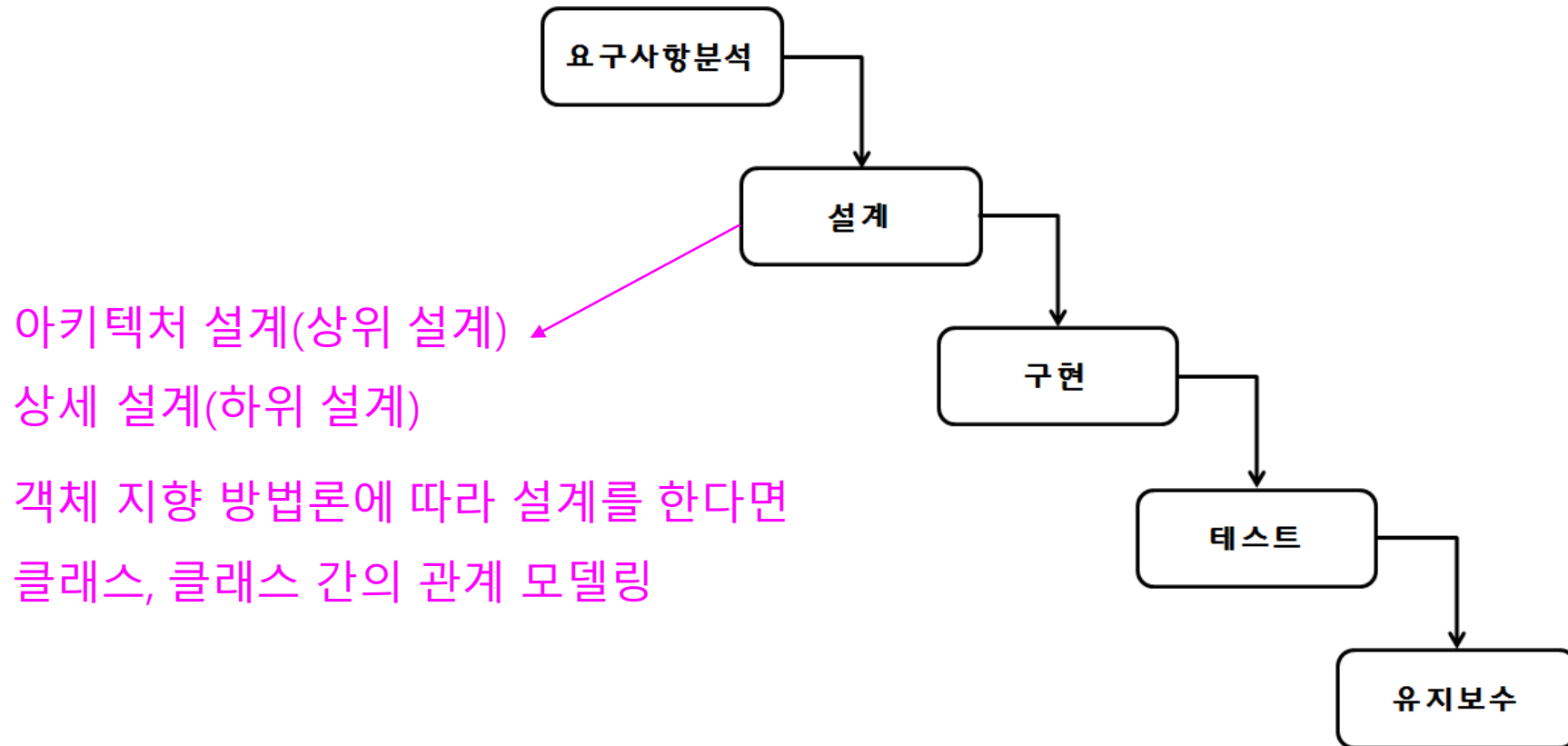


소프트웨어공학

클래스 다이어그램 2



소프트웨어 개발 프로세스



UML(Unified Modeling Language)

