

소프트웨어공학

클래스 다이어그램



UML(Unified Modeling Language)

- 클래스 다이어그램
 - 프로그램을 구성하는 클래스의 모습과 클래스 간의 연관 관계에 대한 것을 정리한 것

프로그램 제작에서 필요한 클래스 자체와 클래스 간의 관계로 분리하여 모델링

분류	다이어그램 유형	목적	
구조 다이어그램 (structure diagram)	클래스 다이어그램 (class diagram)	시스템을 구성하는 클래스들 사이의 관계를 표현한다.	
	객체 다이어그램 (object diagram)	객체 정보를 보여준다.	
	복합체 구조 다이어그램 (composite structure diagram)	복합 구조의 클래스와 컴포넌트 내부 구조를 표현한다.	
	배치 다이어그램 (deployment diagram)	소프트웨어, 하드웨어, 네트워크를 포함한 실행 시스템의 물리 구조를 표현한다.	
	컴포넌트 다이어그램 (component diagram)	컴포넌트 구조 사이의 관계를 표현한다.	
	패키지 다이어그램 (package diagram)	클래스나 유즈 케이스 등을 포함한 여러 모델 요소들을 그룹화해 패키지를 구성하고 패키지들 사이의 관계를 표현한다.	
행위 다이어그램 (behavior diagram)	활동 다이어그램 (activity diagram)	업무 처리 과정이나 연산이 수행되는 과정을 표현한다.	
	상태 머신 다이어그램 (state machine diagram)	객체의 생명주기를 표현한다.	
	유즈 케이스 다이어그램 (use case diagram)	사용자 관점에서 시스템 행위를 표현한다.	
	상호작용 다이어그램 (interaction diagram)	순차 다이어그램 (sequence diagram)	시간 흐름에 따른 객체 사이의 상호작용을 표현한다.
		상호작용 개요 다이어그램 (interaction overview diagram)	여러 상호작용 다이어그램 사이의 제어 흐름을 표현한다.
		통신 다이어그램 (communication diagram)	객체 사이의 관계를 중심으로 상호작용을 표현한다.
		타이밍 다이어그램 (timing diagram)	객체 상태 변화와 시간 제약을 명시적으로 표현한다.

객체 지향의 개념

객체 지향 프로그래밍의 핵심 개념

[객체 지향 프로그래밍의 데이터 및 메소드 다루기]

- 클래스 : 프로그램에서 사용하는 데이터와 메소드를 묶어서 만드는 사용자 정의 자료형
- 객체 : 클래스의 형태를 가지는 데이터 이름

```
(예)    int salary;           // 일반 프로그램에서의 데이터 선언,  
                                     // int ➔ 데이터 타입, salary ➔ 데이터의 이름  
                                     // int는 기본 데이터타입이므로 크기를 알고있어서 별도의 new가 불필요  
  
    Book book = new Book(); // 객체지향 프로그램에서의 데이터 선언  
                             // Book ➔ 데이터 타입(사용자정의형), book ➔ 데이터의 이름  
                             // new를 사용하는 이유는 선언한 데이터타입(Book)의 크기를 미리 알수 없기 때문
```

캡슐화(Encapsulation)

클래스를 만들어서, 잘 사용하고 있다.
그런데, 클래스에 있는 메서드와 데이터는 한번에 동시에 수행되어야 하는 작업단위이므로 외부에서 아무나 접근하는 것은 조금 위험해 보인다.
그래서, 클래스에 캡슐화의 개념이 도입되었다

캡슐화란 클래스에 있는 데이터나 메소드를 개발자가 정한 규정에 의해서만 접근이 가능하도록 하는 것(private, public)과 수정을 제한 하는 것(const)을 말한다

(예) 클래스가 사용하는 어떤 데이터를 private로 만들고
이 데이터를 사용할수 있는 코드를 따로 만들어서 public으로 선언하는 것

[캡슐화의 장점]

- 사용하기 쉽다 (사용되는 문법이 간단하다)
- 관리하기 쉽다 (예상하지 못한 버그나 잘못된 발생을 줄여준다)
- 유연하다 (기능의 추가나 재사용하는 경우에 좋다)

지금 만드는 클래스의 캡슐화를 잘 해 놓으면, 객체가 수행하는 업무 작업을 “블랙박스”로 보고 재사용하는 것이 가능하다.

변동되어서는 안되는 것은 private나 const로 선언하면, 외부에서 변경 불가하다.
(private는 자체에서 변동 되는 경우, const는 자체에서도 변동되지 않는 경우에 사용된다)

상속(Inheritance)

- 큰 프로젝트를 수행하다 보면,
클래스를 여러 개 선언하게 되는데,
서로 다른 클래스에 똑같은 코드를 반복해서 집어 넣게 되는 상황이 많이 발생하게 된다
- 이때, 서로 다른 클래스에 동일 코드가 반복되어서 생기는 비효율성과 에러의 가능성을 줄이기 위하여
연관있고 유사한 행동을 공유하는 클래스를 만들어 해결할 수 있는데 이것이 상속이다(inheritance).

➔ 상속은 클래스를 정리하는 개념일뿐, 객체와는 무관

(상속을 사용해서 메모리가 적어지거나, 객체가 작아지는 것은 아니다)

➔ 상속은 대규모의 프로그램을 개발하는 경우에 클래스의 체계적 관리가 가능하지만,
한번 제작된 상속은 변경이 어렵고, 프로그램에서 사용하는 객체의 크기가 커지는 문제가 있다

➔ 상속은 사용자 정의 자료형이 가지는 다양한 종류를 체계적으로 정리하기 위한 방법이다

.

[인터페이스와 추상클래스]

- 상속이 중복성을 제거해서 많은 편리함을 주었지만,
- 프로그램을 작성하다 보면, 객체가 어떤 클래스를 상속 받는지 보다는 **어떤 일을 할 수 있는 지를 기준으로 묶어 주어야** 하는 경우가 있다. 이런 경우에 사용되는 것이 바로 **인터페이스(interface)**이다
- 인터페이스에는 메서드나 속성을 선언만 하고 본체(=실제 구현 내용)가 없다. 본체는 상속받는 클래스에서 구현해야 한다.

그래서 인터페이스에 들어가는 모든 메소드는 추상 메소드(abstract method)라고 한다

- 상속과 인터페이스는 잘 고려할 필요가 있다. 둘다, 특정 특성을 하위로 물려주는 성질을 가지지만, **상속된 클래스의 객체는 상위 클래스 전체의 속성을 가지는 큰 클래스가 되고 인터페이스의 객체는 인터페이스만을 가지는 작은 클래스가 된다**
- 그리고, 상속 트리를 그리다 보면, 중간 정도에 있는 클래스는 하위 클래스가 공동으로 사용하는 메소드를 가지지만, 별도의 객체를 생성할 필요가 없는 경우가 있다.

이때 사용하는 것이 **추상 클래스(Abstract Class)**이다. 그러므로 당연히 추상클래스는 인스턴스를 만들 수 없다

다형성(Polymorphism)

상속으로 인하여 발생하는 특성으로, 상위 클래스의 Print 함수와 하위 클래스의 Print 함수가 동일한 이름이지만 다른 결과를 나타내는 것을 말한다

