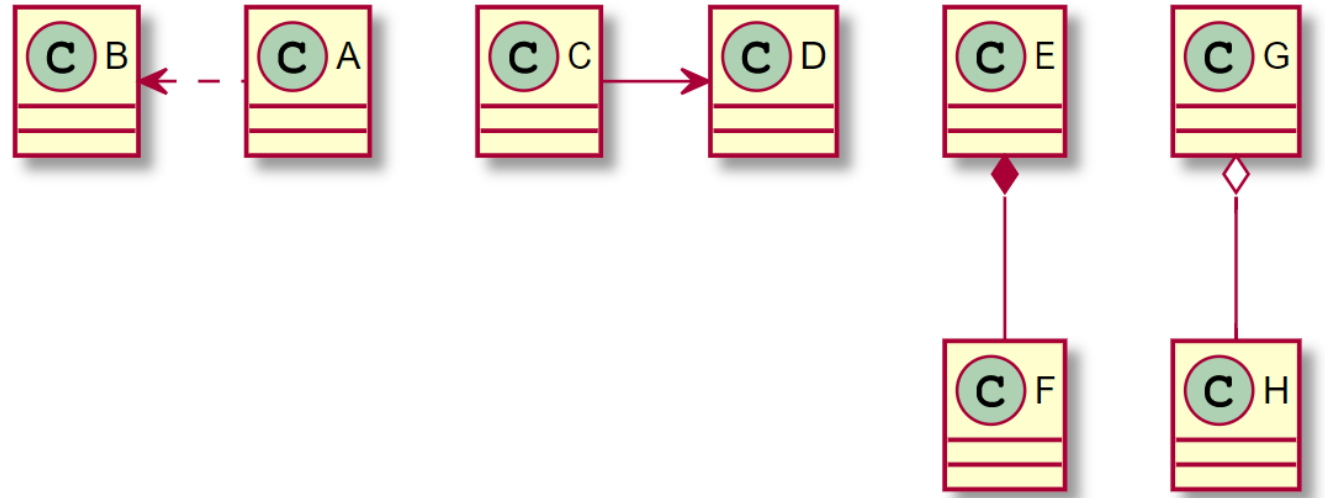
클래스 연관

관계	Plant UML 스크립트 작성 방법
연관	-- (실선, 양방향), --> (화살표, 단방향)
의존	.. (점선, 양방향), ..> (점선 화살표, 단방향)
집합	o--
구성	*--

- 클래스 다이어그램에서 클래스의 상대적인 위치를 오른쪽, 왼쪽, 위쪽, 아래쪽에 둘 수 있도록 right, left, up, down으로 지정 가능
- 다이아몬드 모양이 채워져 있음

클래스 연관

```
@startuml
class A
class B
class C
class D
class E
class F
class G
class H
C -right-> D
A .left-> B
E *-- F
G o-- H
@enduml
```



실습 문제 1: PC를 만들어본다

□ 문제

- 개인용 컴퓨터인 PC(personal computer)는 일반적으로 본체와 모니터로 구성되지만 애플의 아이맥처럼 본체와 모니터가 붙어 있는 일체형 PC도 있음
- 이러한 경우를 고려해서 클래스를 만들어보기

□ 요구사항

- PC는 본체, 모니터 외에 키보드 마우스도 있음
 - 여기서는 키보드/마우스는 제외
- 컴퓨터 - 전원 공급 장치(power supply), CPU, 메모리, 하드 디스크를 속성으로 포함
- 모니터- 전원 공급 장치, 크기, 컬러 등을 속성으로 포함
- 속성의 자세한 자료형은 지정하지 않음
- UML로 클래스간의 상관 관계에 대해서 다이어그램 작성

실습 문제 2: 수강 신청하는 프로그램 작성

□ 문제

- 이번 학기에 개설되는 실습 과목인 객체 지향 강좌를 수강 신청하는 프로그램을 작성
- 최대 인원은 30명
- 강사는 김갑돌
- 강의실은 G416

□ 요구사항

- 문제의 목적은 클래스 사이의 연관 관계를 살펴보는 것
- 따라서 각 클래스의 모든 기능을 사용하는 코드를 작성하지는 않음
- 객체 지향 강좌를 수강하는 프로그램을 작성하지만, 클래스들은 가능하면 일반화시킬 것

실습 문제 2: 수강 신청하는 프로그램 작성

□ 해결

■ 클래스 찾기(명사 찾기)

- 학기, 개설, 강좌, 실습 과목, 강의실, 강사, 인원

- 우선 강좌를 클래스로 만들기로 함

- 클래스 이름은 ObjectOrientedCourse

- 학생 인원은 30명으로 정함 (배열로 만들 것)

- 등록하는 학생에게 현재까지의 등록 인원을 공유할 수 있어야 함

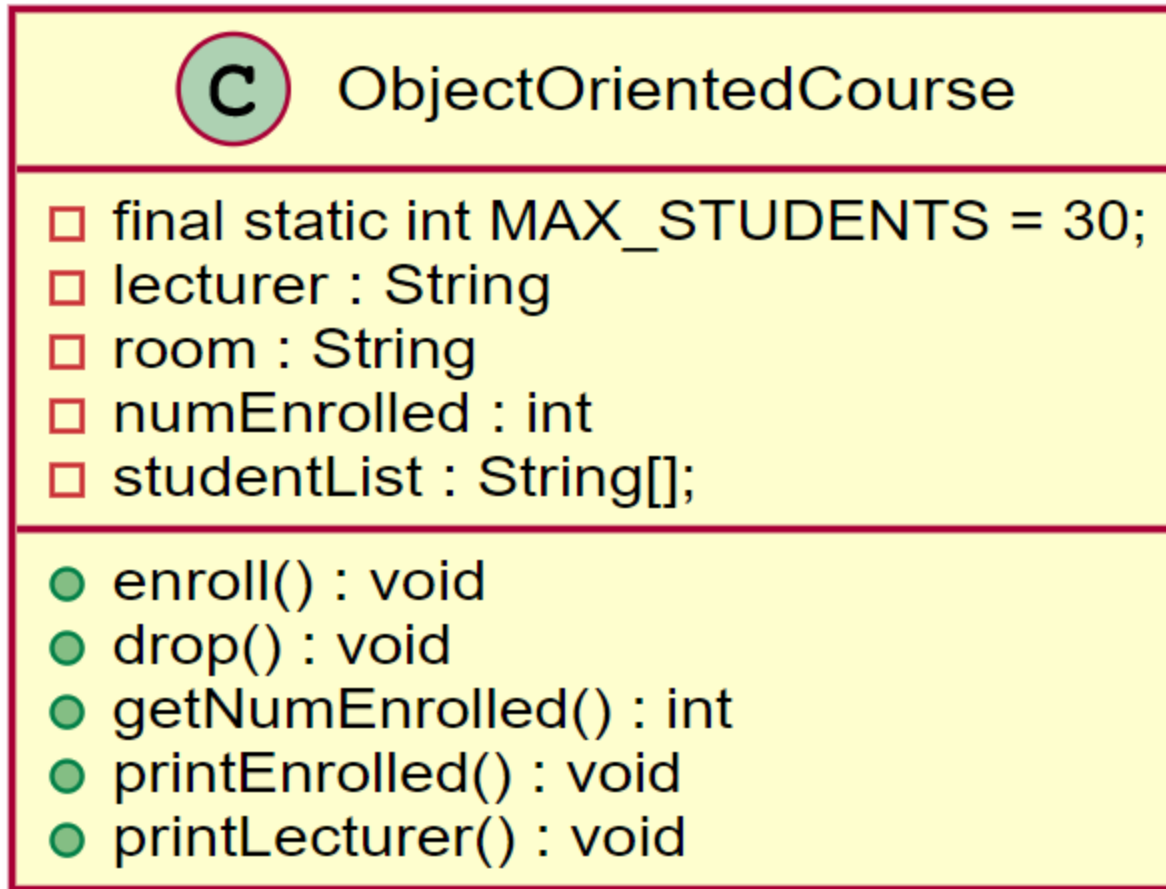
실습 문제 2: 수강 신청하는 프로그램 작성

□ 버전 1: 강좌 클래스 구현

- 학생 수는 30명 - 상수로 지정
- 등록할 때마다 학생 수를 늘리고 취소하면 학생 수를 줄여야 함 → 변수로 만들어서 관리
- 학생에게 공유할 수 있도록 메소드 제공
- 강의실은 강좌가 만들어질 때 정해짐
 - 추후에 바뀔 수 있으므로 다시 지정할 수 있는 메소드 제공

실습 문제 2: 수강 신청하는 프로그램 작성

□ 클래스 다이어그램



```
class ObjectOrientedCourse {
    private final static int MAX_STUDENTS = 30;
    private int numEnrolled = 0;
    private String lecturer;
    private String room;
    private String[] students = new String[MAX_STUDENTS];

    public ObjectOrientedCourse(String lecturer,
                                String room) {
        this.lecturer = lecturer;
        this.room = room;
    }
    public setRoom(String room) { this.room = room; }
    public void enroll(String s) {
        if (numEnrolled < MAX_STUDENTS) {
            students[numEnrolled] = s;
            numEnrolled++;
        }
    }
}
```

```
public void drop(String s) {  
    if (numEnrolled > 0) {  
        for (int i = 0; i < students.length; i++) {  
            // 검색해서 s를 찾으면 뒤에 있는 요소들을  
            // 움직여서 앞으로 채움  
            if (students[i] == s) {  
                for (int j = i + 1; j < students.length; j++) {  
                    students[j - 1] = students[j];  
                }  
                numEnrolled--;  
                break;  
            }  
        }  
    }  
}  
  
public int getNumEnrolled() { return numEnrolled; }  
public void printEnrolled() {  
    for (int i = 0; i < numEnrolled; i++) {  
        System.out.println(students[i]);  
    }  
}  
}
```

실습 문제 2: 수강 신청하는 프로그램 작성

□ 문제점

- 등록하고 취소할 때 학생 이름을 사용
- 강사도 이름만 저장됨
- 클래스 이름도 너무 특화되어 있음

□ 버전 2: "학생" 클래스 추가 및 이름 변경

- 클래스 이름을 일반화된 Course로 변경
- String으로 되어 있던 학생을 Student로 변경
- 새로 작성된 코드

```
class Course {  
    private final static int MAX_STUDENTS = 30;  
    private int numEnrolled = 0;  
    private String lecturer;  
    private String room;  
    private Student[] students = new Student[MAX_STUDENTS];  
}
```

```
Course(String lecturer, String room) {  
    this.lecturer = lecturer;  
    this.room = room;  
}  
public setRoom(String room) { this.room = room; }  
public void enroll(Student s) {  
    if (numEnrolled < MAX_STUDENTS) {  
        students[numEnrolled] = s;  
        numEnrolled++;  
    }  
}  
public int getNumEnrolled() {  
    return numEnrolled;  
}  
public void printEnrolled() {  
    for (int i = 0; i < numEnrolled; i++) {  
        System.out.println(students[i]);  
    }  
}
```

실습 문제 2: 수강 신청하는 프로그램 작성

```
public void drop(Student s) {  
    if (numEnrolled > 0) {  
        for (int i = 0; i < students.length; i++) {  
            if (students[i] == s) {  
                for (int j = i + 1; j < students.length; j++) {  
                    students[j - 1] = students[j];  
                }  
                numEnrolled--;  
                break;  
            }  
        }  
    }  
}
```