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

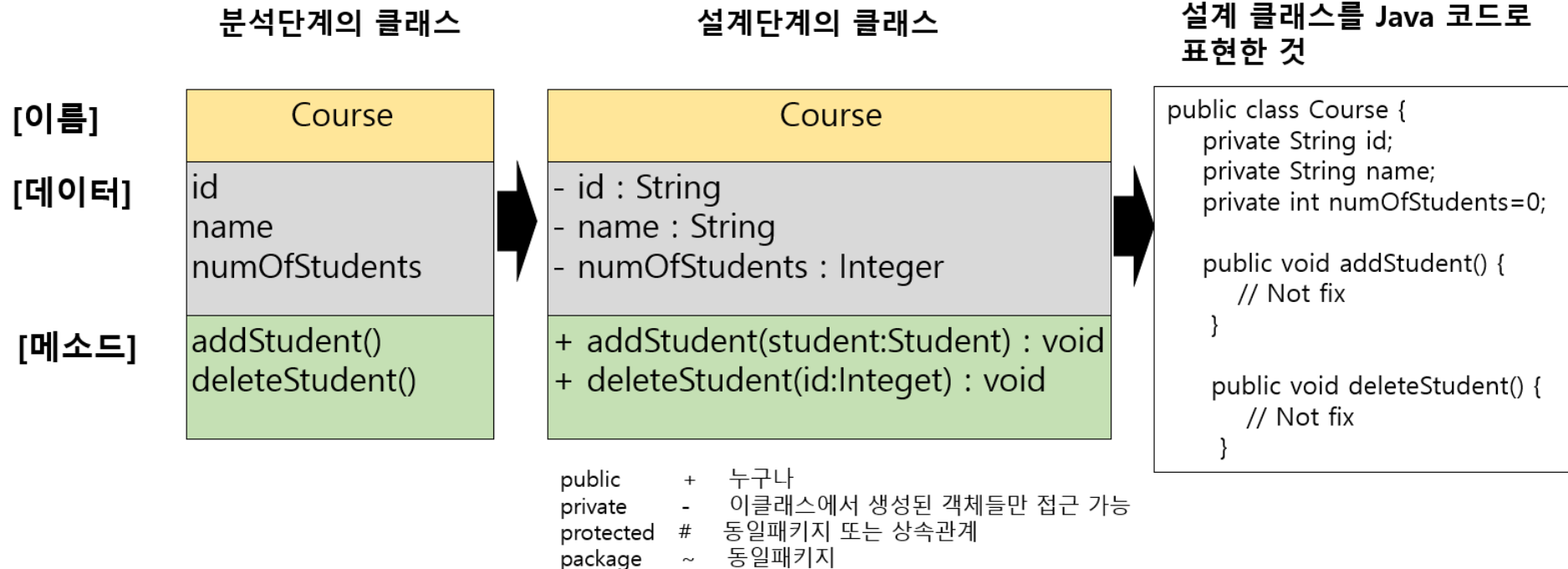
프로그램 제작에서 필요한 클래스 자체와 클래스 간의 관계로 분리하여 모델링

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.

클래스 다이어그램

■ 클래스 다이어그램

- . 클래스와 클래스의 속성과 연산, 객체 사이의 관계를 모델링하기 위한 그래픽 표기법 제공
- . 소프트웨어 구조 표현
- . 가장 중요한 다이어그램



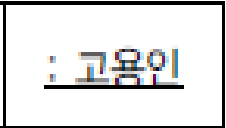
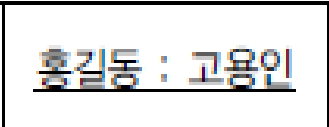
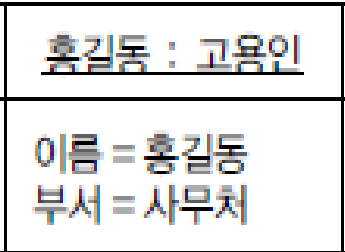
클래스 다이어그램