➔ 오버라이딩, 오버로딩에 의해 구현된다

: 오버로딩 ➔ 이름은 같지만, 함수를 구성하는 매개변수 타입이나 개수가 다르면 다른 함수로 인식하는 것

오버라이딩 ➔ 상속받은 클래스에서 상위 클래스의 메소드 재정의에 통한 구현

요구사항 정의

- 유스케이스 다이어그램
- 순차다이어그램
- 활동다이어그램

분석

- 클래스 다이어그램
- 객체 다이어그램
- 상태 머신 다이어그램
- 통신 다이어그램
- 상호작용 다이어그램
- 타이밍 다이어그램

설계

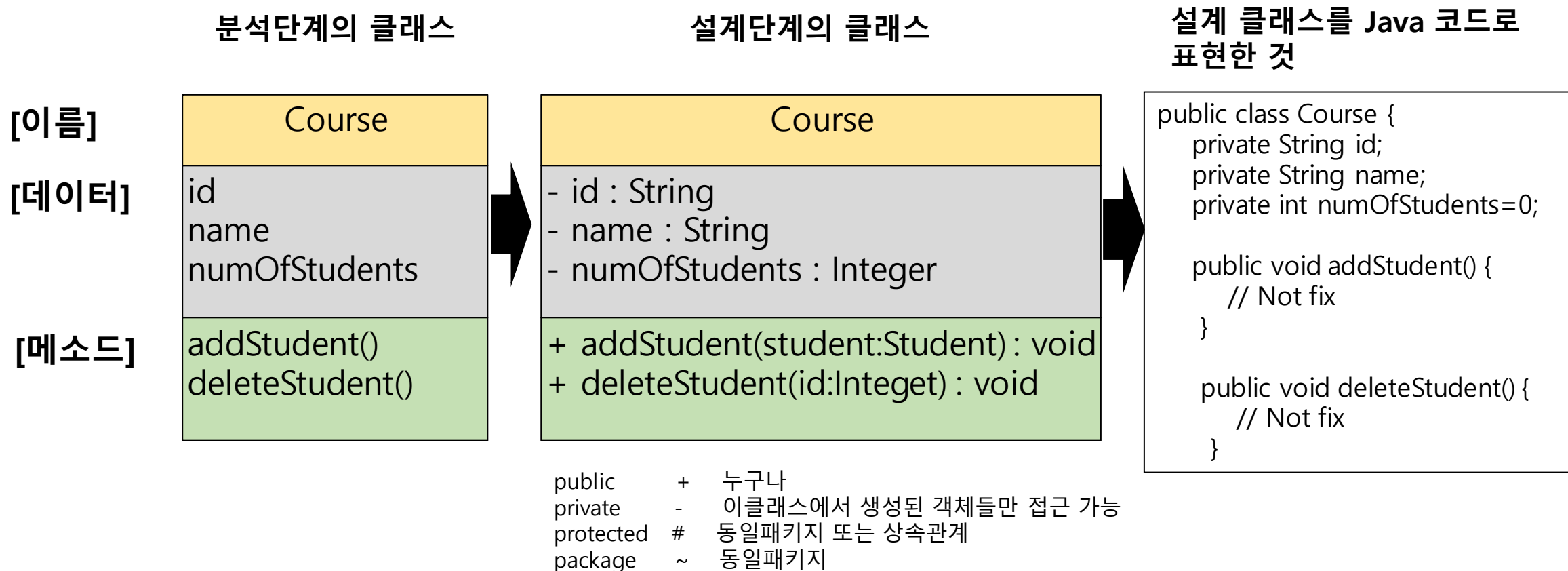
구현

테스트

구현과 테스트는 앞의 작업
결과물을 전부 활용한다

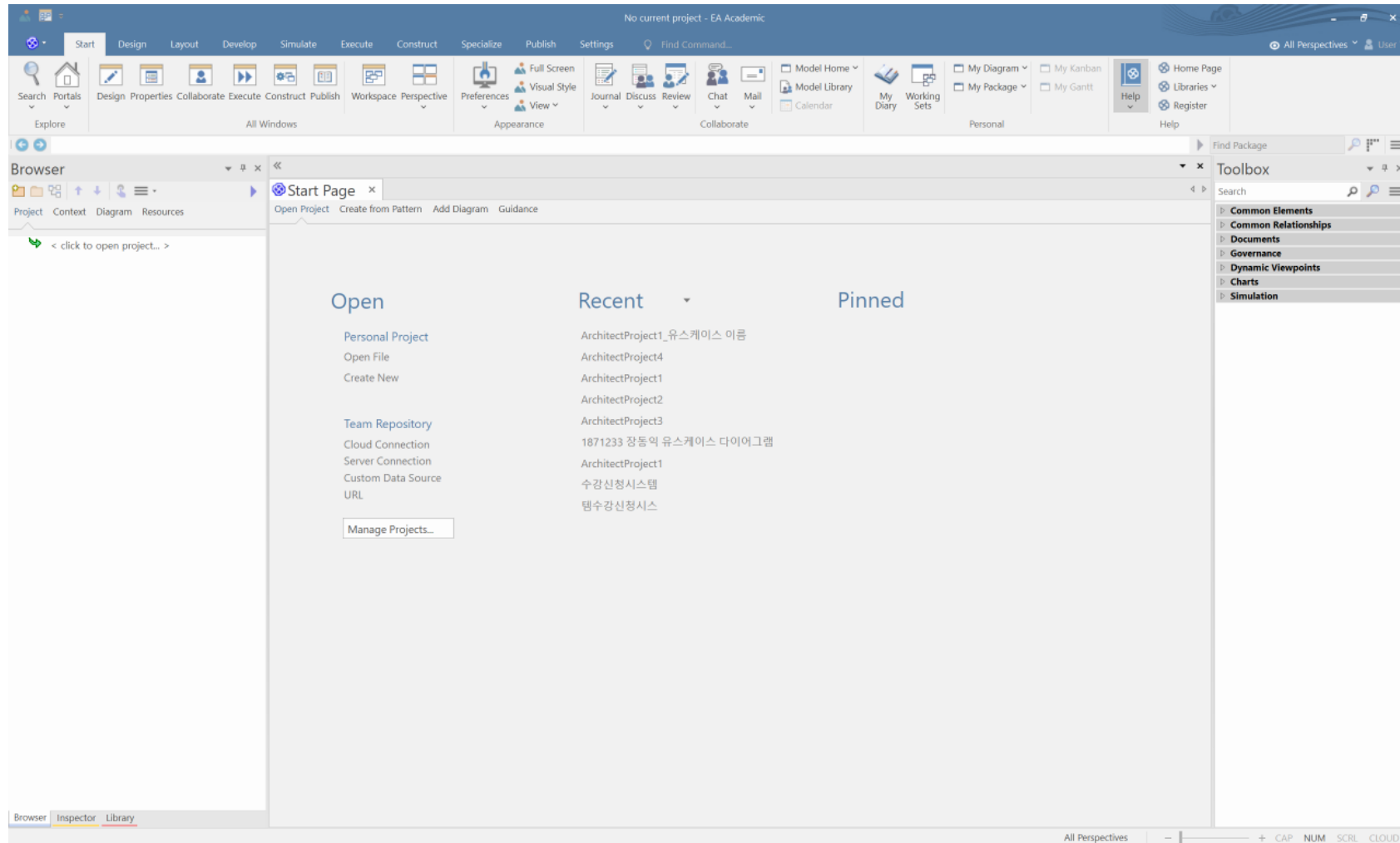
클래스의 표현

- 클래스의 모델링은
 - 프로그램 제작에서 필요하여 정의한 클래스를
 - 클래스 자체와
 - 클래스 간의 연관 관계로 분리하여
 - 모델링하는 것을 말합니다



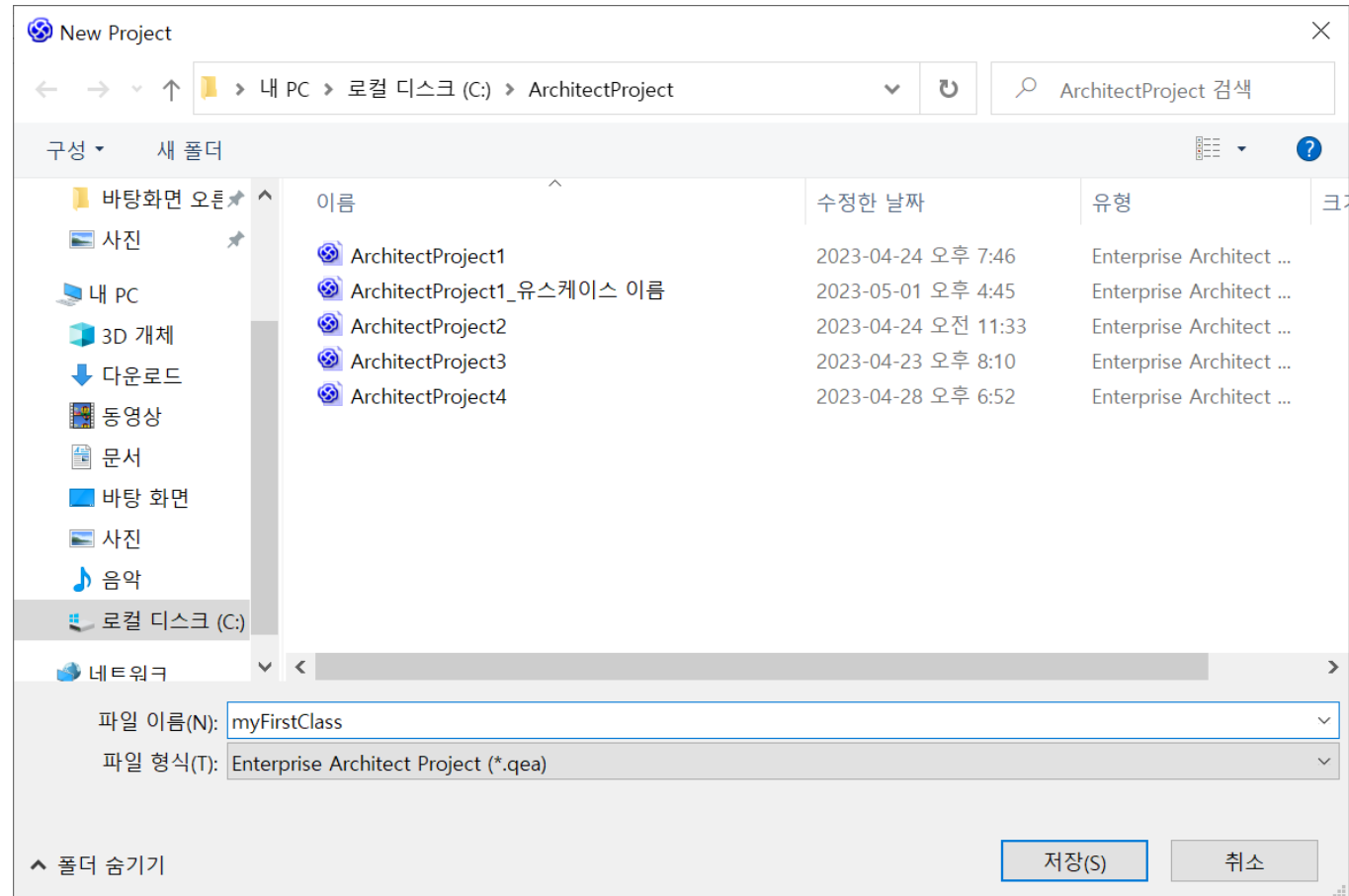
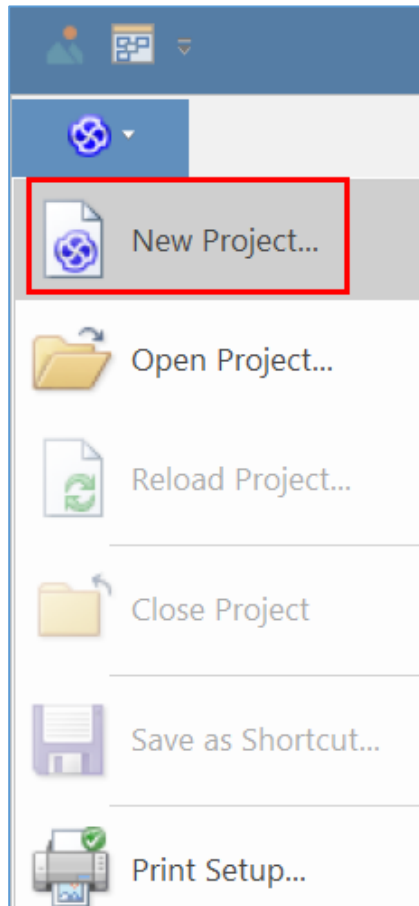
클래스 다이어그램