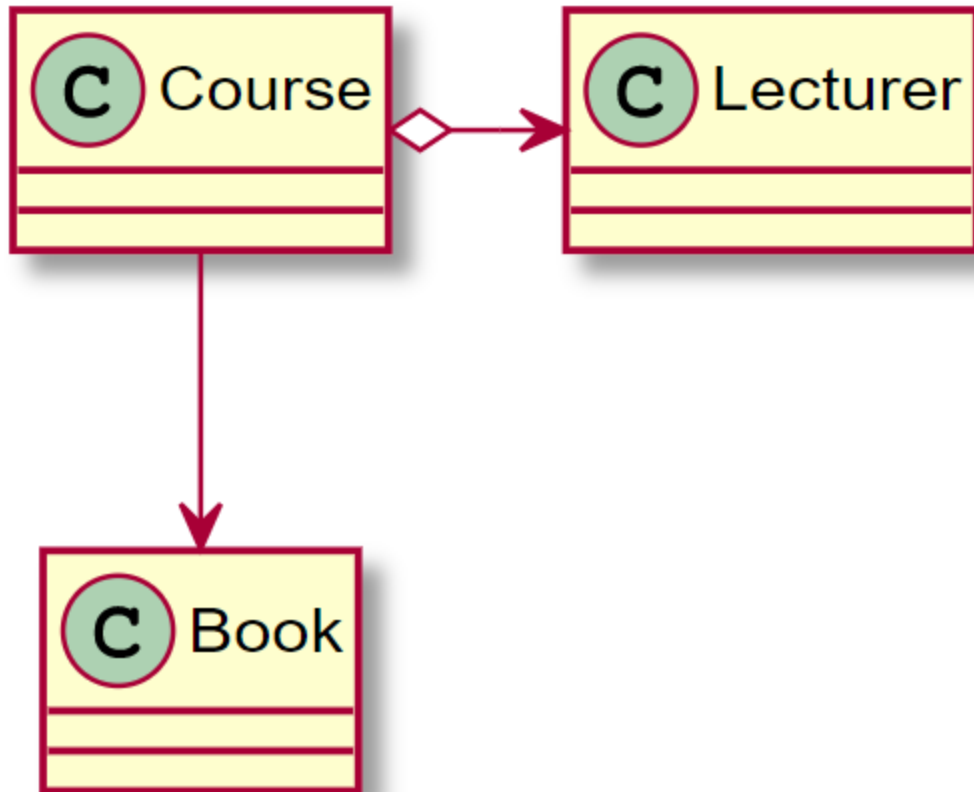

실습 문제 2: 수강 신청하는 프로그램 작성

▣ Student 클래스

```
// Student.java
class Student {
    private String stNo;
    private String name;
    public Student(String stNo, String name) {
        this.stNo = stNo;
        this.name = name;
    }
    public String getStNo() { return stNo; }
    public String getName() { return name; }
    public String toString() {
        return "Name: " + name + ", Student Number: " + stNo;
    }
}
```

실습 문제 2: 수강 신청하는 프로그램 작성

- ❑ 버전 3: "강사"와 "교재" 클래스 추가
- ❑ 클래스 다이어그램



실습 문제 2: 수강 신청하는 프로그램 작성

▣ 새로 수정된 Course 클래스

```
// Course.java
class Course {
    private final int MAX_STUDENTS = 30;
    private int numEnrolled = 0;
    private Lecturer lecturer;
    private String room;
    private Book ref;
    private Student[] students = new Student[MAX_STUDENTS];

    Course(Lecturer lecturer, String room) {
        this.lecturer = lecturer;
        this.room = room;
    }
    public void setRoom(String room) { this.room = room; }
    public void setBook(Book r) { ref = r; }
```

```
public void enroll(Student s) {
    if (numEnrolled < MAX_STUDENTS) {
        students[numEnrolled] = s;
        numEnrolled++;
    }
}

public void drop(Student s) {
    if (numEnrolled > 0) {
        for (int i = 0; i < students.length; i++) {
            if (students[i] == s) {
                for (int j = i + 1; j < students.length; j++) {
                    students[j - 1] = students[j];
                }
                numEnrolled--;
                break;
            }
        }
    }
}
```

```
public int getNumEnrolled() { return numEnrolled; }
public void printEnrolled() {
    for (int i = 0; i < numEnrolled; i++) {
        System.out.println(students[i]);
    }
}
```

▣ 강사 클래스

```
// Lecturer.java
class Lecturer {
    private String email;
    private String name;
    private String room;
    public Lecturer(String name, String room, String
email) {
        this.room = room;
        this.name = name;
        this.email = email;
    }
}
```

```
public String getEmail() {  
    return email;  
}  
public String getName() {  
    return name;  
}  
public String getRoom() {  
    return room;  
}  
}
```

▣ 교재 클래스

```
// Book.java  
class Book {  
    String name;  
    String author;  
  
    public Book(String name, String author) {  
        this.name = name;  
        this.author = author;  
    }  
}
```

실습 문제 2: 수강 신청하는 프로그램 작성

```
String getName() {  
    return name;  
}  
String getAuthor() {  
    return author;  
}  
}
```

실습 문제 2: 수강 신청하는 프로그램 작성

▣ 최종 코드

```
// TestCourse.java
class TestCourse {
    public static void main(String[] args) {
        Lecturer lecturer = new Lecturer("김복동", "G412",
            "bkim3234829@su.ac.kr");

        Course oo1 = new Course(lecturer, "G416");

        Student s1 = new Student("202011111", "js1");
        Student s2 = new Student("202011112", "bdk");
        Student s3 = new Student("202011112", "cho");
        oo1.enroll(s1);
        System.out.printf("Num of enrolled Students: %d\n",
            oo1.getNumEnrolled());
    }
}
```


실습 문제 2: 수강 신청하는 프로그램 작성

```
        oo1.enroll(s2);
        System.out.printf("Num of enrolled Students: %d\n",
oo1.getNumEnrolled());
        oo1.enroll(s3);
        System.out.printf("Num of enrolled Students: %d\n",
oo1.getNumEnrolled());
        oo1.drop(s2);
        System.out.printf("Num of enrolled Students: %d\n",
oo1.getNumEnrolled());
        oo1.printEnrolled();
    }
}
```

실습 문제 2: 수강 신청하는 프로그램 작성

□ 실행 결과

```
C:\Code\java\07>java TestCourse  
Num of enrolled Students: 1  
Num of enrolled Students: 2  
Num of enrolled Students: 3  
Num of enrolled Students: 2  
Name: js1, Student Number: 202011111  
Name: cho, Student Number: 202011112
```

실습 문제 3: 약속 – 약속 클래스에서 약속 장소 및 사람 분리

□ 문제

- 누군가와 약속을 정하고 특정 장소에서 만나는 것을 약속 (Appointment)이라는 클래스로 만들것
- 어떤 정보들을 멤버 변수로 넣을지 정리하고 클래스 구현

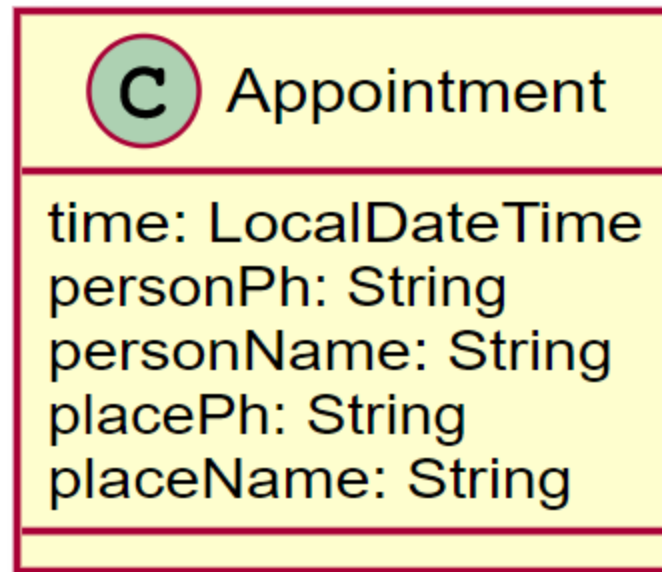
□ 요구사항

- 약속 클래스를 1개 만들고 화면에 출력
- 구현하는 모든 클래스에 toString() 함수를 구현하고 이를 이용해서 화면에 출력

실습 문제 3: 약속 – 약속 클래스에서 약속 장소 및 사람 분리

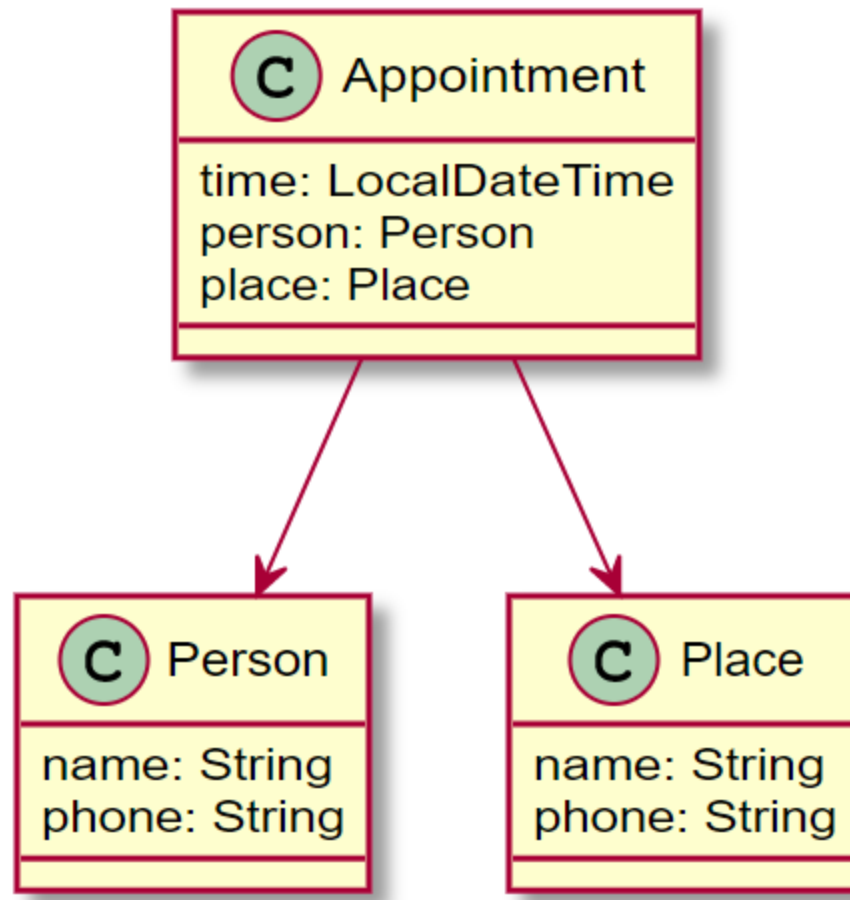
□ 해결

- 약속과 관련된 정보를 정리
 - 시간, 장소, 장소 연락처, 만나는 사람의 이름, 만나는 사람의 연락처
- 클래스 다이어그램



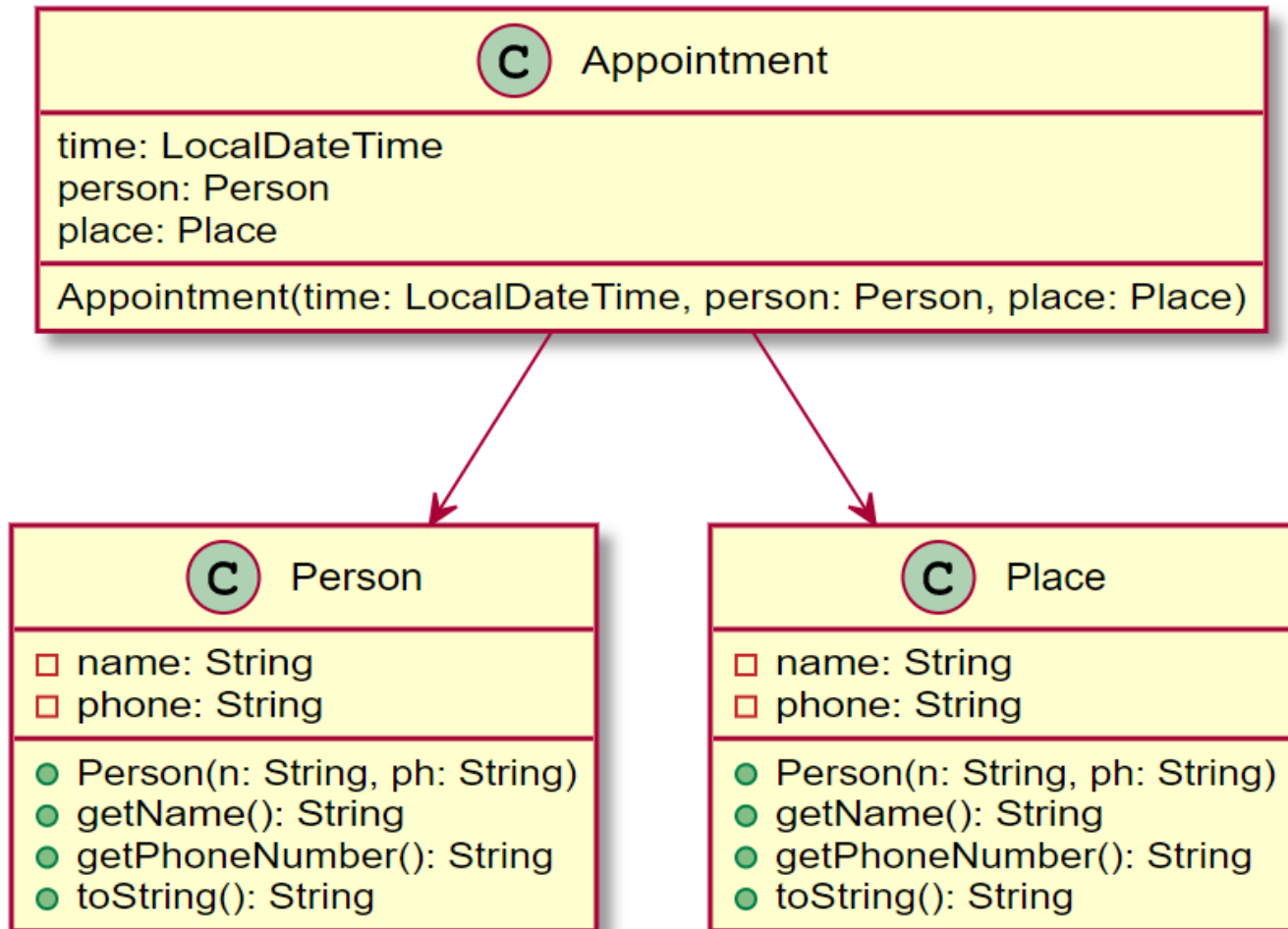
실습 문제 3: 약속 - 약속 클래스에서 약속 장소 및 사람 분리

□ 새로 수정된 클래스 다이어그램



실습 문제 3: 약속 - 약속 클래스에서 약속 장소 및 사람 분리

□ 새로 수정된 클래스 다이어그램 (메소드 포함)