■ 클래스 표기법 보기

보기	의미
<div>고용인</div>	속성과 연산 정보를 생략한 '고용인' 클래스를 표현한다.
<div>고용인</div> <div>이름: string 부서:</div>	연산 정보를 생략한 클래스 표현이다. '고용인' 클래스는 '이름'과 '부서' 속성을 갖는다. '이름' 속성 타입은 'string' 이고, '부서' 속성에 대한 타입은 아직 결정되지 않았다
<div>고용인</div> <div>이름: string 부서:</div> <div>부서이동() 임금계산(보너스: int) : double</div>	'고용인' 클래스에 대한 속성과 연산 정보를 보인다. '고용인' 클래스는 '부서이동()'과 '임금계산()' 연산을 가지며 '임금계산()'에 대한 매개변수 이름은 '보너스'이고 타입은 'int', 반환 타입은 'double'이다. '부서이동()' 연산에 대한 매개변수는 없거나 아직 결정되지 않았다.

클래스 다이어그램

■ 객체 표기법 보기

보기	의미
	'고용인' 클래스의 특정 인스턴스 객체를 표현한다. 객체 이름은 명세하지 않았다.
	'고용인' 클래스의 '홍길동' 객체를 표현한다.
	'고용인' 클래스의 '홍길동' 객체의 이름은 '홍길동'이고 부서는 '사무처'이다.

클래스 표기법 자바 코드

- UML 클래스/객체 표기에 대한 자바코드


UML 클래스	자바 코드			
<table><tr><td>고용인</td></tr><tr><td>- 이름: string - 부서: string</td></tr><tr><td>+ 임금계산(보너스: int) : double</td></tr></table>	고용인	- 이름: string - 부서: string	+ 임금계산(보너스: int) : double	<pre>class Employer { private String name; private String dept; public double computeSalary(int bonus){ } }</pre>
고용인				
- 이름: string - 부서: string				
+ 임금계산(보너스: int) : double				
<table><tr><td><u>홍길동 : 고용인</u></td></tr><tr><td>이름 = 홍길동 부서 = 사무처</td></tr></table>	<u>홍길동 : 고용인</u>	이름 = 홍길동 부서 = 사무처	<pre>void main() { Employer 홍길동 = new Employer("홍길동", "사무처"); }</pre>	
<u>홍길동 : 고용인</u>				
이름 = 홍길동 부서 = 사무처				

일반화 관계와 실현(실체화) 관계

- 일반화 관계와 실현 관계

- 일반화 관계 : 클래스 사이의 일반화(상속) 관계를 표현하면 '~이다(is-a)'로 해석

- ※ 서브(하위)클래스는 슈퍼(상위)클래스의 한 종류이다.

표기법  자바의 extends 키워드로 구현

- 실현(실체화) 관계 : 특정 클래스의 명세를 실현(구현)하는 관계를 표현한다.