■ Enterprise Architect 실행



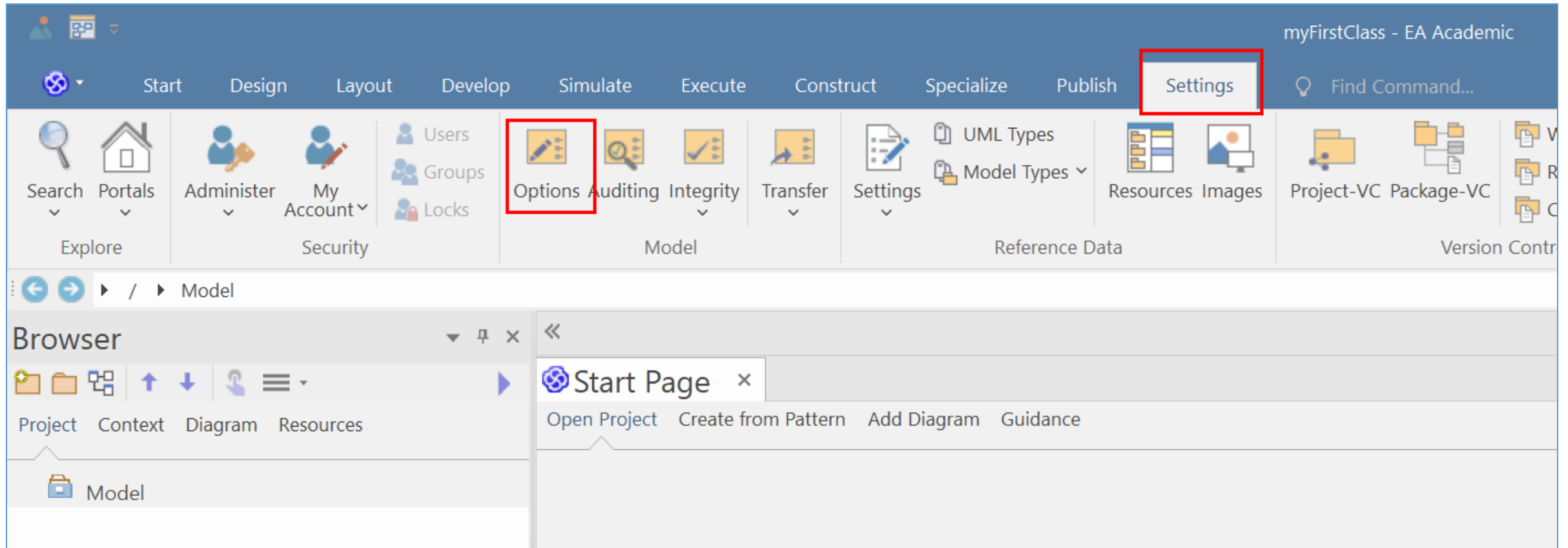
클래스 다이어그램

- 클래스 다이어그램 생성, [옵션] - [Nee Project], 프로젝트 이름 지정(myFirstClass)



클래스 다이어그램

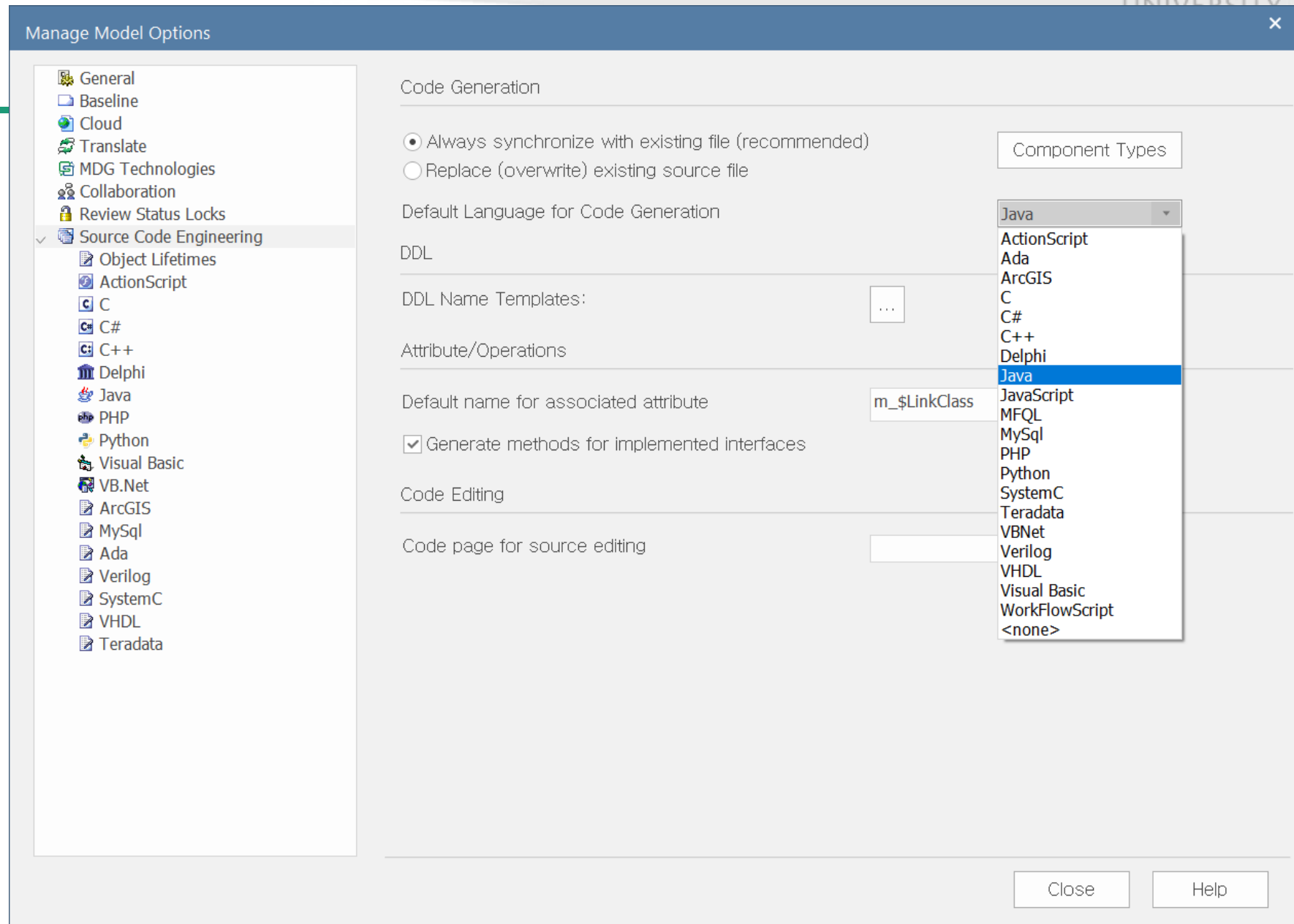
- 상단의 Model 클릭 후 [Settings] - [Options]