실습 문제 3: 약속 - 약속 클래스에서 약속 장소 및 사람 분리

```
// Person.java
class Person {
    private String name;
    private String phone;
    public Person(String n, String ph)
        name = n;
        phone = ph;
    }
    public String getName() {
        return name;
    }
    public String getPhoneNumber() {
        return phone;
    }
    public String toString() {
        return "Name: " + name + " Phone: " + phone;
    }
}
```

실습 문제 3: 약속 - 약속 클래스에서 약속 장소 및 사람 분리

```
// Place.java
class Place {
    private String name;
    private String phone;
    public Place(String n, String ph) {
        name = n;
        phone = ph;
    }
    public String getName() {
        return name;
    }
    public String getPhoneNumber() {
        return phone;
    }
    public String toString() {
        return "Name: " + name + " Phone: " + phone;
    }
}
```


실습 문제 3: 약속 – 약속 클래스에서 약속 장소 및 사람 분리

- Appointment 클래스에서는 약속 시간을 위해 `java.time.LocalDateTime` 클래스 사용
 - 현재 시각 정보를 이용해서 객체를 생성하는 `now()` 함수
 - 주어진 시각 정보를 이용해서 객체를 생성하는 `of()` 함수
- 사용법

```
jshell> LocalDateTime dt = LocalDateTime.now();
```

```
jshell> System.out.println(dt);  
2020-05-19T16:12:47.609360400
```

```
jshell> LocalDateTime dt2  
      = LocalDateTime.of(2020, 5, 19, 16, 12, 47);
```

```
jshell> System.out.println(dt2);  
2020-05-19T16:12:47
```

```
// Appointment.java
import java.time.LocalDateTime;

class Appointment {
    private LocalDateTime time;
    private Person person;
    private Place place;
    Appointment(LocalDateTime time,
                Person person, Place place) {
        this.time = time;
        this.person = person;
        this.place = place;
    }
    public String toString() {
        return "DateTime: " + time.toString()
            + "\nPerson: " + person.toString()
            + "\nPlace: " + place.toString();
    }
}
```

실습 문제 3: 약속 - 약속 클래스에서 약속 장소 및 사람 분리

```
jshell> import java.time.LocalDateTime;

jshell> LocalDateTime d
           = LocalDateTime.of(2020, 5, 19, 16, 30, 0);

jshell> Person person
           = new Person("cho", "111-1111-1111");

jshell> Place place
           = new Place("jongno", "222-2222-2222");

jshell> Appointment apnt
           = new Appointment(d, person, place);

jshell> System.out.println(apnt);
DateTime: 2020-05-19T16:30
Person: Name: cho Phone: 111-1111-1111
Place: Name: jongno Phone: 222-2222-2222
```

실습 문제 3: 약속 - 약속 클래스에서 약속 장소 및 사람 분리

□ 최종 코드

```
// TestAppointment.java
import java.time.LocalDateTime;

public class TestAppointment {
    public static void main(String[] args) {
        Appointment appt = new Appointment(
            LocalDateTime.of(2020, 5, 19, 16, 30, 0),
            new Person("cho", "111-1111-1111"),
            new Place("jongno", "222-2222-2222"));
        System.out.println(appt);
    }
}
```

```
C:\Code\java\07>java TestAppointment
```

```
DateTime: 2020-05-19T16:30
```

```
Person: Name: cho Phone: 111-1111-1111
```

```
Place: Name: jongno Phone: 222-2222-2222
```

실습 문제 4: 병원 약속

□ 문제

- 병원 약속(HospitalAppointment) 클래스를 만들 것
- 병원 약속 클래스는 의사와 환자, 시간 등을 지정해서 객체 생성

□ 요구사항

- 의사(Doctor) 클래스는 의사의 아이디와 이름을 지정해서 객체를 생성하고 이들 속성들을 확인할 수 있음
 - 의사의 아이디는 이미 정해져 있는 값을 사용
- 환자는 객체를 생성할 때 환자의 이름을 지정
 - 객체가 생성될 때 환자의 고유 아이디를 만들어서 부여하고 이를 확인할 수 있는 멤버 함수 제공
- 병원 약속 객체들을 배열에 추가하고 화면에 출력
- 모든 클래스에 toString() 함수 구현하고 이를 출력에 활용

실습 주문 5: 커피 주문

□ 문제

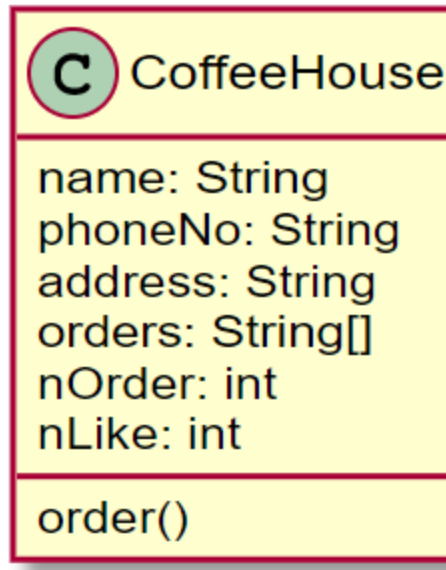
- 커피를 주문하는 프로그램을 작성. 좋은 커피 집이 어디 있는지 업소명, 위치, 전화번호를 찾음. "좋아요"가 많은 곳을 선택

□ 요구사항

- 같은 곳에서 10회 이상 주문하면, 커피를 한 잔 무료로 받을 수 있음

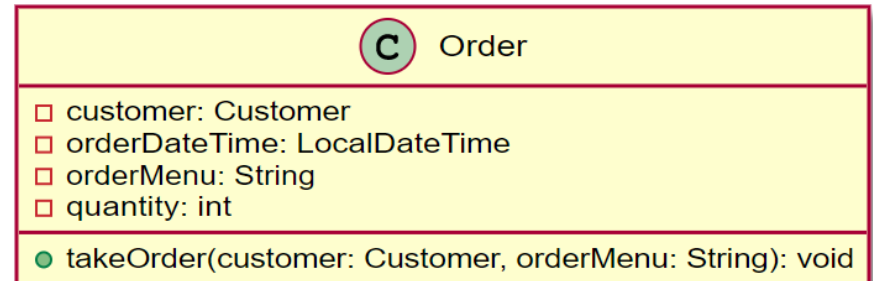
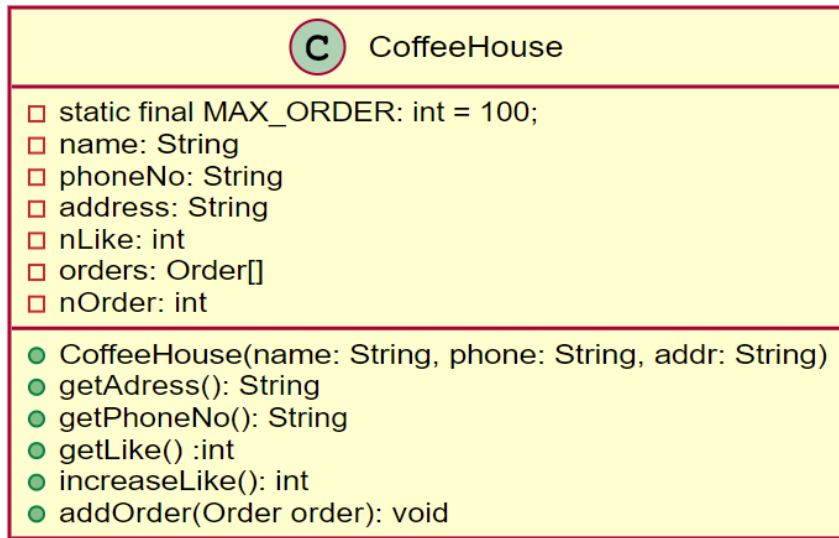
실습 주문 5: 커피 주문

□ 버전 1

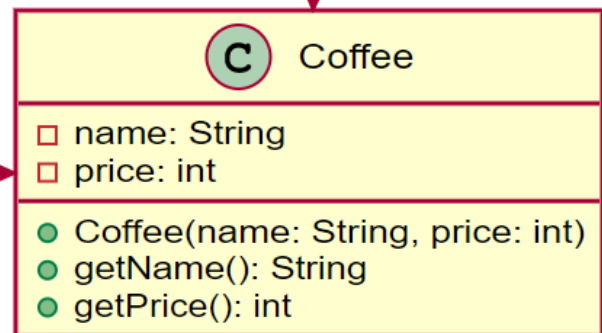
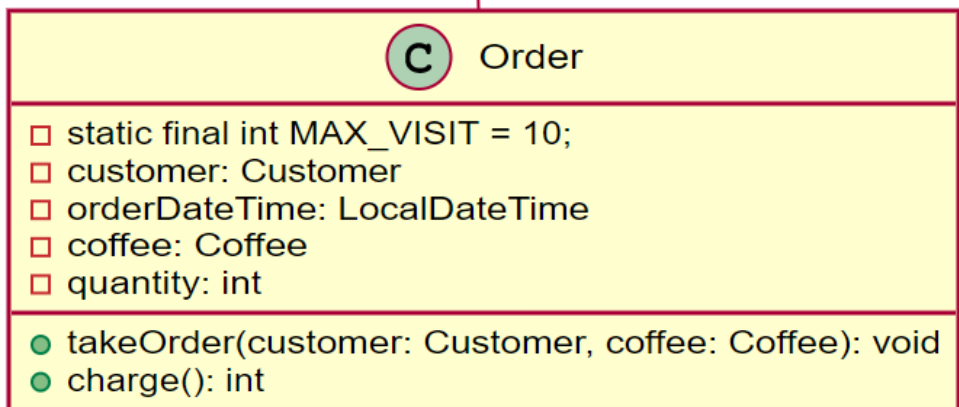
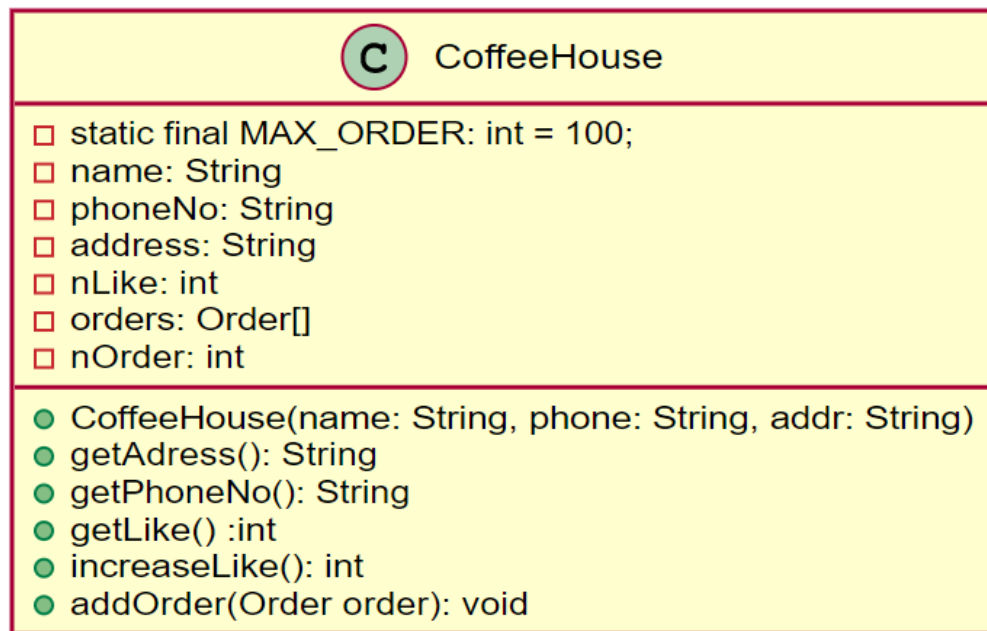
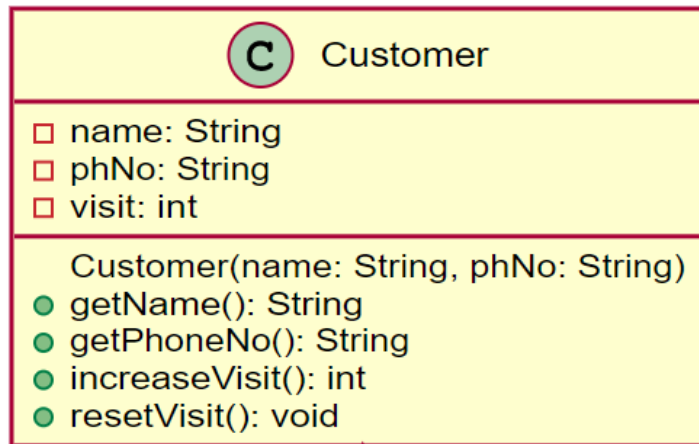