- ※ 구현하는 클래스와 인터페이스 관계를 말함

표기법  자바의 implements 키워드로 구현

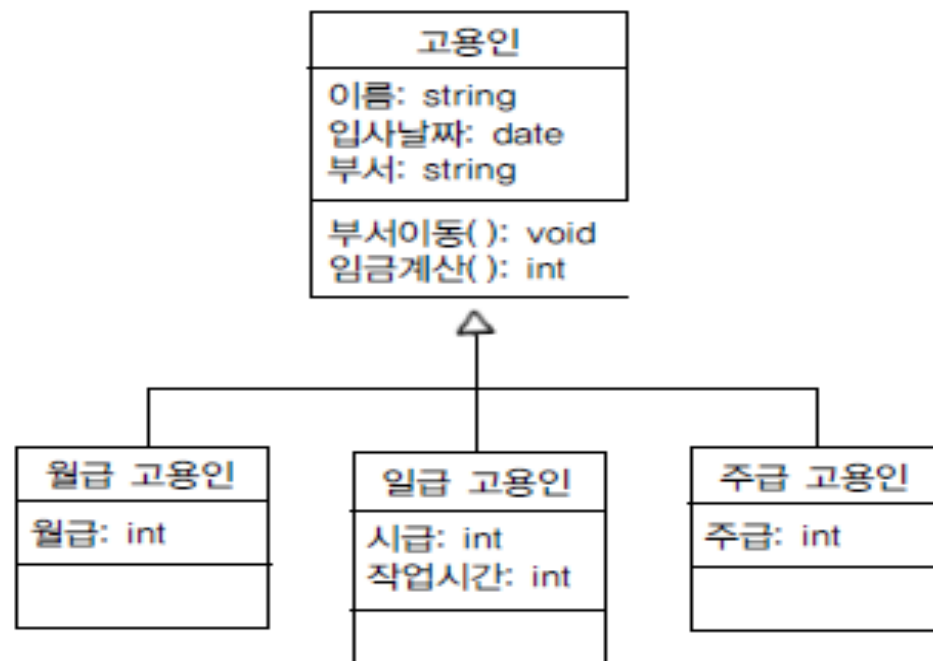
일반화 관계 자바코드

- '고용인' 클래스의 일반화 관계 및 자바코드

표기법



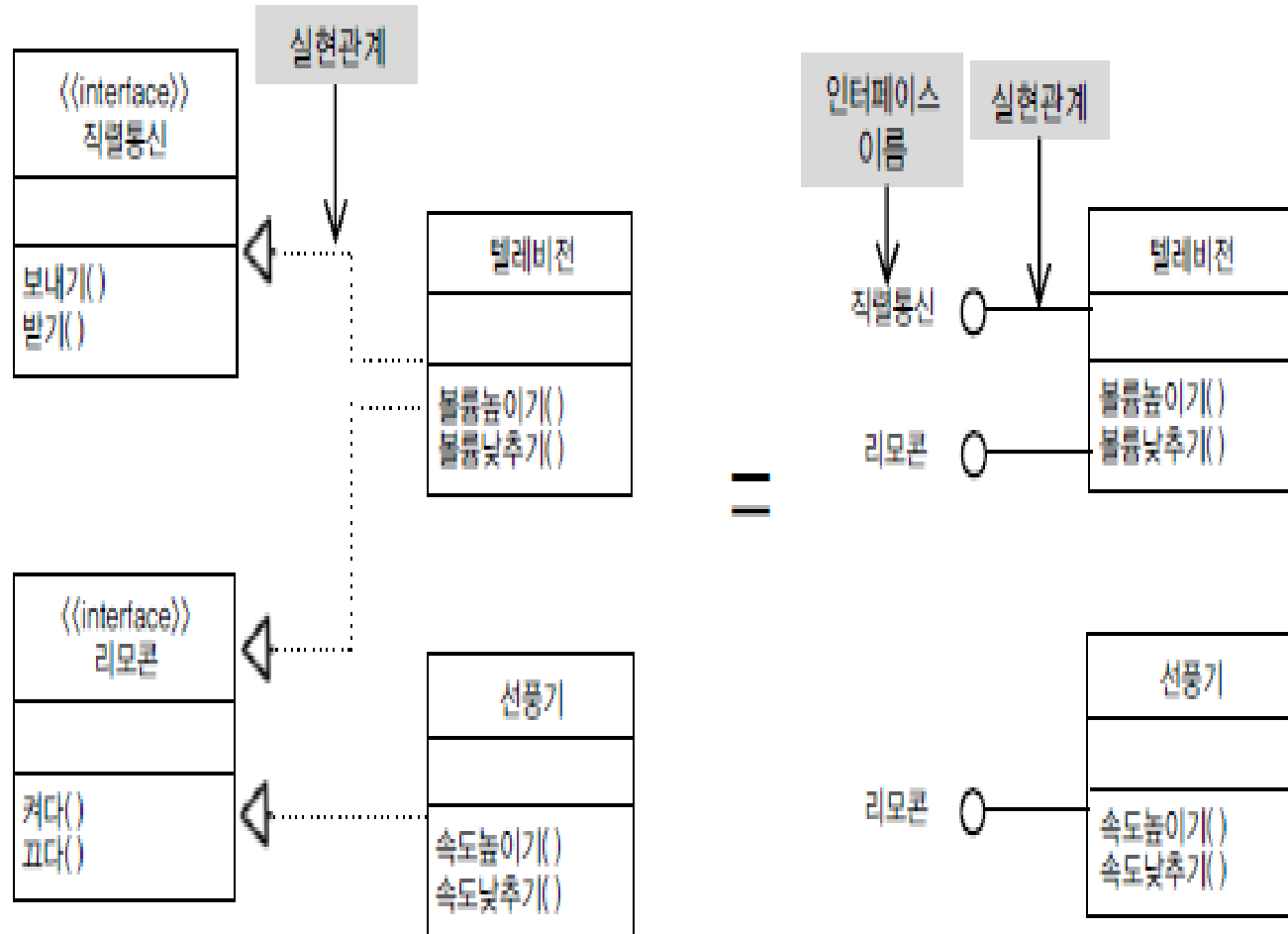
자바의 extends 키워드로 구현



```
class 고용인 {  
    String 이름;  
    Date 입사날짜;  
    String 부서;  
    void 부서이동( );  
    int 임금계산( );  
}
```

```
class 월급고용인 extends 고용인 {  
    int 월급;  
}  
class 일급고용인 extends 고용인 {  
    int 시급;  
    int 작업시간;  
}  
class 주급고용인 extends 고용인 {  
    int 주급;  
}
```