클래스 다이어그램

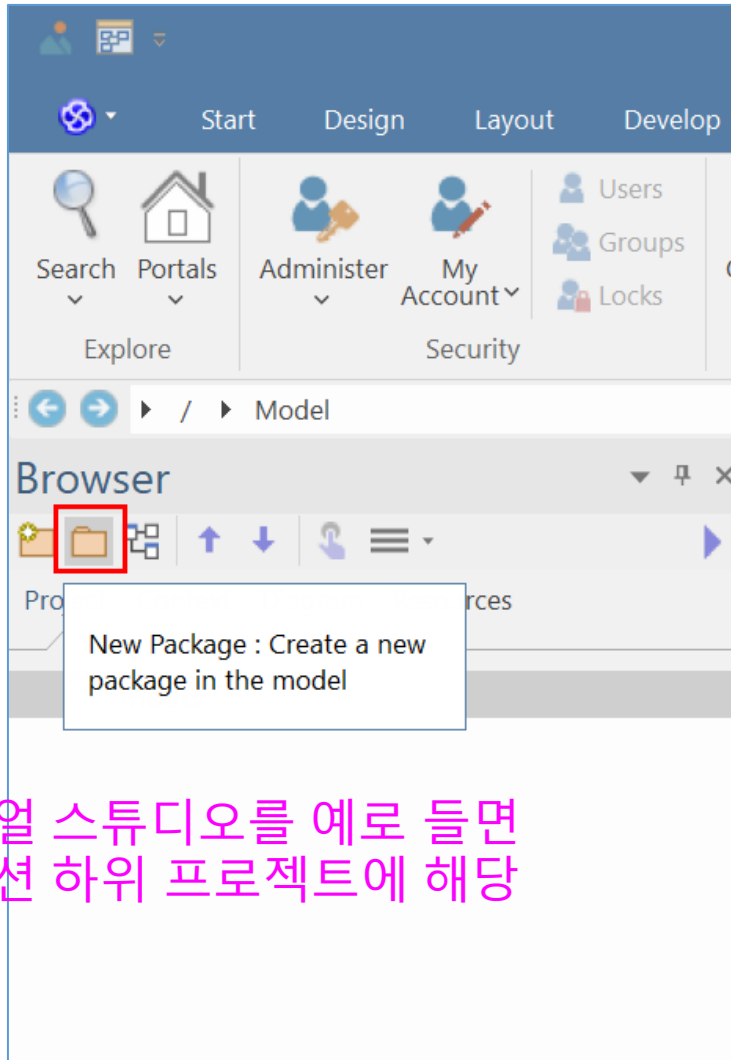
■ 우측에서 지원되는 언어 확인

- 언어 변경 가능
- 우리는 디폴트 Java
- 추후 자동 완성 등을 지원
- Close

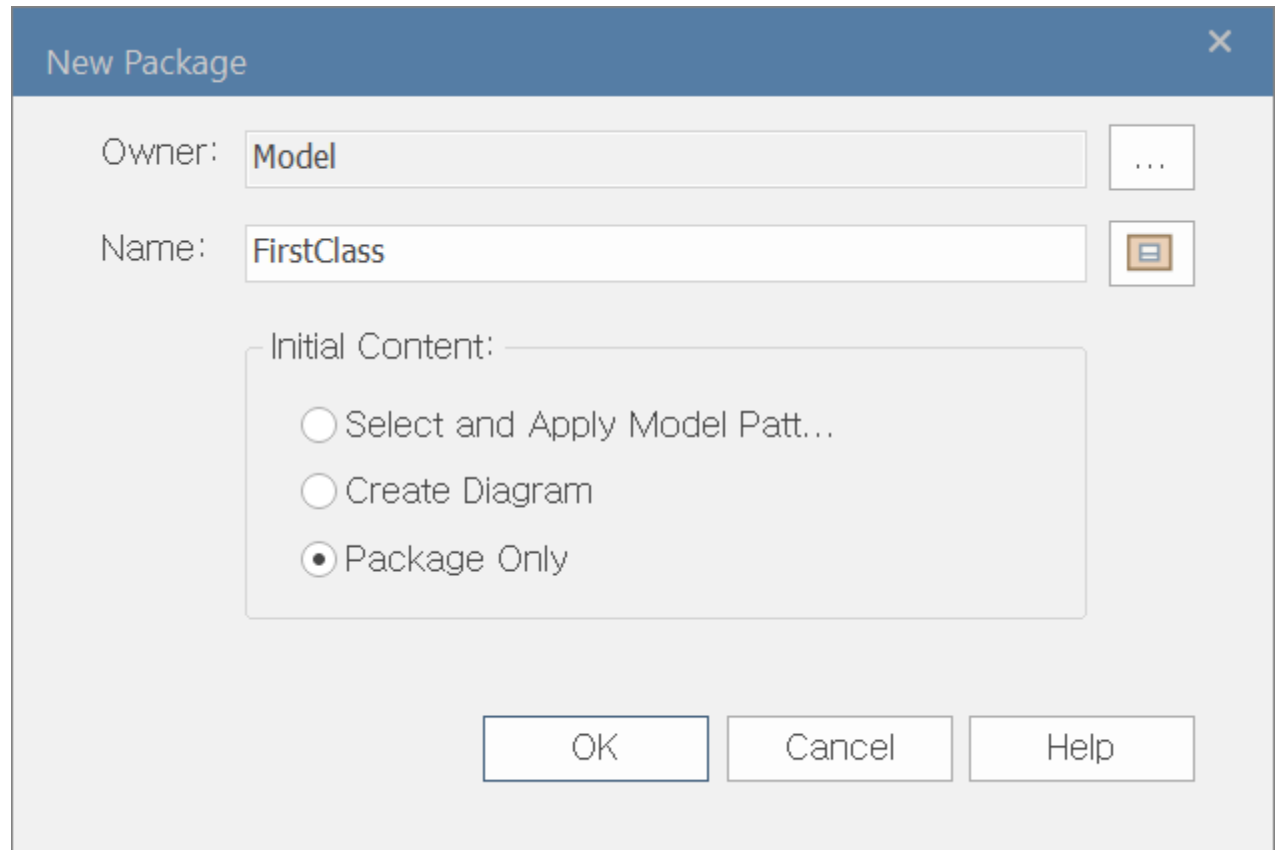


클래스 다이어그램

- 클래스 다이어그램 생성, New Package, Name 입력 후 OK(Package Only)

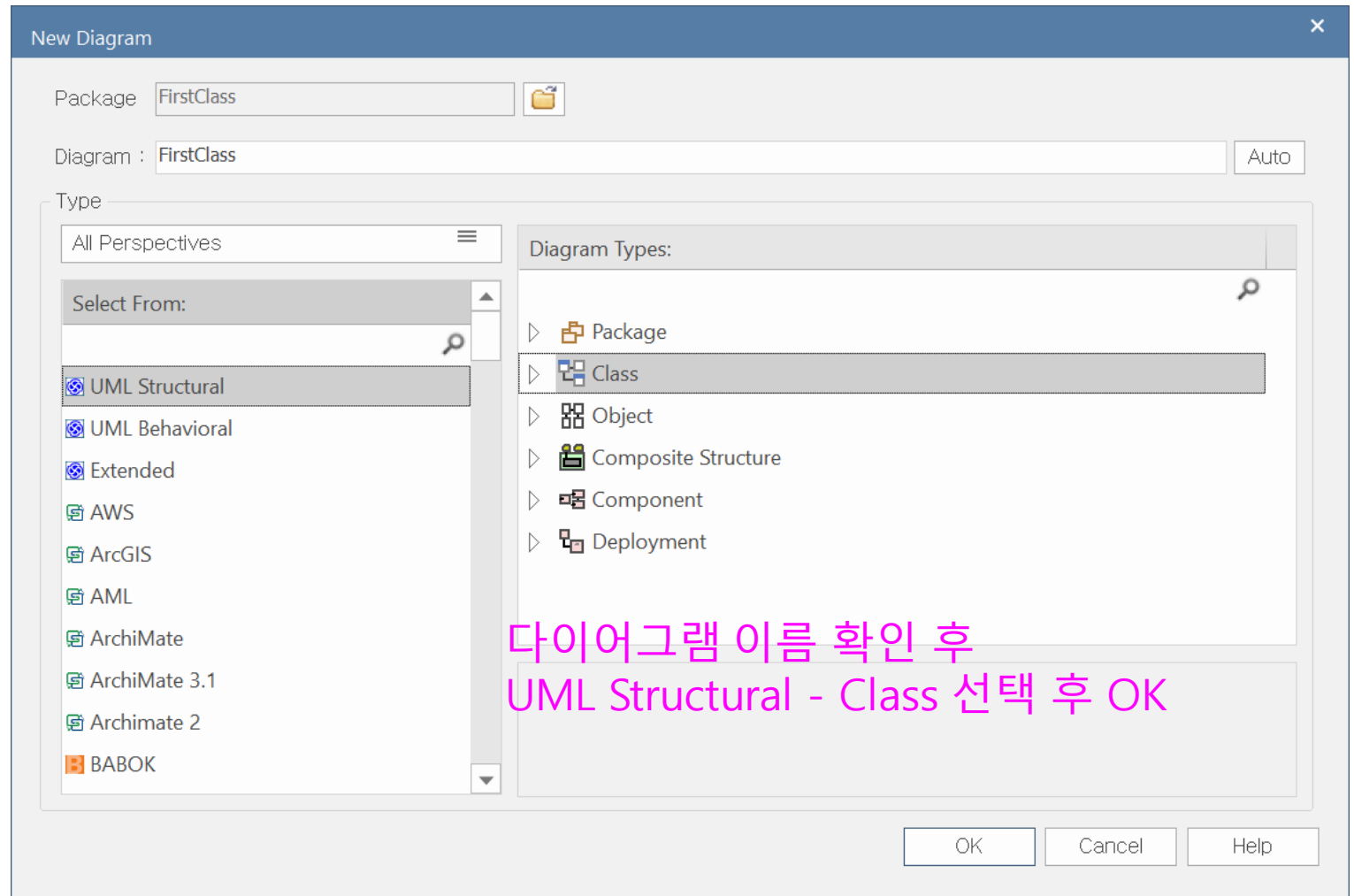
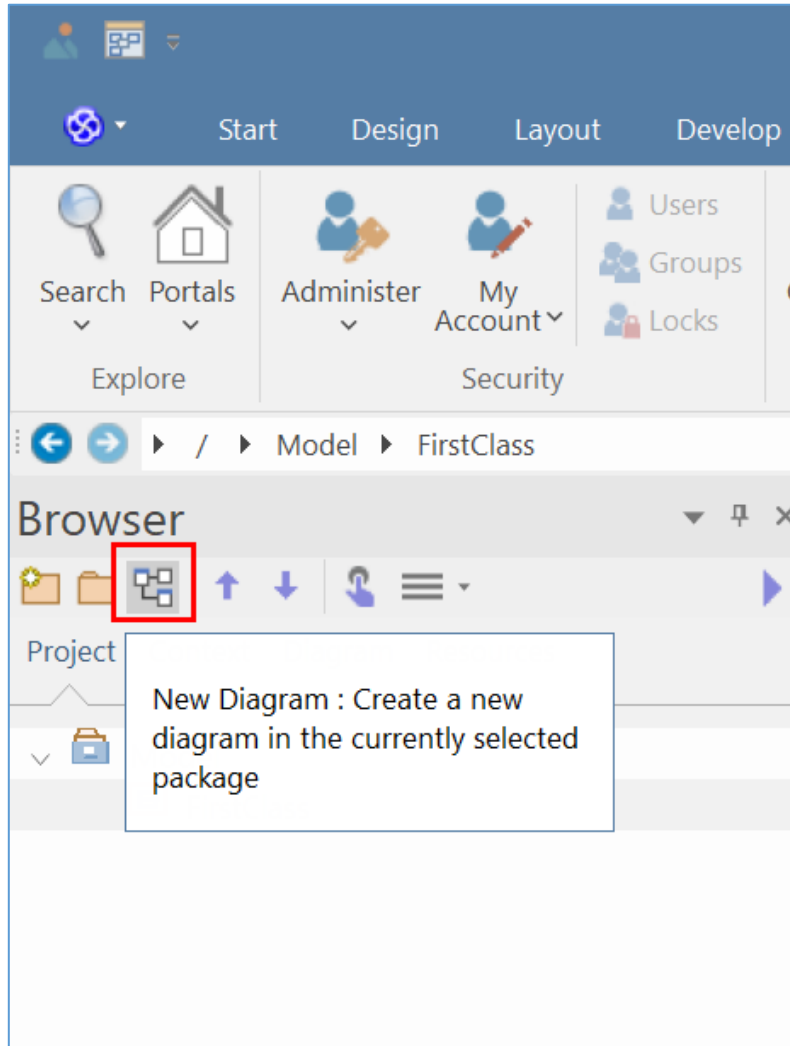