실습 주문 5: 커피 주문

□ 버전 2



버전 3



패키지(Package)

□ 패키지

- 관련된 클래스들을 묶어서 관리
- 소스 코드와 클래스 파일들을 폴더로 나누어서 관리
- 클래스 이름이 같아 충돌하는 것을 막기 위해 사용

□ 이미 패키지를 사용해왔음

```
import java.time.LocalDateTime;  
import java.util.Date;
```

□ 자바의 패키지

- java로 시작되는 패키지들은 기본 제공 패키지
- javax는 확장 개념
- time, util은 서브 그룹 개념

패키지(Package)

□ 패키지과 클래스 예

```
java.lang.Object  
java.applet.AudioClip  
java.awt.Component.AccessibleAWTComponent  
javax.swing.text.html.HTML  
javax.imageio.ImageReader
```

□ 패키지 이름 결정 방법

- 충돌하지 않게 자유롭게 정할 수 있음
- 관례적으로 URL을 거꾸로 씀
- 상명대학교 (www.smu.ac.kr)에서 만들면 kr.ac.smu 패키지 사용 권장

패키지(Package)

- ❑ 패키지는 소스 코드나 class 파일(컴파일된 결과물)들을 디렉토리(폴더) 구조를 이용해서 관리
- ❑ 패키지 이름은 폴더 구조와 관련되어 있음
- ❑ 패키지 이름은 단어를 '.'으로 구분
- ❑ 단어는 폴더 이름, '.'은 폴더를 구분하는 '₩' (윈도우) 또는 '/' (유닉스 계열)
 - 예를 들어 java.time.LocalDateTime 클래스 파일은 java/time 또는 java₩time 폴더에 LocalDateTime.class라는 이름으로 저장됨
 - 같은 폴더에는 해당 패키지에 포함되어 있는 다른 클래스 파일들도 있음
 - LocalDateTime.java 파일도 java/time 폴더에 있어야 함

패키지(Package)

▣ 다른 예

- 소스 코드는 src, 컴파일된 코드는 classes에 있다고 가정

패키지를 포함한 클래스 이름	폴더 위치
java.applet.AudioClip	src/java/applet/AudioClip.java classes/java/applet/AudioClip.class
javax.swing.text.html.HTML	src/javax/swing/text/html/HTML.java classes/javax/swing/text/html/HTML.class
kr.co.smu.Dice	src/kr/co/smu/Dice.java classes/kr/co/smu/Dice.class

패키지 사용법

- 패키지의 클래스를 사용하려면 어떤 패키지에 속한 클래스인지 정확하게 코드에 표기해야 함
- 예

```
java.util.Date date = new java.util.Date();  
  
// 현재 년도에서 1900을 뺀 값을 출력  
System.out.println(date.getYear());
```

- import 사용

```
import java.util.Date;  
  
Date date = new Date();  
  
// 현재 년도에서 1900을 뺀 값을 출력  
System.out.println(date.getYear());
```

패키지 사용법

- 같은 패키지에 있는 클래스를 여러 개 사용한다면?

```
import java.util.Vector;  
import java.util.Random;  
import java.util.ArrayList;  
import java.util.Calendar;  
import java.util.Timer;  
import java.util.UUID;
```

- 패키지 내 전체 클래스를 한 번에 import

```
import java.util.*;
```

- 섞어서 사용하는 것도 가능 (필요는 없음)

```
import java.util.ArrayList;  
java.util.Calendar;  
import java.util.*;  
ArrayList alist = new ArrayList();
```

패키지 사용법, 패키지를 UML로 표기하기

- 패키지 이름이 다르지만 클래스 이름이 같다면?

```
import java.util.*;  
import kr.co.smu.*;  
  
java.util.Random rand1;  
kr.co.smu.Random rand2;
```

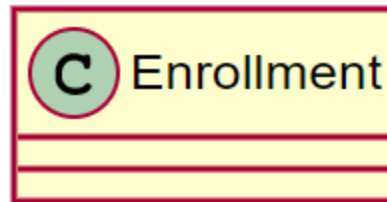
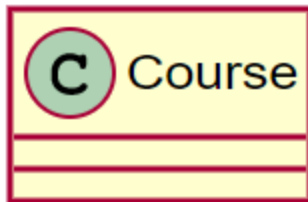
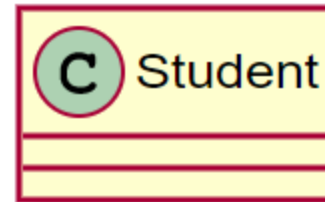
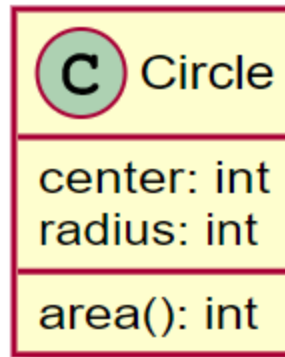
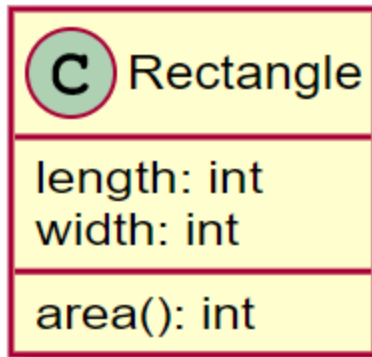
- 패키지를 UML로 표기하기

- PlantUML에서 표기 방법

```
package 패키지이름 {  
    패키지에 포함되는 클래스 이름  
}
```


패키지 사용법, 패키지를 UML로 표기하기

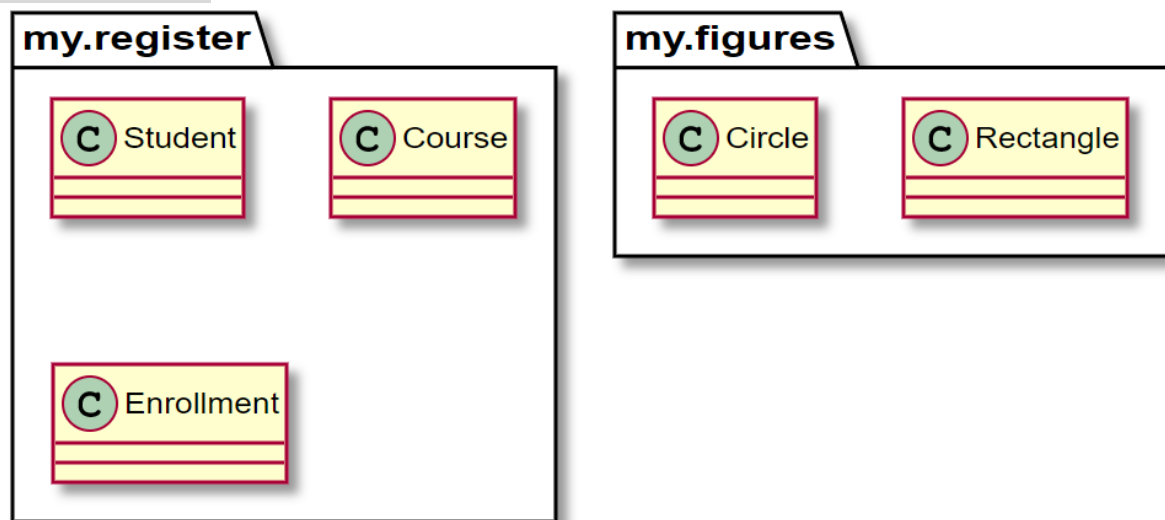
▣ 패키지로 묶지 않은 클래스



패키지를 UML로 표기하기

▣ 패키지로 묶은 경우

```
@startuml
package my.register {
    class Student
    class Course
    class Enrollment
}
package my.figures {
    class Rectangle
    class Circle
}
@enduml
```



패키지 만들기

- 소스 코드와 클래스 파일들을 적절한 폴더 구조를 구성해서 넣어야 함
- 자바 소스 코드에 "package 패키지_이름"을 적고 코드 구현
- 예

```
package my.figures;  
class Rectangle {  
    ...  
}
```

- Rectangle.java를 my/figures/Rectangle.java로 저장

실습 문제 6: 지정된 패키지로 코드를 작성하고 컴파일하고 실행

□ 문제

- Hello 클래스를 com.my.hello 패키지로 만들고, 이 클래스를 사용하는 com.my.app 패키지의 Main 클래스를 만들기

□ 해결

- 패키지 구성해서 코드 작성 순서
 - 패키지 이름 정하기
 - 이미 com.my.hello와 com.my.app으로 정해짐
 - 디렉토리 생성
 - com\my\hello와 com\my\app 폴더 생성
 - 코드 작성
 - 패키지 이름을 소스에 넣기

실습 문제 6: 지정된 패키지로 코드를 작성하고 컴파일하고 실행

▣ 코드 작성

```
// Hello.java
package com.my.hello;

public class Hello {
    String toWhom = "world";
    public Hello() {
    }
    public Hello(String toWhom) {
        this.toWhom = toWhom;
    }
    public void sayHello() {
        System.out.printf("hello %s!", toWhom);
    }
}
```

실습 문제 6: 지정된 패키지로 코드를 작성하고 컴파일하고 실행

```
// Main.java
package com.my.app;

import com.my.hello.Hello;

public class Main {
    public static void main(String[] args) {
        Hello hm = new Hello("ycho");
        hm.sayHello();
    }
}
```

□ 컴파일

```
c:\code\java\07\PackageTest>javac -d classes
-sourcepath src src\com\my\hello\Hello.java
C:\code\java\07\PackageTest>javac -d classes -cp
.;classes; src\com\my\app\Main.java
```

실습 문제 6: 지정된 패키지로 코드를 작성하고 컴파일하고 실행

□ 실행

```
c:\code\PackageTest>java -cp .;./classes com.my.app.Main
```

□ 실행 결과

```
C:\Code\java\07\PackageTest>java -cp .;./classes com.my.app.Main  
hello ycho!
```

패키지와 접근 제어자

- 디폴트(default) 또는 package private
 - 패키지 내부에서 private
 - 패키지 내부에 있는 다른 클래스에서는 자유롭게 접근하고 사용 가능
 - 패키지 외부에 있는 클래스에서는 접근 불가
- 클래스에도 public이 없으면 다른 패키지에서 접근 불가