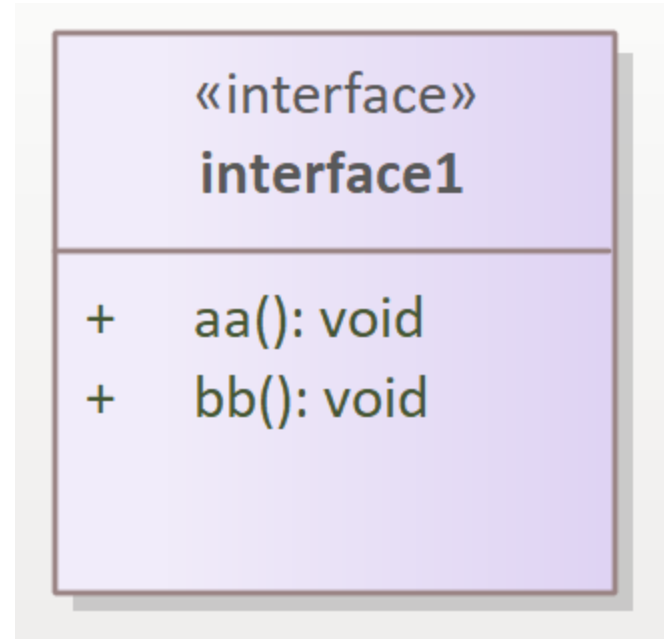
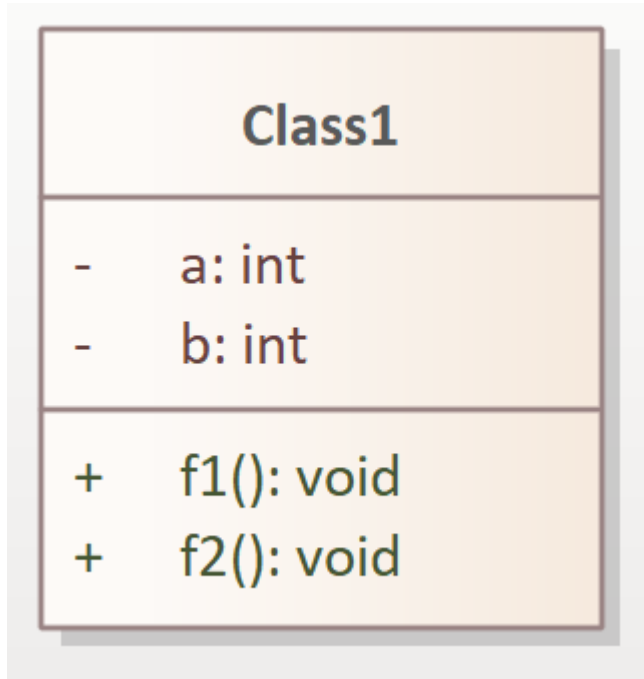
실현(실체화) 관계 구현하는 클래스와 인터페이스 관계를 말함