비주얼 스튜디오를 예로 들면
솔루션 하위 프로젝트에 해당



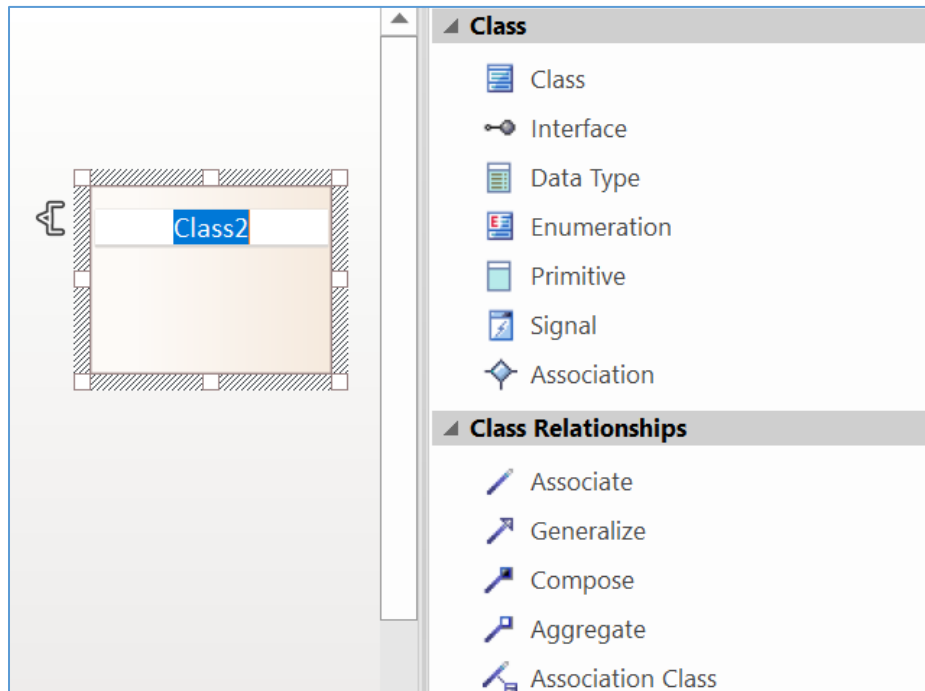
클래스 다이어그램

■ 다이어그램 생성, New Diagram



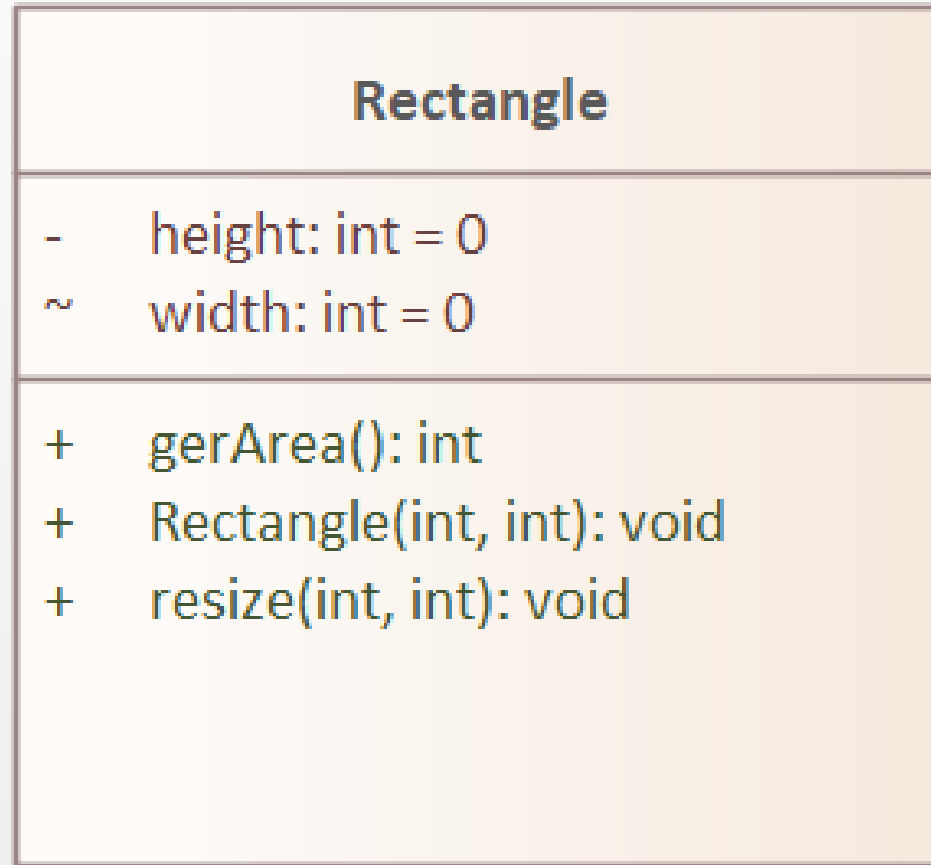
클래스 다이어그램

- 툴박스에서 Class 드래그앤드롭, 이름 입력



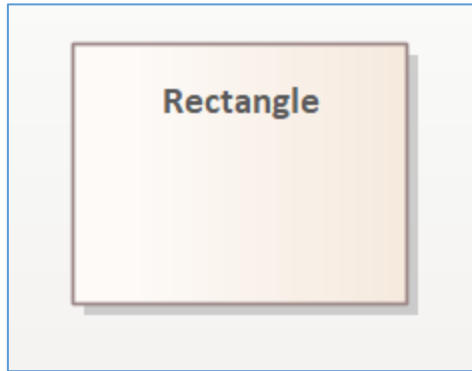
클래스 다이어그램

- 실습 결과



클래스 다이어그램

- 클래스를 두 번 클릭하여 이름 변경 가능



Class : Rectangle

Properties

- General
- Responsibilities
 - Requirements
 - Constraints
 - Scenarios
- Files
- Related
 - Links

Name: Rectangle

Format: B I U A [List] [X1] [X2] [Image] [File]

Stereotype: [Dropdown]

Status: Proposed

Alias: [Text]

Keywords: [Text]