패키지와 접근 제어자

```
// Hello.java
package com.my.hello;

class Hello { // public이 없음에 주의
    String toWhom = "world";
    public Hello() {
    }
    public Hello(String toWhom) {
        this.toWhom = toWhom;
    }
    public void sayHello() {
        System.out.printf("hello %s!", toWhom);
    }
}
```

```
C:\Code\java\07\PackageTest>javac -d classes -  
sourcepath src src\com\my\hello\Hello.java
```

```
C:\Code\java\07\PackageTest>javac -d classes -cp  
.;classes; src\com\my\app\Main.java  
src\com\my\app\Main.java:4: error: Hello is not public  
in com.my.hello; cannot be accessed from outside  
package  
import com.my.hello.Hello;  
                        ^
```

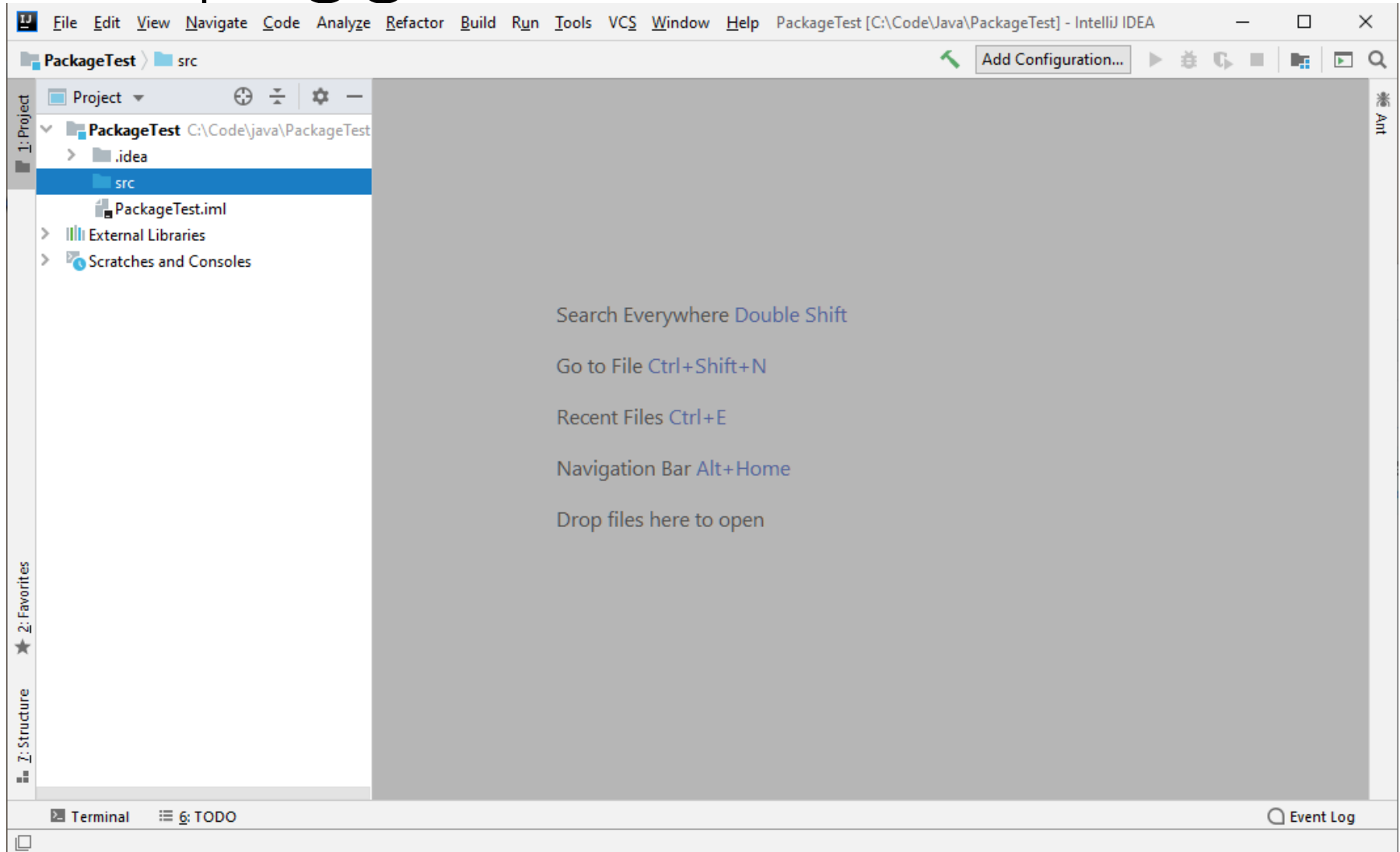
```
src\com\my\app\Main.java:8: error: Hello is not public  
in com.my.hello; cannot be accessed from outside  
package  
    Hello hm = new Hello("ycho");  
    ^
```

```
src\com\my\app\Main.java:8: error: Hello is not public  
in com.my.hello; cannot be accessed from outside  
package  
    Hello hm = new Hello("ycho");  
                  ^
```