직렬통신	리모콘
<pre> public interface SerialCom { void send(); void receive(); } </pre>	<pre> public interface RemoteControl { void turnOn(); void turnOff(); } </pre>
텔레비전	선풍기
<pre> public class TV implements SerialCom, RemoteControl { public void send() {.....}; public void receive() {.....}; public void turnOn() {.....}; public void turnOff() {.....}; public void volumeUp() {.....}; public void volumeDown() {.....}; } </pre>	<pre> public class Fan implements RemoteControl { public void turnOn() {.....}; public void turnOff() {.....}; public void speedUp() {.....}; public void speedDown() {.....}; } </pre>

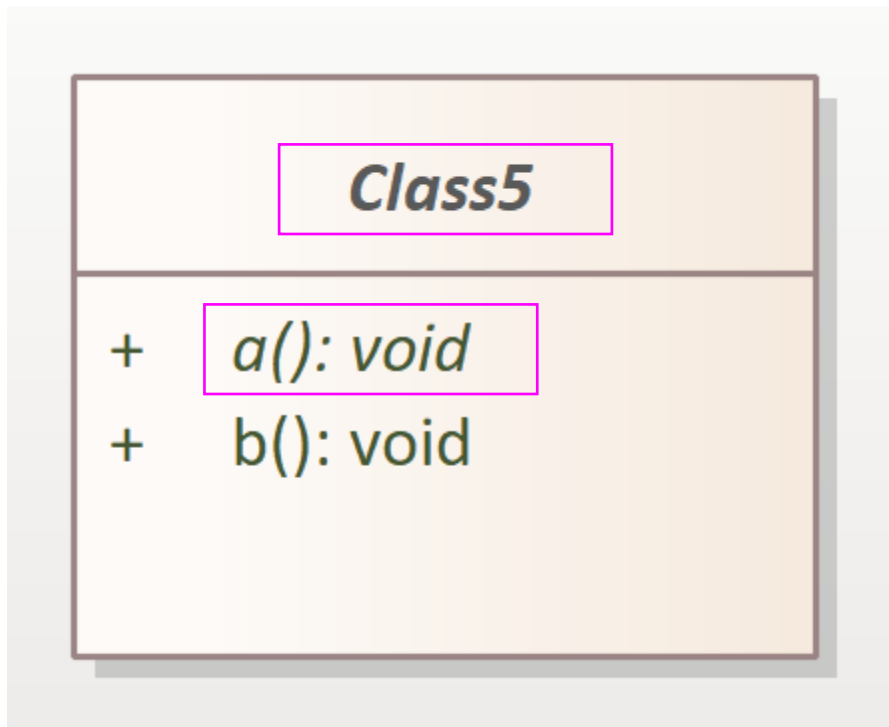
실습

- 다음 클래스와 인터페이스를 작성해 보세요.