Author: Soyul

Complexity: Easy

Language: Java

Version: 1.0

Phase: 1.0

Package: FirstClass

Created: 2023-05-08 오전 11:41:03

Modified: 2023-05-08 오전 11:44:08

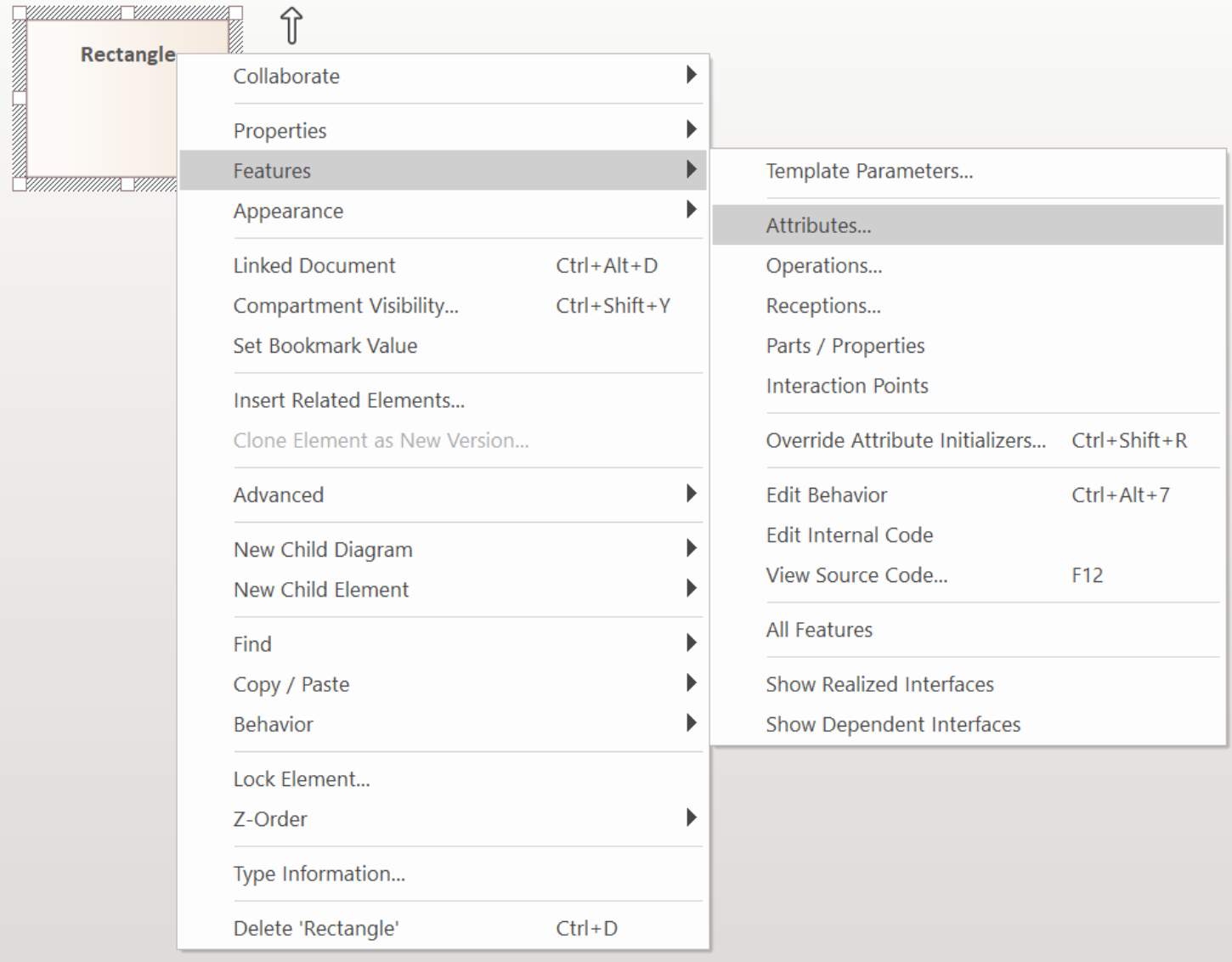
Main Details Advanced Tags

확인 취소 적용(A) 도움말

클래스 다이어그램

■ 멤버 변수 추가

클래스에서 마우스 우클릭 후



클래스 다이어그램

- 하단 Features 창에서 멤버 변수 이름, 멤버 변수 타입, 접근지정자 설정

The screenshot shows the 'Features' window with the 'Attributes' tab selected. A table lists the member variable 'height' with type 'int' and scope 'Private'. Red arrows point from the 'int' and 'Private' cells to their respective dropdown menus. The type dropdown shows 'int' selected, and the scope dropdown shows 'Private' selected. A legend box explains the scope access specifiers.

Name	Type	Scope
height	int	Private

Rectangle
- height: int

Type Selection List:
int
boolean
byte
char
double
float
long
short
<none>
Select Type...



Scope Selection List:
Private
Public
Protected
Private
Package

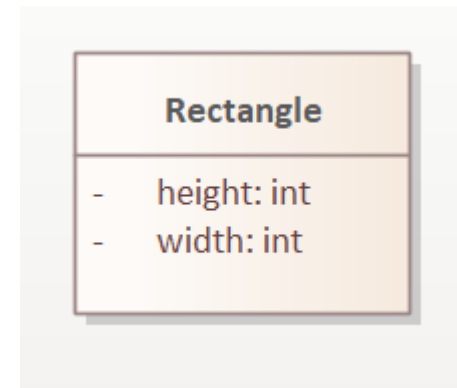
- public(+) : 어떤 클래스에서라도 접근 가능
- private(-) : 해당 클래스 내부에서만 접근 가능
- protected(#) : 동일 패키지 내 클래스 또는 해당 클래스를 상속받은 외부 패키지 클래스에서 접근 가능
- package(~) : 동일 패키지 내부에 있는 클래스에서만 접근 가능

클래스 다이어그램

▪ width 멤버 변수 추가

개발 팀원 등에서 나는 어떻게 클래스를 구성하겠다 라는 것을 시각적으로 이해, 어떤 멤버 변수와 함수가 있는지 설명 가능
이와 같이 틀을 만들고 세부 개발을 지시할 수도 있음

Features		
Attributes Operations Receptions Parts / Properties Interaction Points		
Name	Type	Scope
 height	int	Private
 width	int	Private



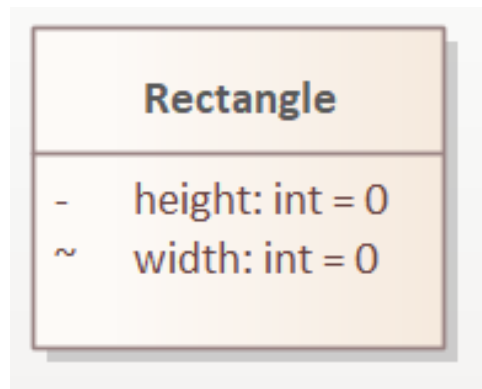
클래스 다이어그램

- Initial Value에서 초기값 지정 가능

Features

Attributes Operations Receptions Parts / Properties Interaction Points

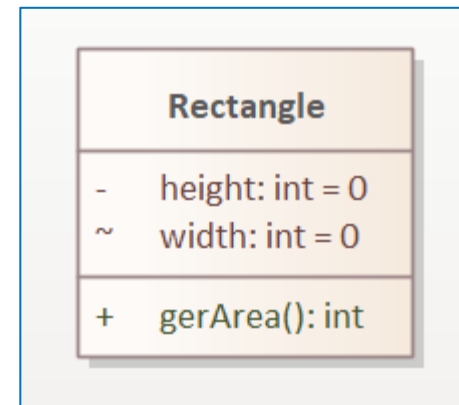
Name	Type	Scope	▲	Stereotype	Alias	Initial Value
<div>New Attribute...</div>						
<div><div></div>width</div>	int	Package				0
<div><div></div>height</div>	int	Private				0



클래스 다이어그램

- getArea 멤버 함수(매서드) 추가, 반환형 추가

Features			
Attributes Operations Receptions Parts / Properties Interaction Points			
Name	Parameters	Return Type	Scope
◆ gerArea		int	Public



클래스 다이어그램

- resize 멤버 함수(매서드) 추가, 매개변수(파라미터) 추가

Parameter	Type	Direction	Notes
a	int	in	
b	int	in	

New Parameter...