3 errors

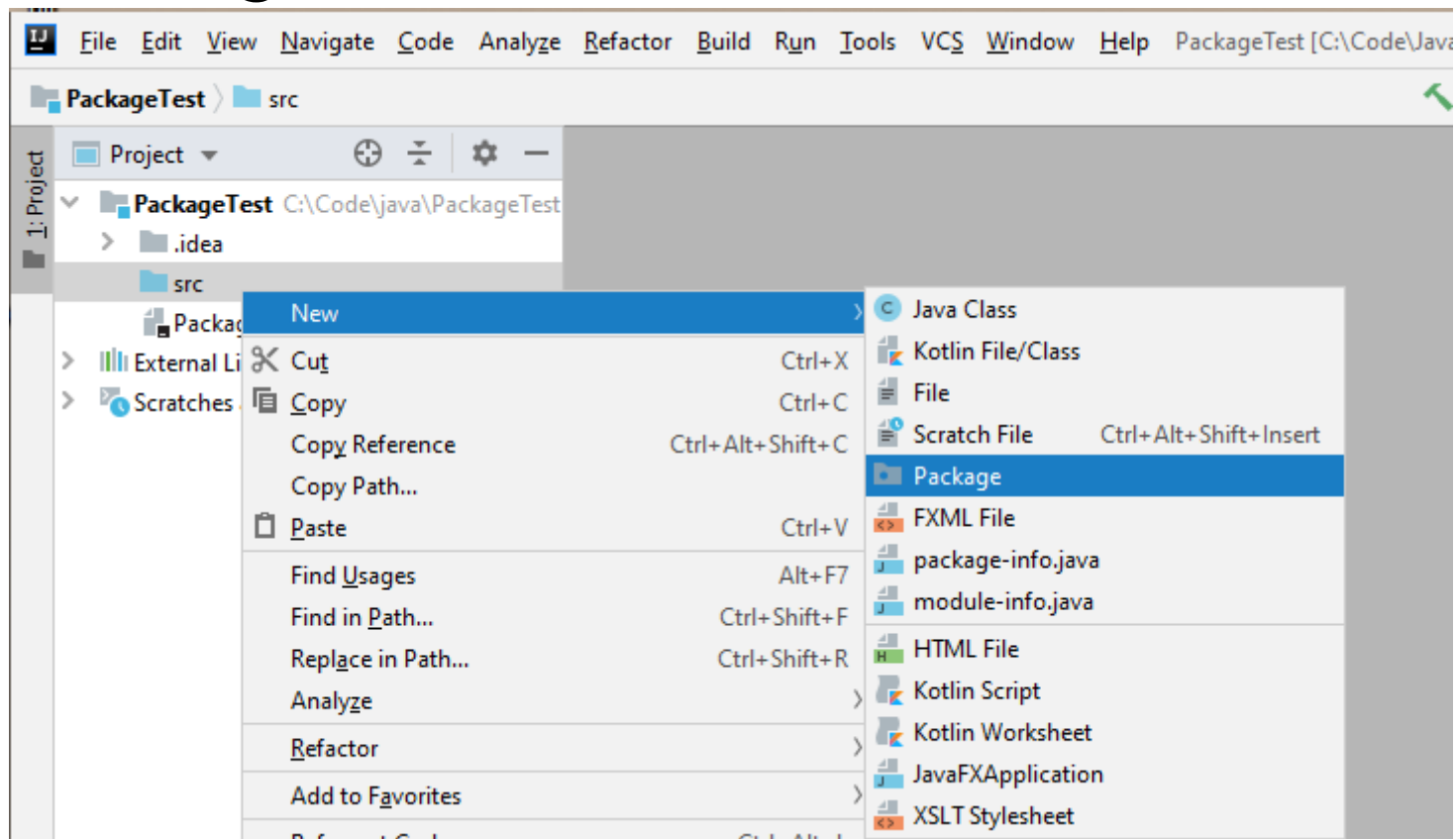
인텔리제이 IDEA에서 패키지 생성하고 컴파일 하기

□ 프로젝트 생성



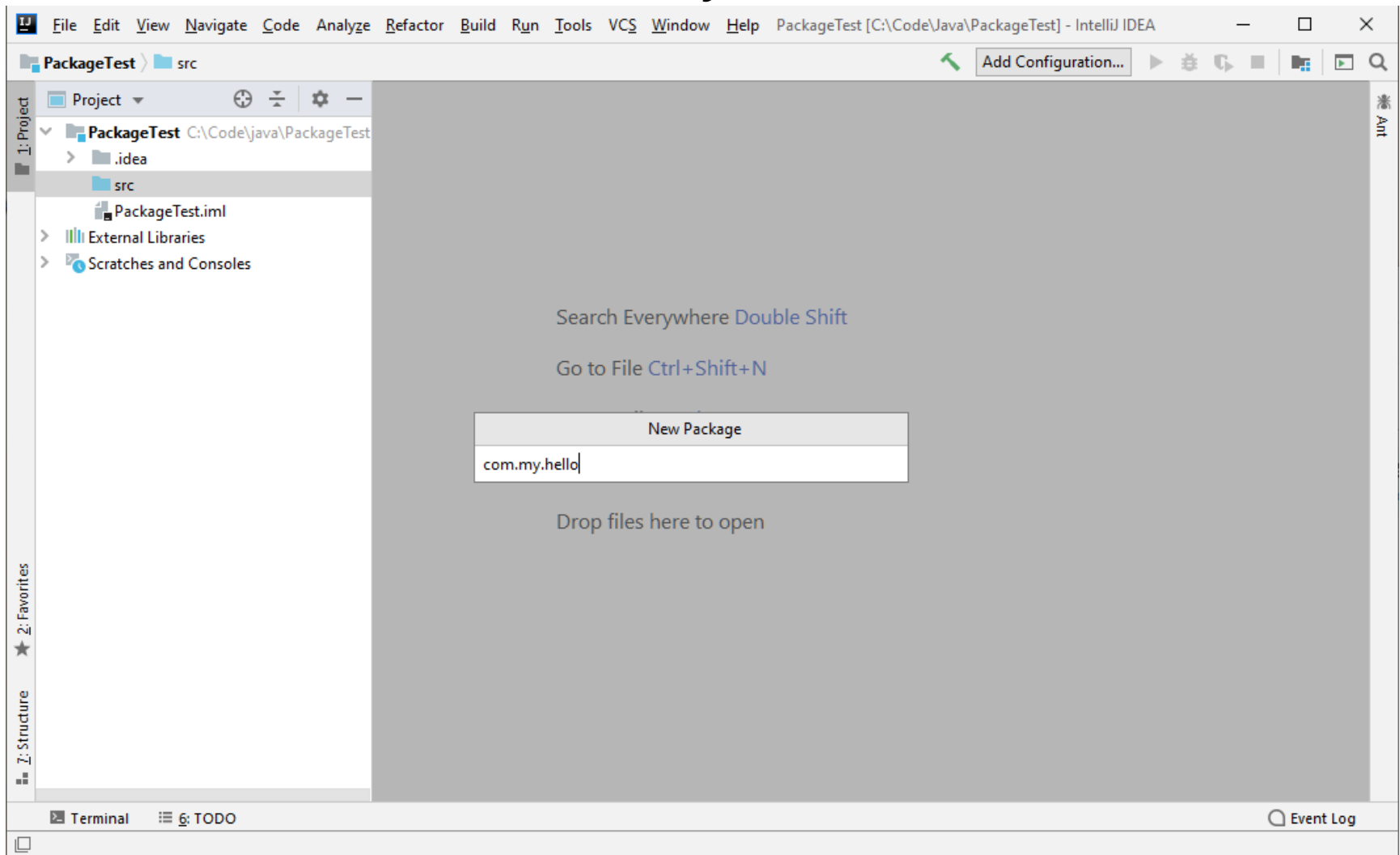
인텔리제이 IDEA에서 패키지 생성하고 컴파일 하기

- 프로젝트에서 src 선택 → 오른쪽 버튼 클릭 → New → Package 선택



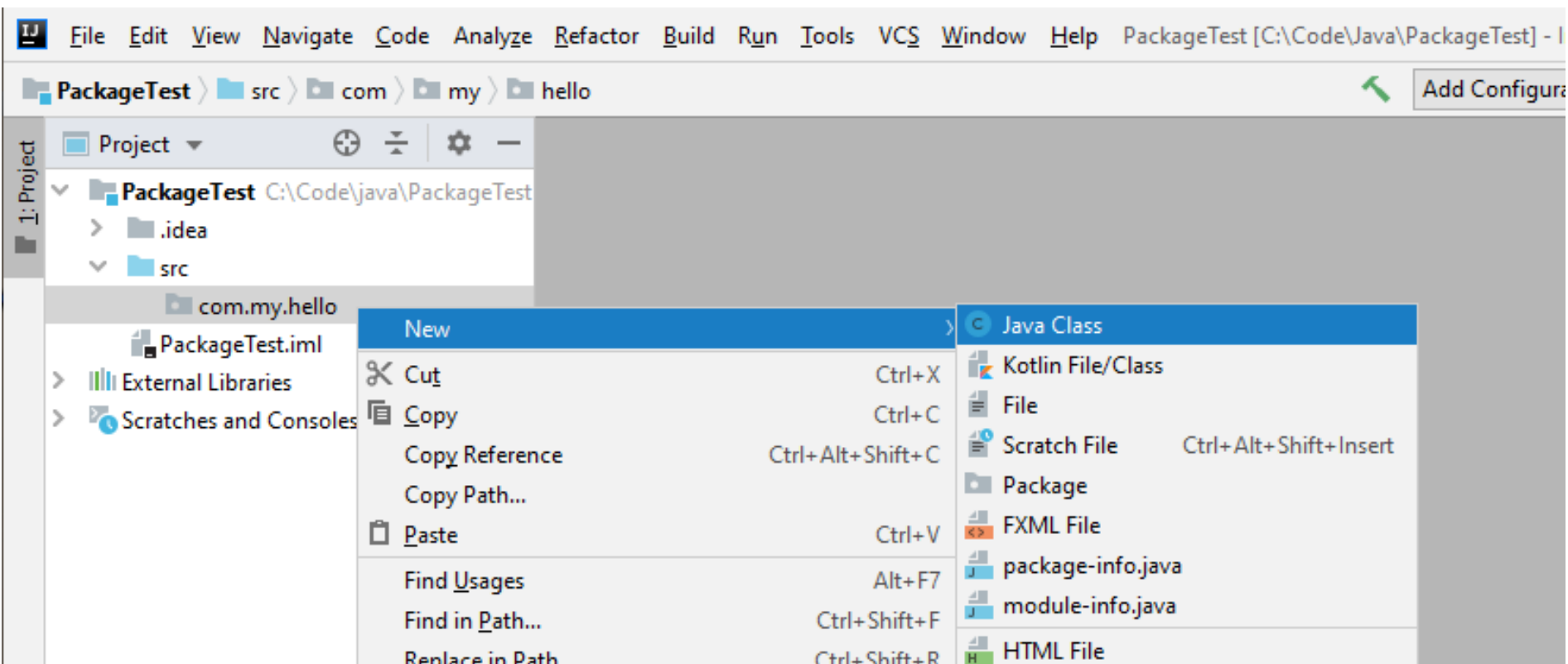
인텔리제이 IDEA에서 패키지 생성하고 컴파일 하기

□ 패키지 이름 입력. com.my.hello 입력



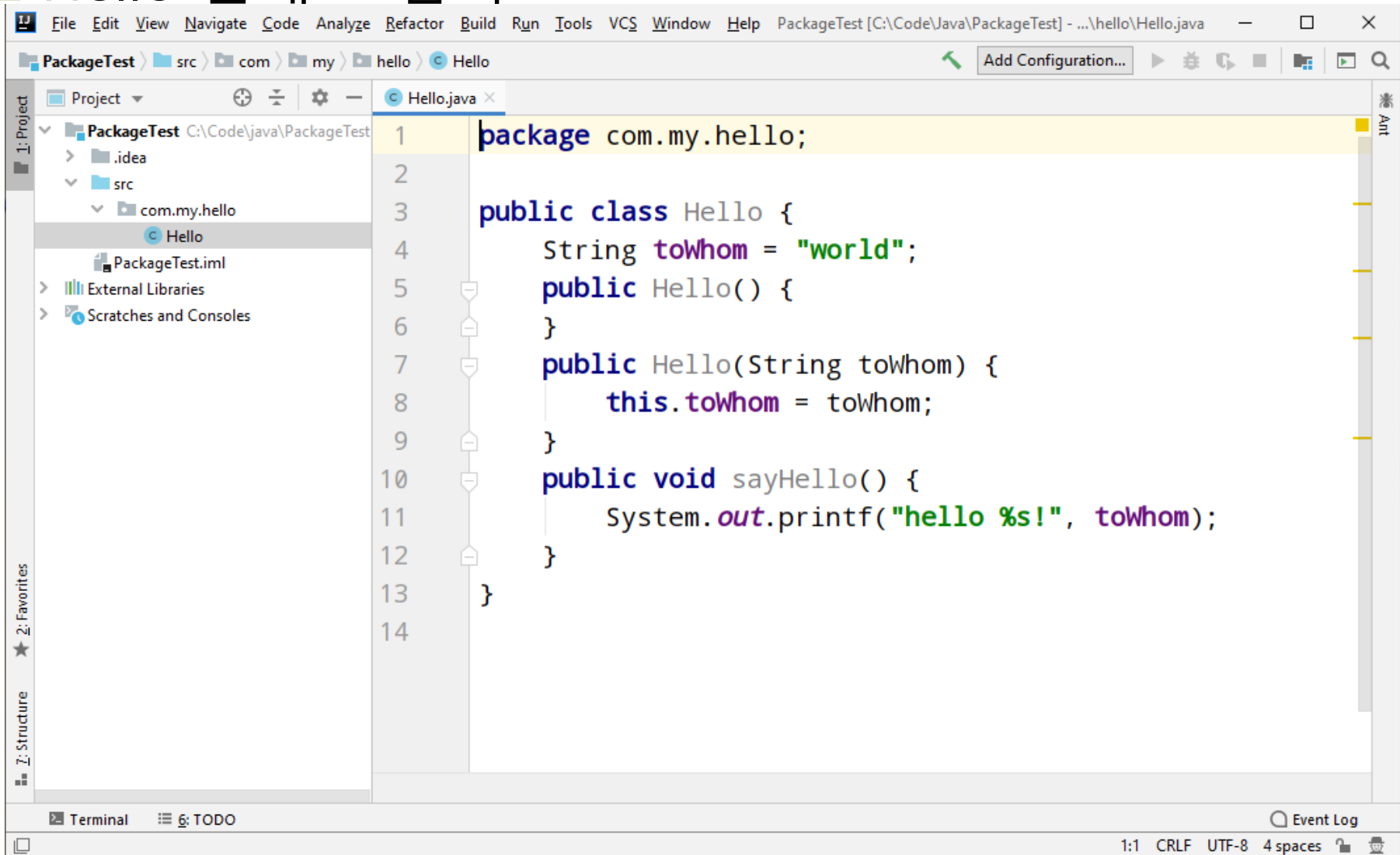
인텔리제이 IDEA에서 패키지 생성하고 컴파일 하기

- 클래스 생성. 앞에서 만들었던 패키지 선택 → 오른쪽 버튼 클릭 → New → Class 선택



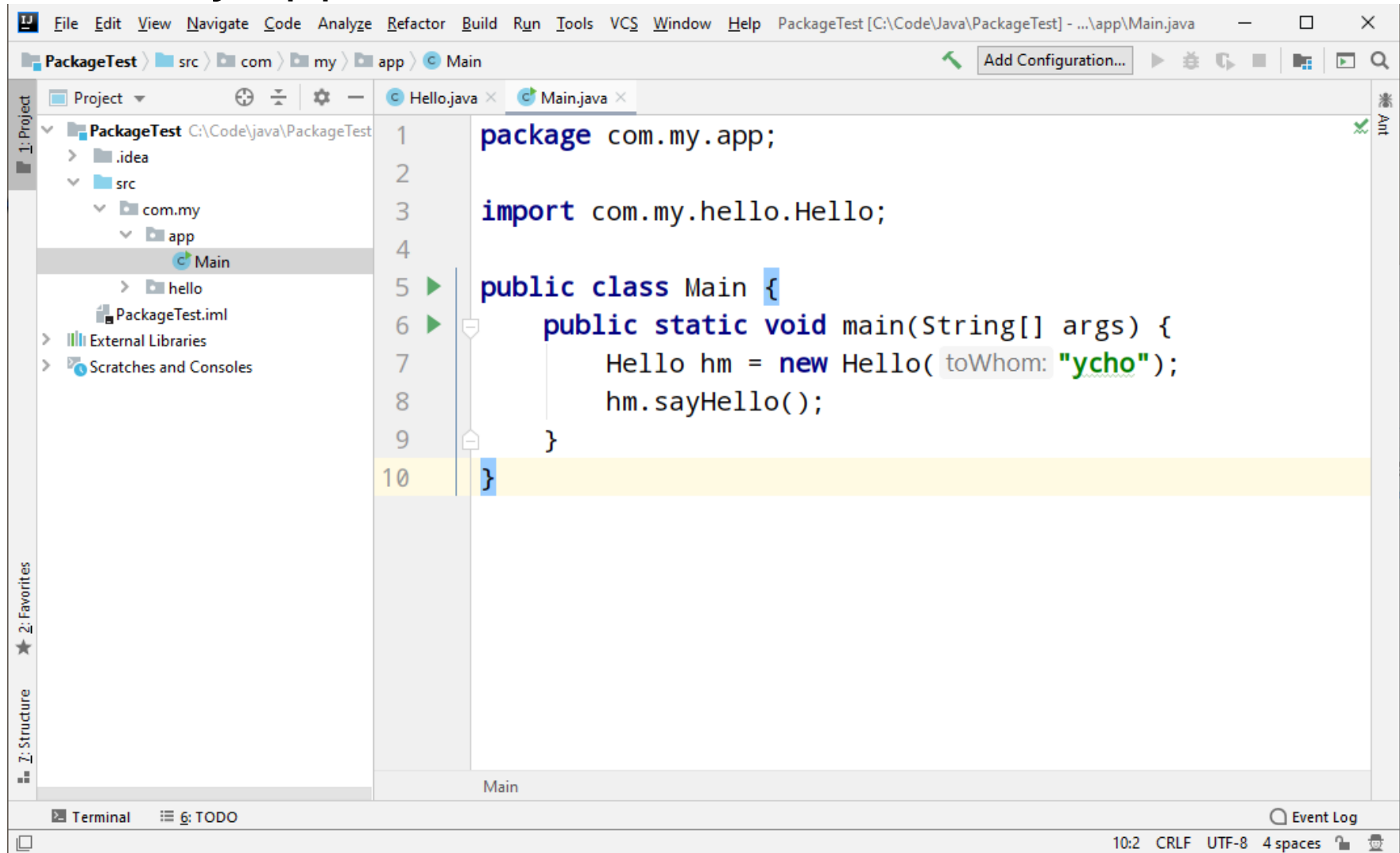
인텔리제이 IDEA에서 패키지 생성하고 컴파일 하기

□ Hello 클래스 입력



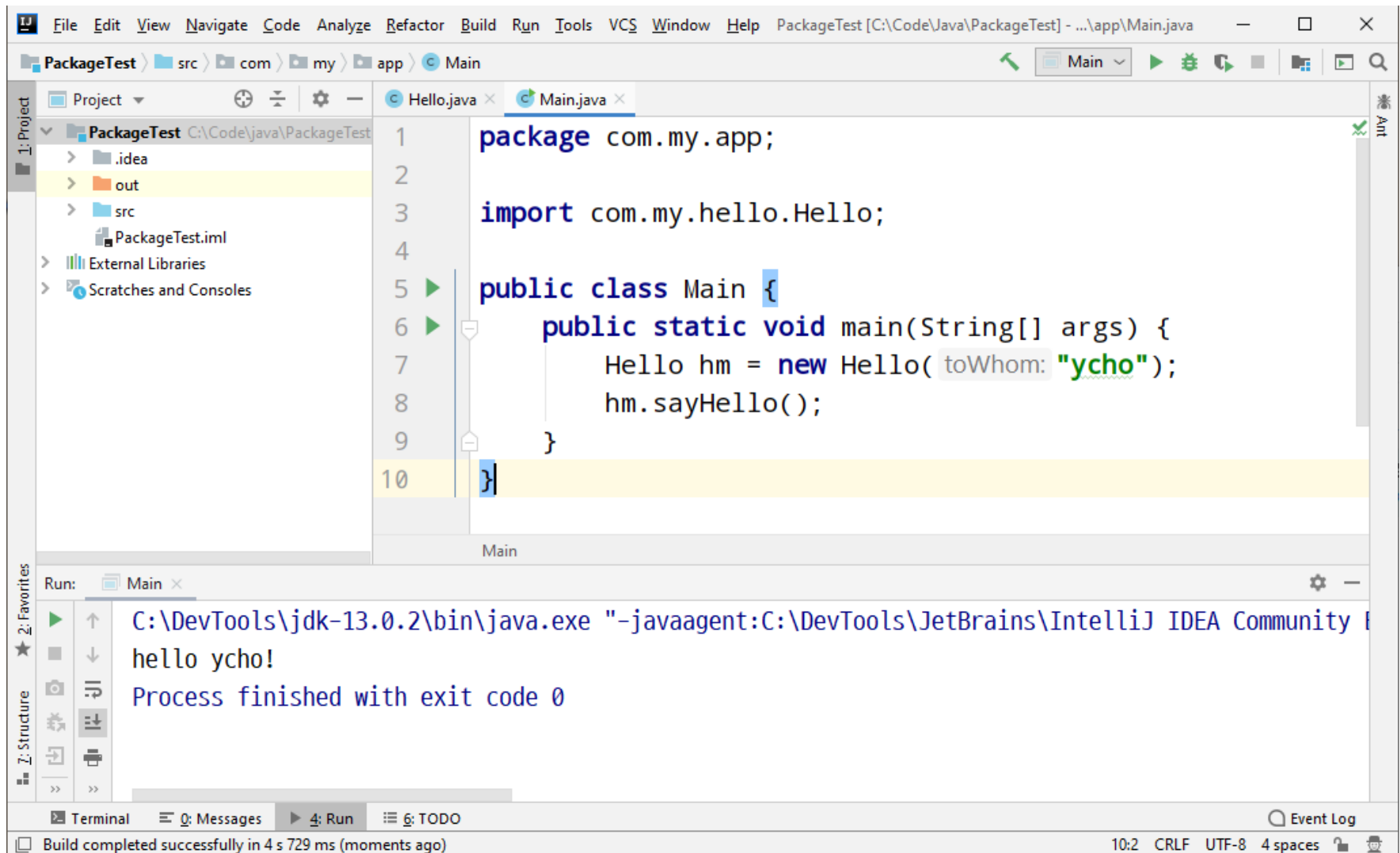
인텔리제이 IDEA에서 패키지 생성하고 컴파일하기

□ com.my.app 패키지 생성하고 Main클래스 구현



인텔리제이 IDEA에서 패키지 생성하고 컴파일 하기

□ Main 실행



jar 파일 – 연관된 클래스를 압축해놓은 꾸러미

- jar파일은 Java Archive의 약자로 자바 클래스 파일들을 묶어서 배포하기 쉽도록 만든 패키지 파일 형식
- jar 파일의 장점
 - 배포가 용이함
 - 패키지를 사용하면 폴더 구조 유지
 - jar파일을 바로 실행 가능
- 실행용 jar 파일 만들려면 매니페스트 파일이 필요

```
Main-Class: com.my.app.Main
```

- 두 번째 빈 줄이 반드시 필요함

jar 파일 생성 및 실행

- manifest.txt 작성 후 저장
- jar 명령 실행

선택 사항	설명
c(reate)	새로운 jar 파일을 생성