클래스 다이어그램에서 추상 메소드와 추상 클래스의 표현

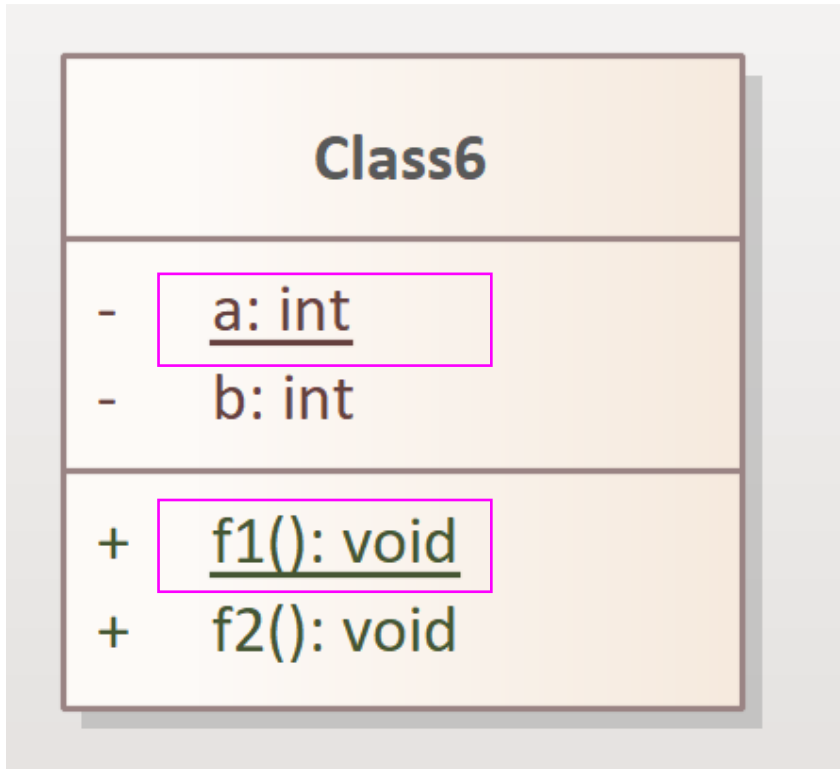
- 추상 메소드/추상 클래스
 - 기울어진 이텔릭체로 표현



```
public abstract class Class5 {  
    public abstract void a();  
    public void b(){  
        }  
}
```