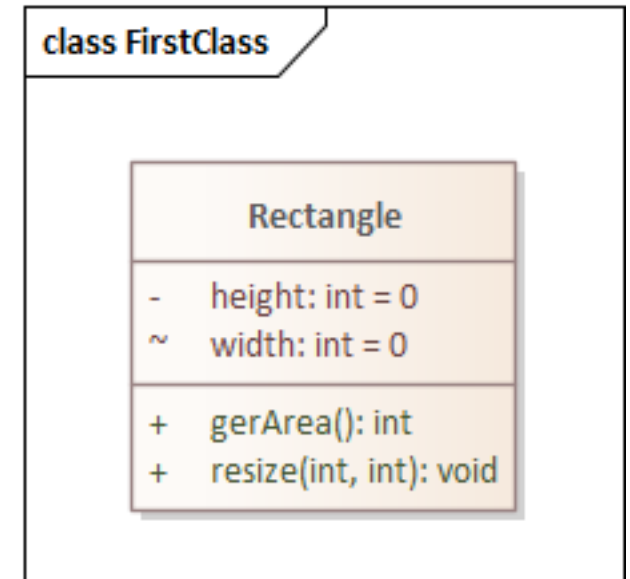
Features

Attributes Operations Reception

Name

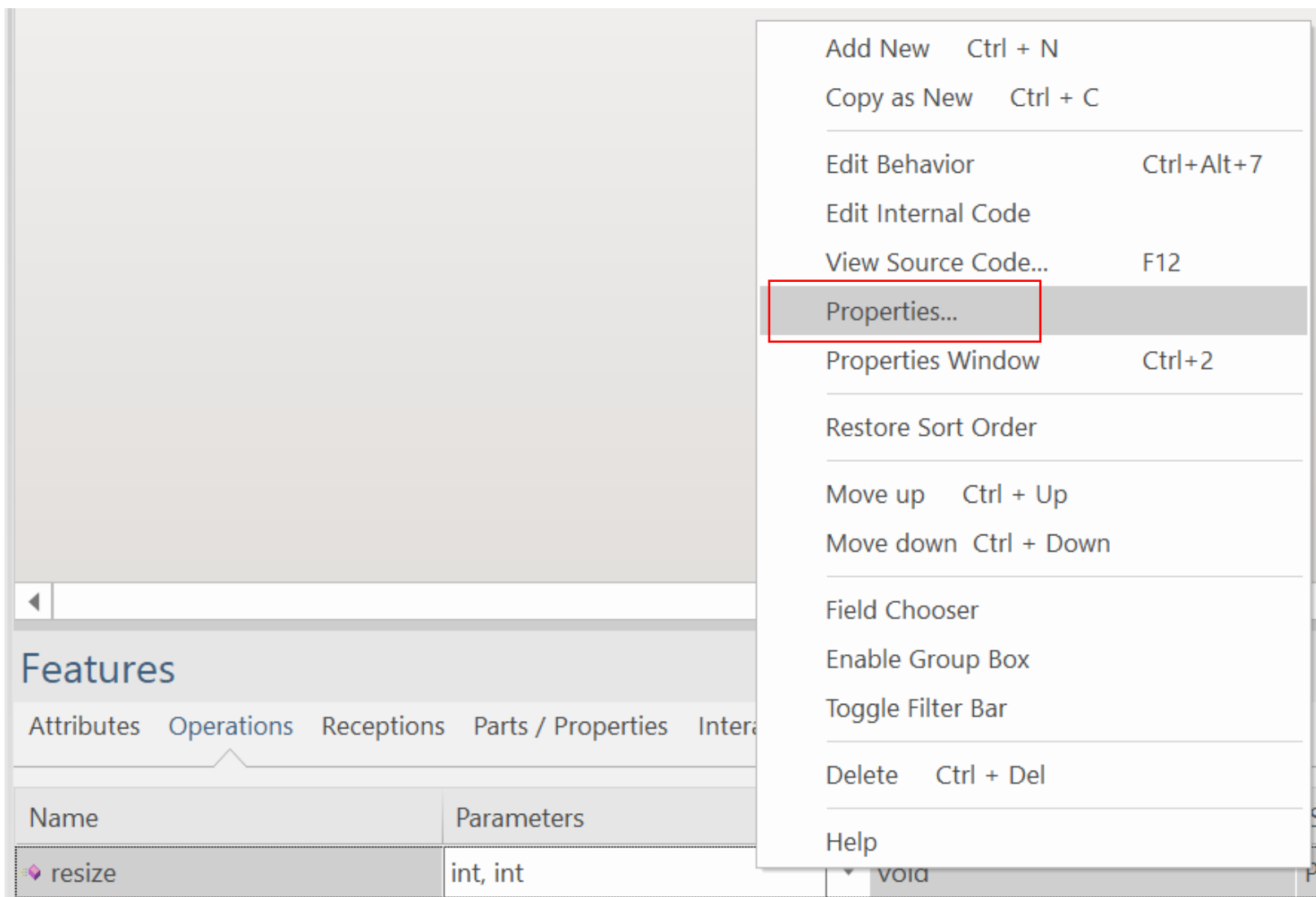
gerArea

resize(int, int): void



클래스 다이어그램

- 함수 내부 코드 작성 가능
 - 생성한 함수를 마우스 오른쪽 클릭 후



클래스 다이어그램

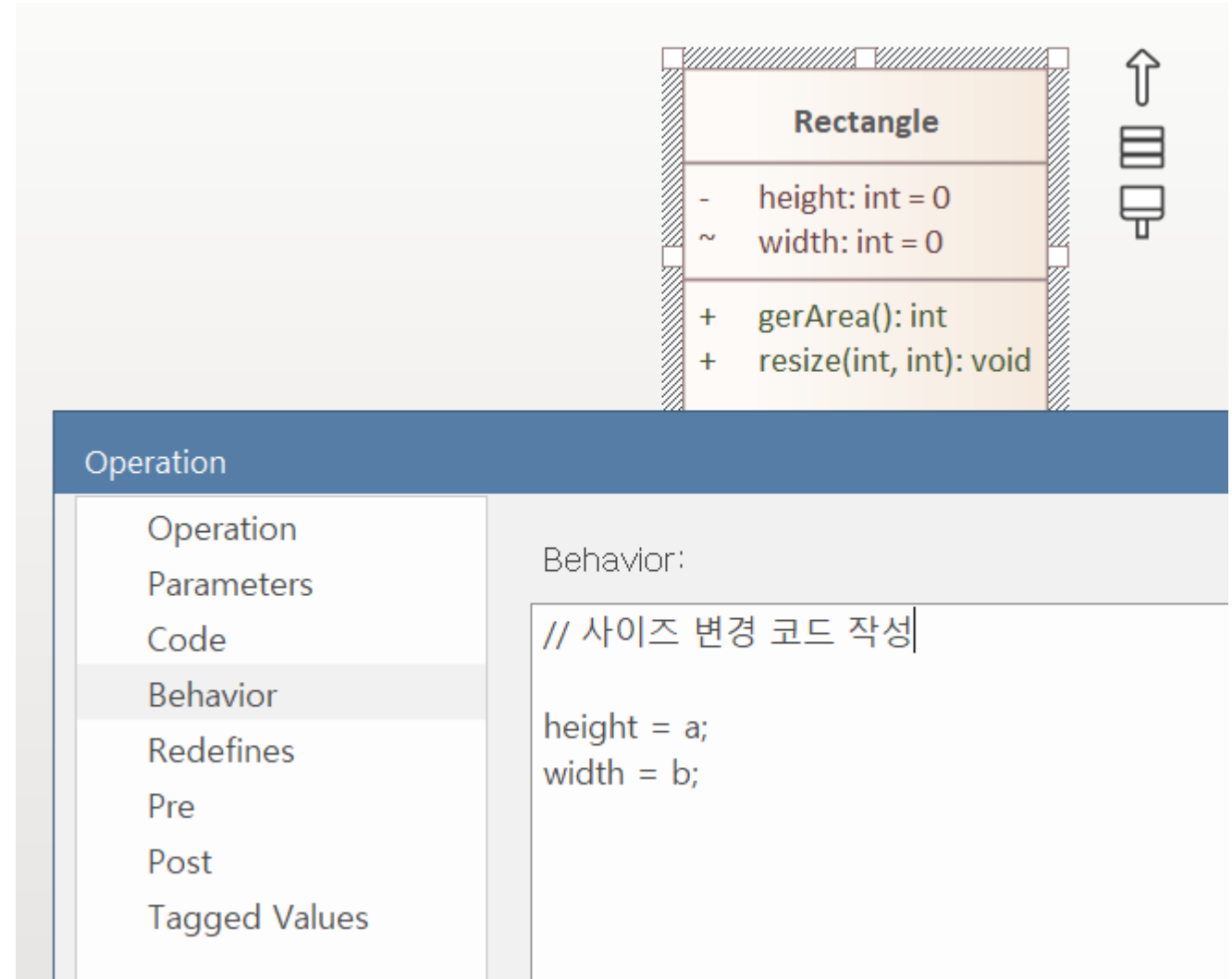
완벽히 코드를 작성할 필요는 없습니다.
이 함수의 기능이 무엇인지 타인에게 이해시키거나 기억할 수 있을 정도