v(erbos)	자세한 정보를 출력함
f(ile name of jar)	생성할 jar 파일의 이름을 지정
m(anifest)	manifest 파일의 이름을 지정
C(hange dir)	지정된 디렉토리로 변경하고 파일 또는 폴더를 포함시킴. 서브 디렉토리가 있으면 재귀적으로 파일을 포함시킴
t(list table of contents)	f 옵션과 함께 사용되어 주어진 jar 파일의 내용을 화면에 출력한다.

jar 파일 생성 및 실행

□ 생성 방법

```
jar -cvfm jarfile-name manifest-filename -C classpath  
class-filename_or_folder_name
```

□ 실행

```
java -jar executable_jar_file_name
```

□ hello.jar 생성

```
c:\code\java\07\PackageTest>jar -cvfm hello.jar  
manifest.txt -C classes .
```

jar 파일 생성 및 실행

□ jar 파일 내용 확인

```
c:\code\java\07\PackageTest> jar -tf hello.jar
```

□ 출력 화면

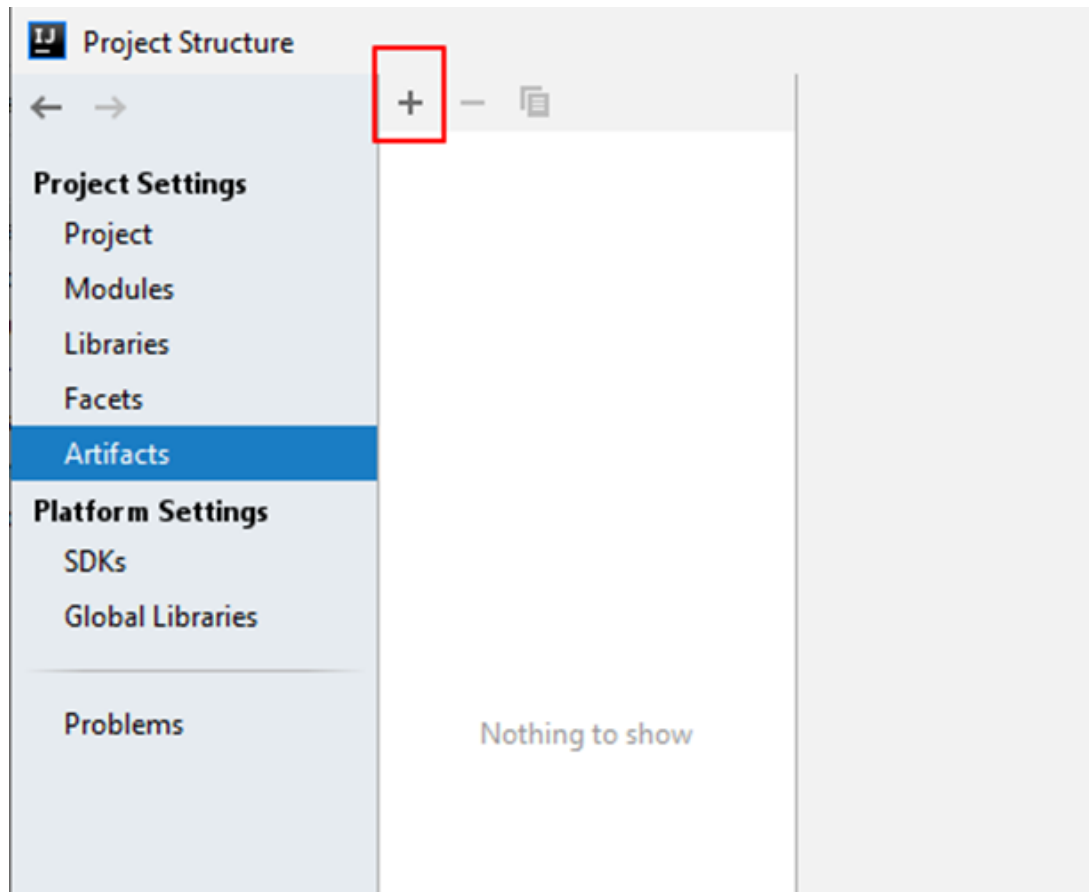
```
C:\code\java\07\PackageTest> jar -tf hello.jar
META-INF/
META-INF/MANIFEST.MF
com/
com/my/
com/my/app
com/my/app/Main.class
com/my/hello/
com/my/hello/Hello.class
```

□ 실행

```
C:\code\java\07\PackageTest> java -jar hello.jar
```

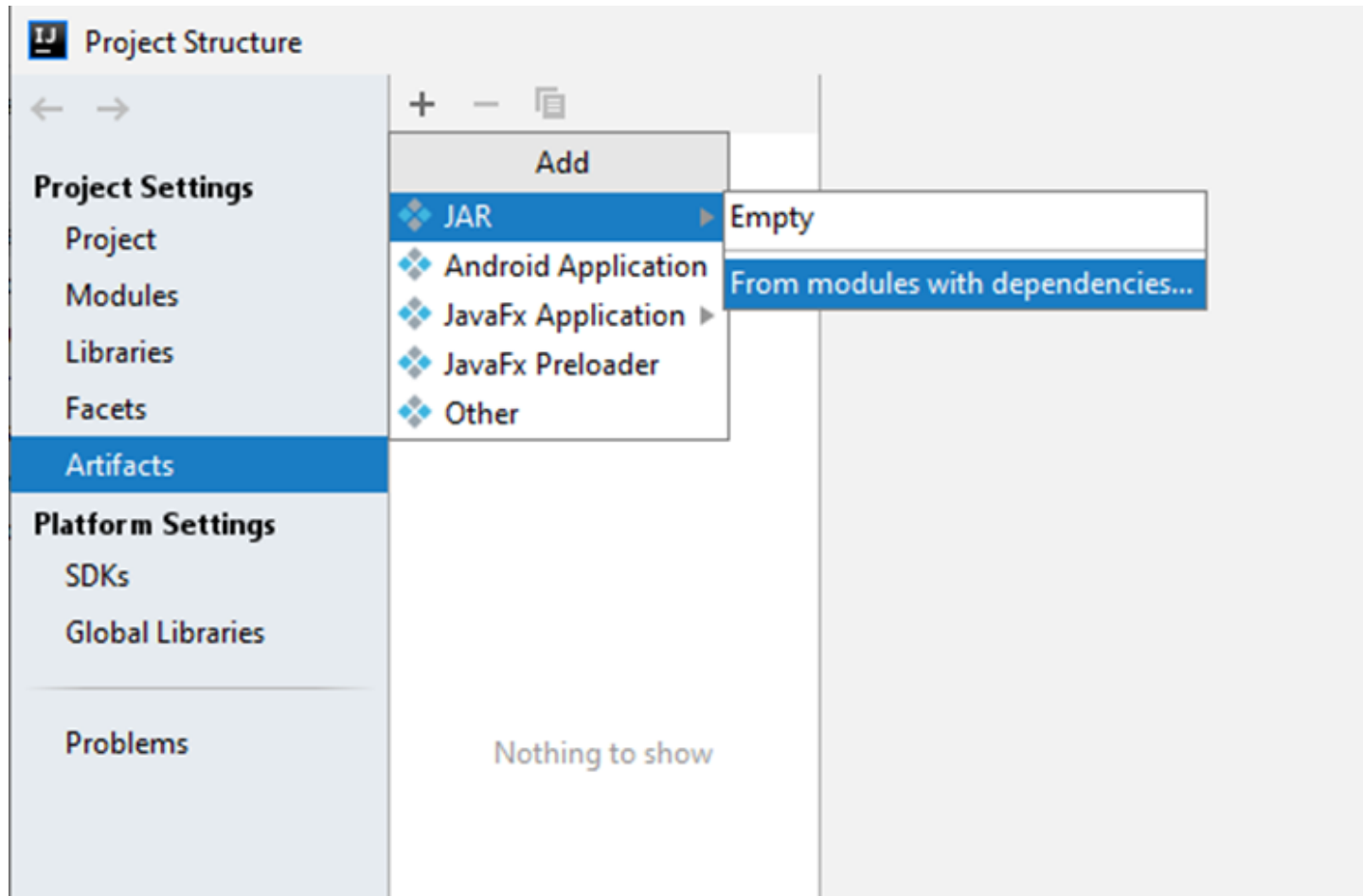
인텔리제이 IDEA에서 jar 파일 생성

- File → Project Structure...를 선택
- Project Structure 대화 상자 왼쪽에서 Artifacts를 선택하고 가운데 부분에서 '+'를 선택