■ 코드 작성 가능

- 초기 일부분의 코드를 직접 작성한다던가
주석 등으로 작성할 코드를 설명 가능

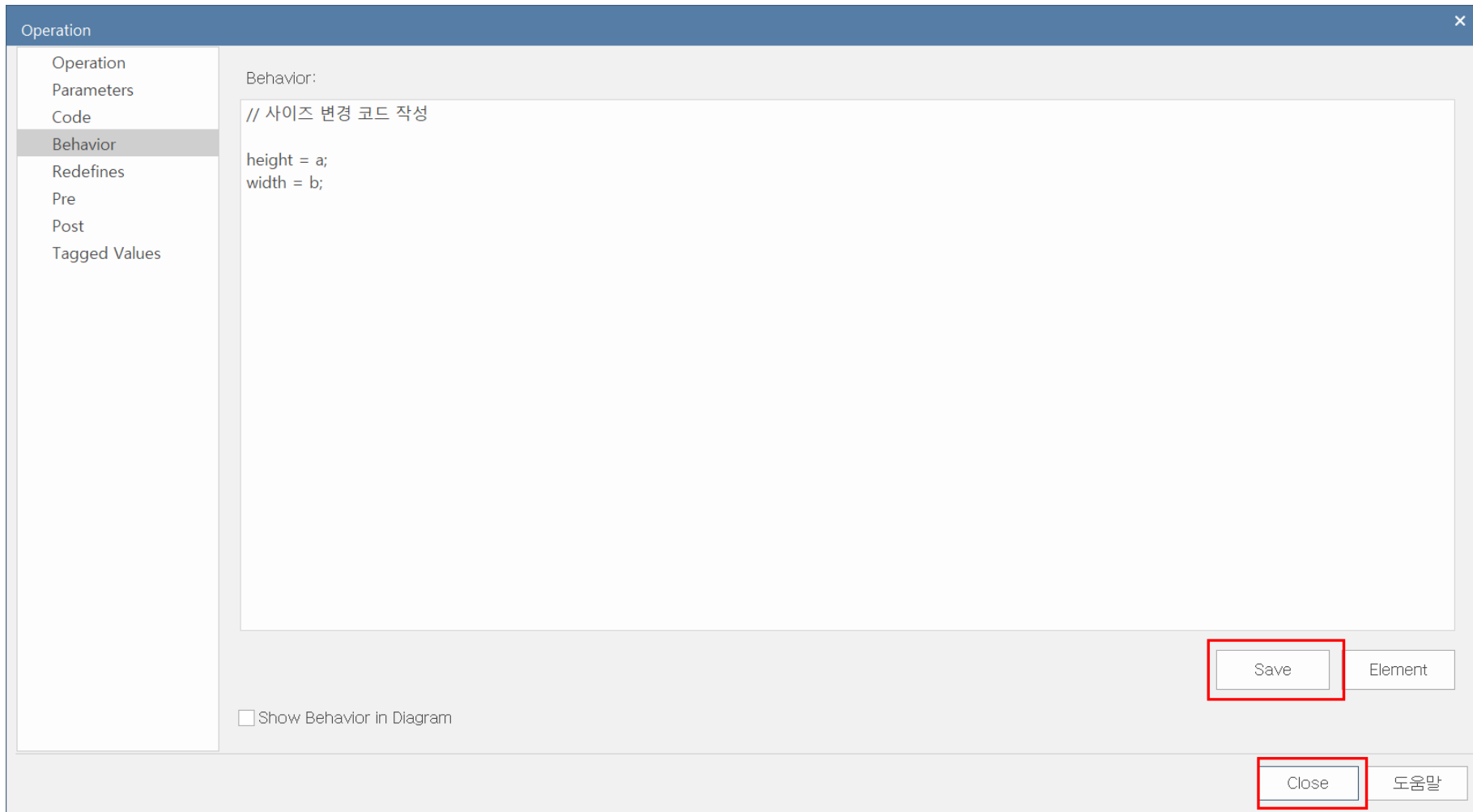
- 파라미터 명이 생각이 안나면 Parameters
에서 확인 가능

Operation		
Operation		
Parameters	Parameter	Type
Code	a	int
Behavior	b	int
Redefines	New Parameter...	
Pre		
Post		



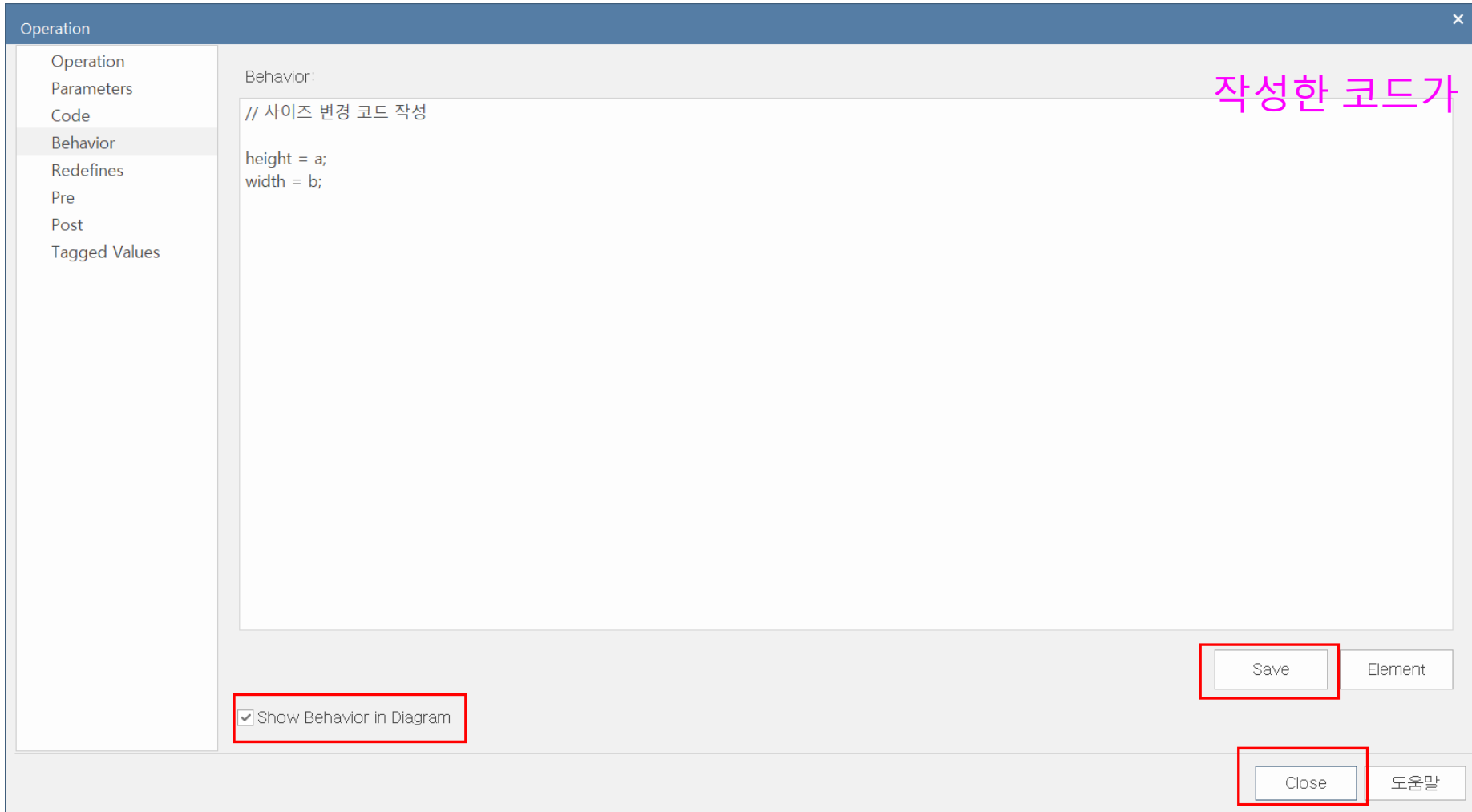
클래스 다이어그램

■ 저장 및 Close

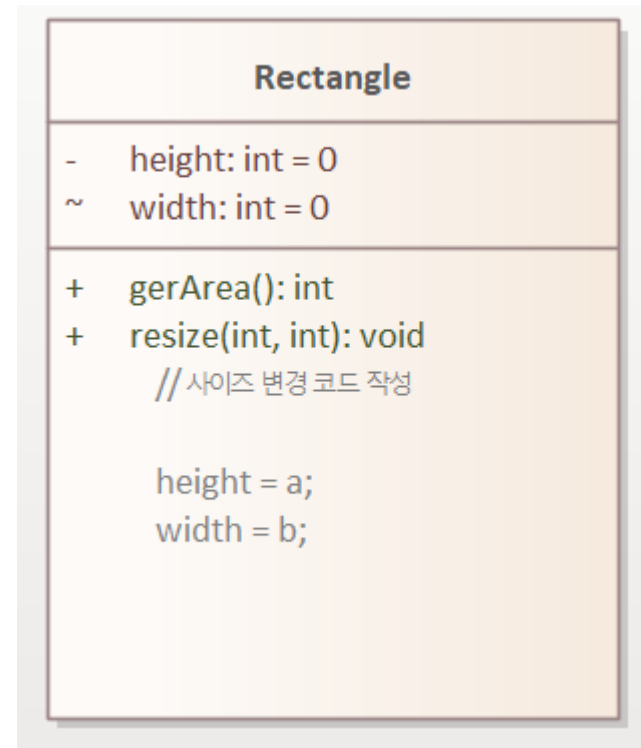


클래스 다이어그램

- (설계 회의 등에서) 만약 코드를 보여주고 싶을 때는 (협업시 활용 가능)



작성한 코드가 클래스 다이어그램에 노출



- | Parameter | Type | Direction | Notes |
|------------------|------|-----------|-------|
| a | int | in | |
| b | int | in | |
| New Parameter... | | | |

Features

Attributes Operations Reception

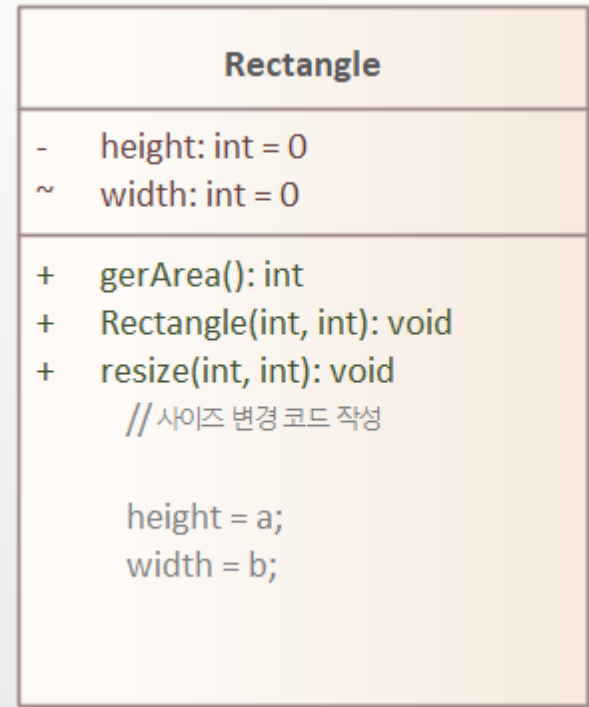
Name

resize

Rectangle

void

Public



해보기 (과제 아님)

- 본인이 원하는 클래스와 멤버 변수 (2개 이상) 및 함수(4개 이상) 작성



 **T h a n k y o u**

TECHNOLOGY

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Velit ex
plicabo ipsum, labore sed tempora ratione asperiores des
cenderat bore sed tempora rati jgert one bore sed tem!