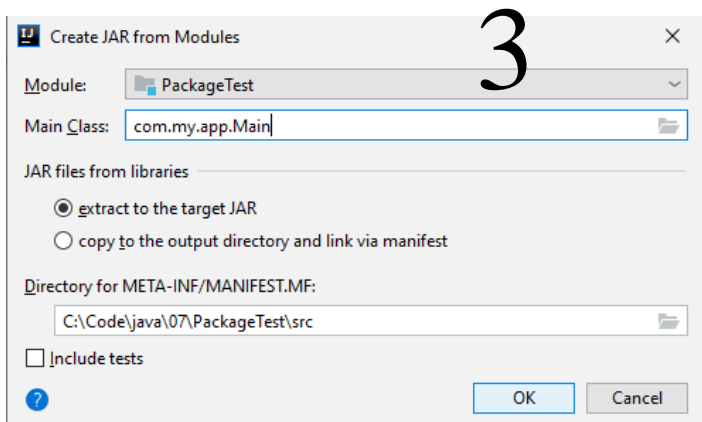
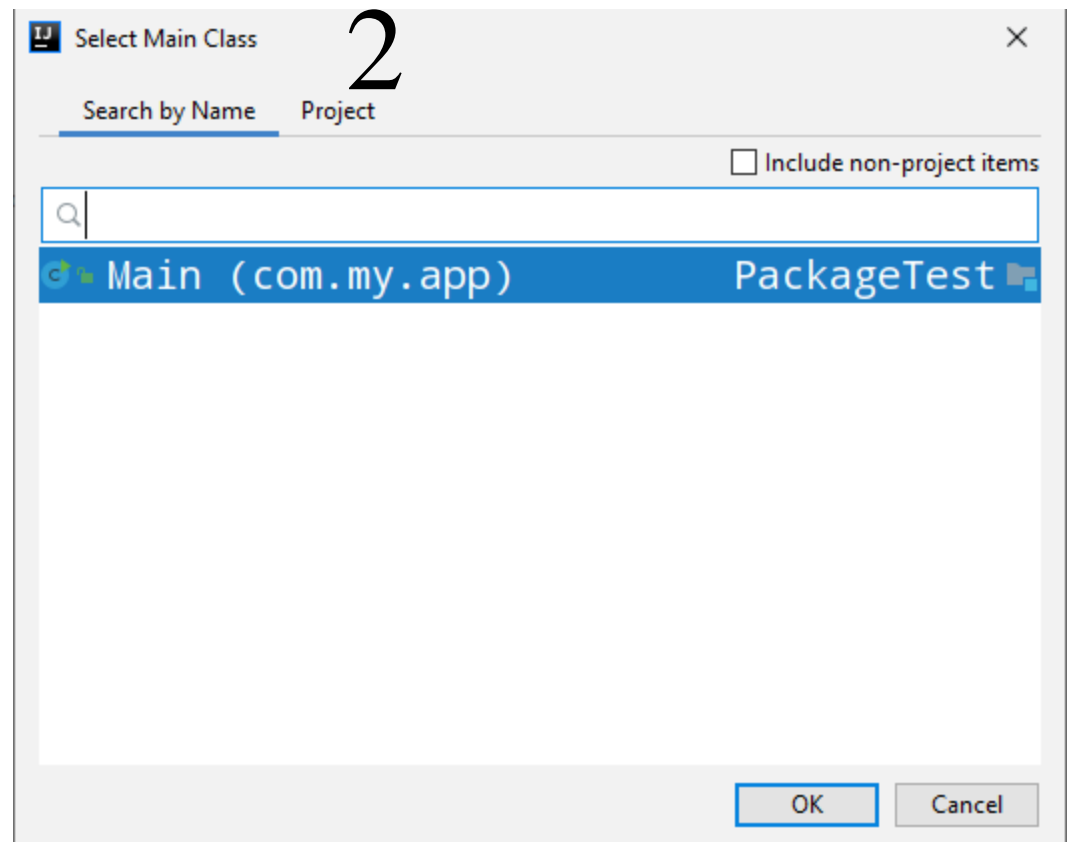
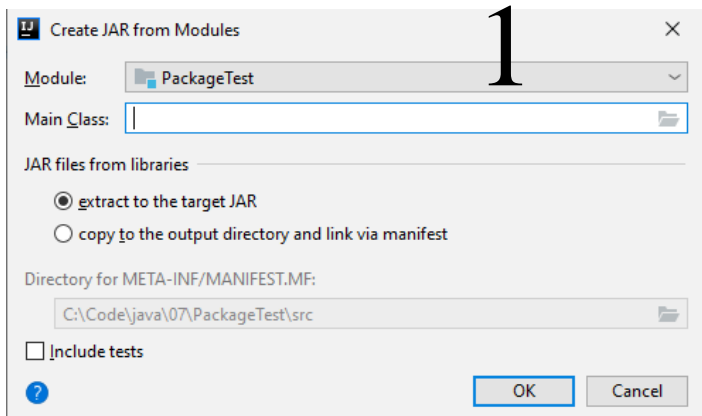
인텔리제이 IDEA에서 jar 파일 생성

- 메뉴에서 JAR → "From modules with dependencies..."를 선택



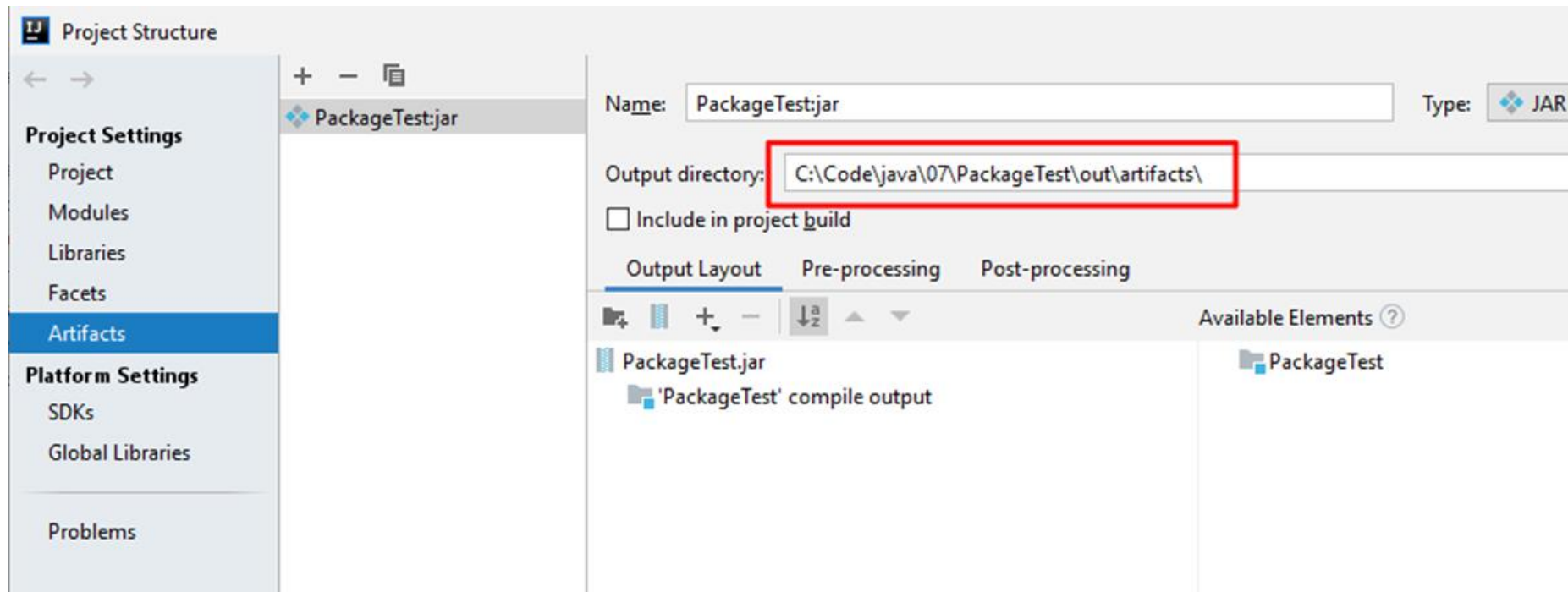
인텔리제이 IDEA에서 jar 파일 생성

- 대화 상자에서 main() 함수가 있는 클래스 선택
 - 여기서는 com.my.app의 Main 클래스 선택



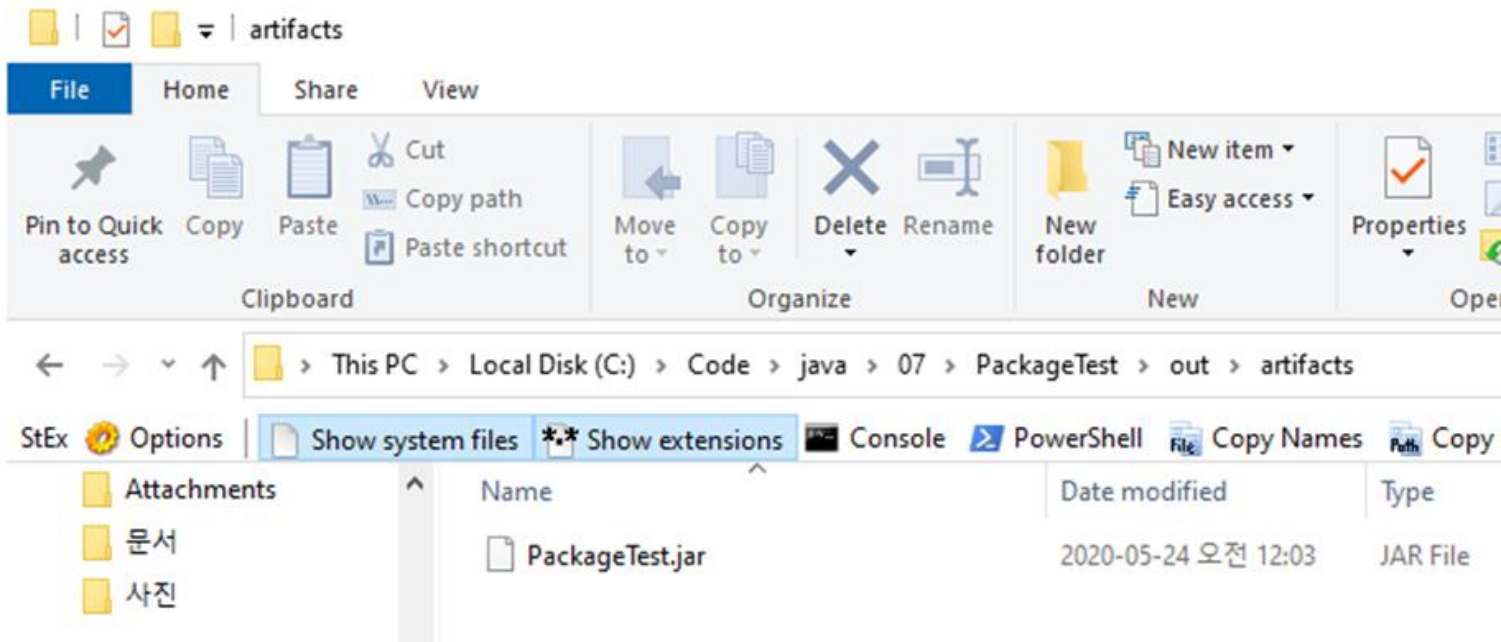
인텔리제이 IDEA에서 jar 파일 생성

▣ jar파일을 생성할 경로 확인 후 OK버튼 클릭



인텔리제이 IDEA에서 jar 파일 생성

- 메뉴에서 Build → Build Artifacts...를 선택
- 경로에 프로젝트 이름으로 jar 파일 생성됨



□ 실행 결과

```
C:\Code\java\07\PackageTest\out\artifacts>java -jar PackageTest.jar  
hello ycho!
```

클래스 라이브러리 경로

- import 키워드를 이용해서 클래스를 사용하겠다고 표시하면, 클래스 라이브러리 경로(classpath)로 지정된 폴더에서 클래스를 찾게 됨
- 패키지가 사용된다면 해당 폴더 구조가 classpath로 지정된 폴더 안에 있어야 함
- 윈도우에서 classpath가 "c:\classes;c:\java\classes"로 지정되었다고 가정
 - "my.register.Student"를 제대로 사용하려면 "c:\classes\my\register\Student.class" 또는 "c:\java\classes\my\register\Student.class"로 존재
- jar파일도 경로에 지정 가능
 - "c:\classes;c:\java\classes;c:\code\my.jar"