클래스 다이어그램에서 static 멤버의 표현

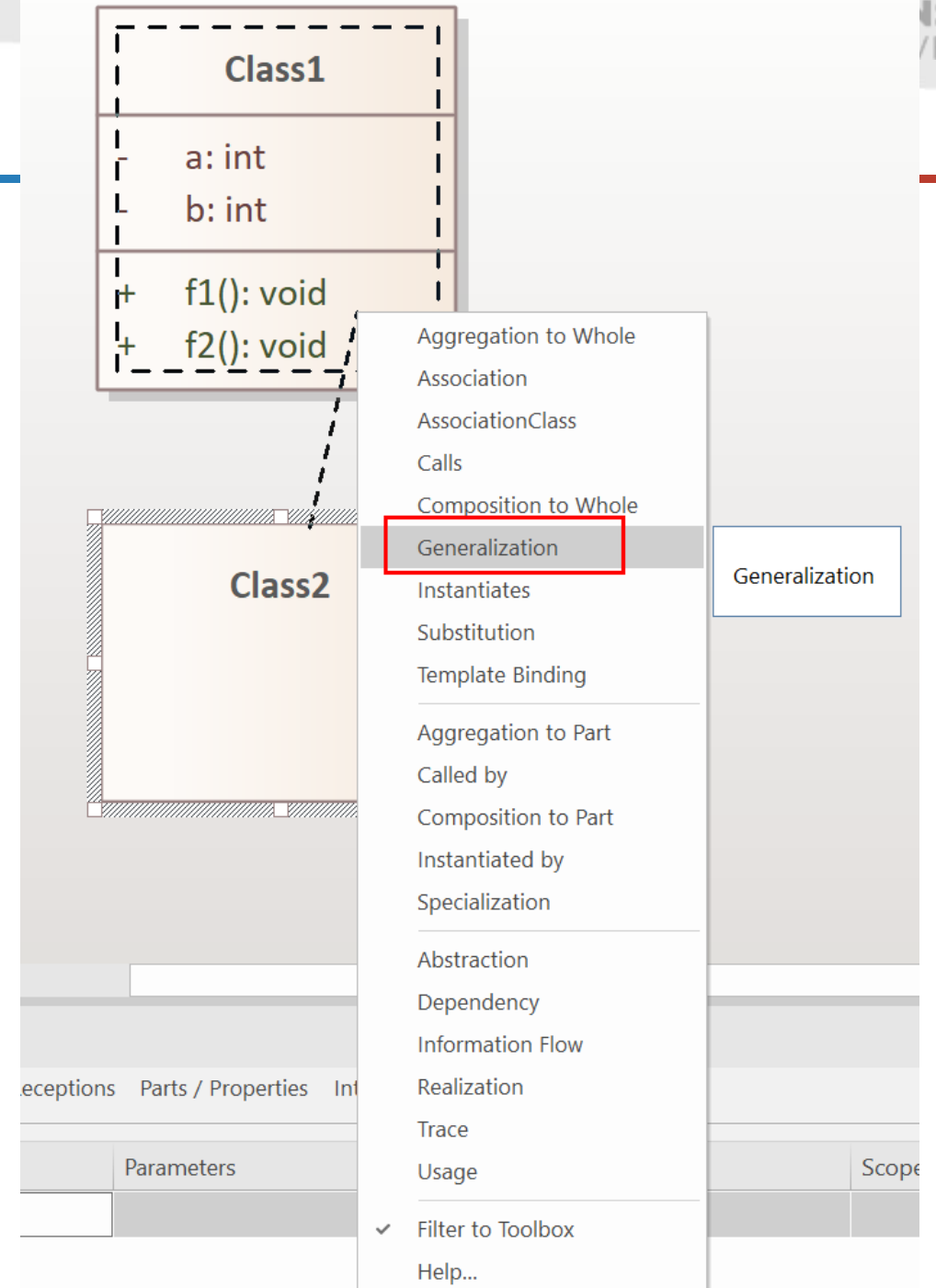
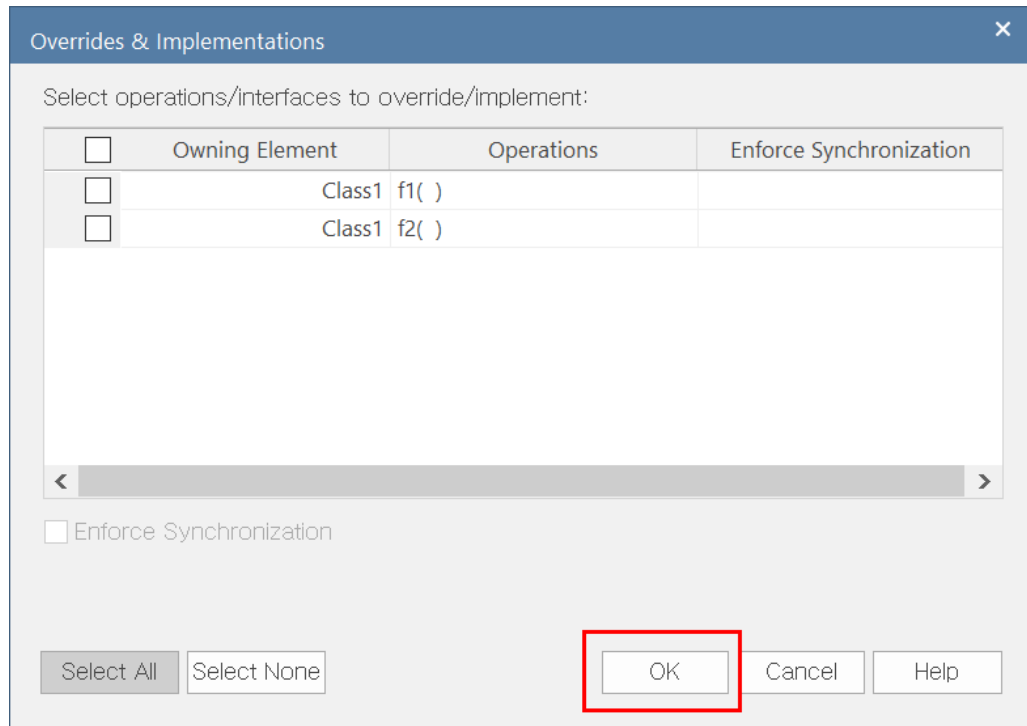
- static 멤버 생성 - 밑줄로 표현



```
public class Class6 {  
  
    private static int a;  
    private int b;  
  
    public static void f1(){  
  
    }  
  
    public void f2(){  
  
    }  
  
}
```

실습

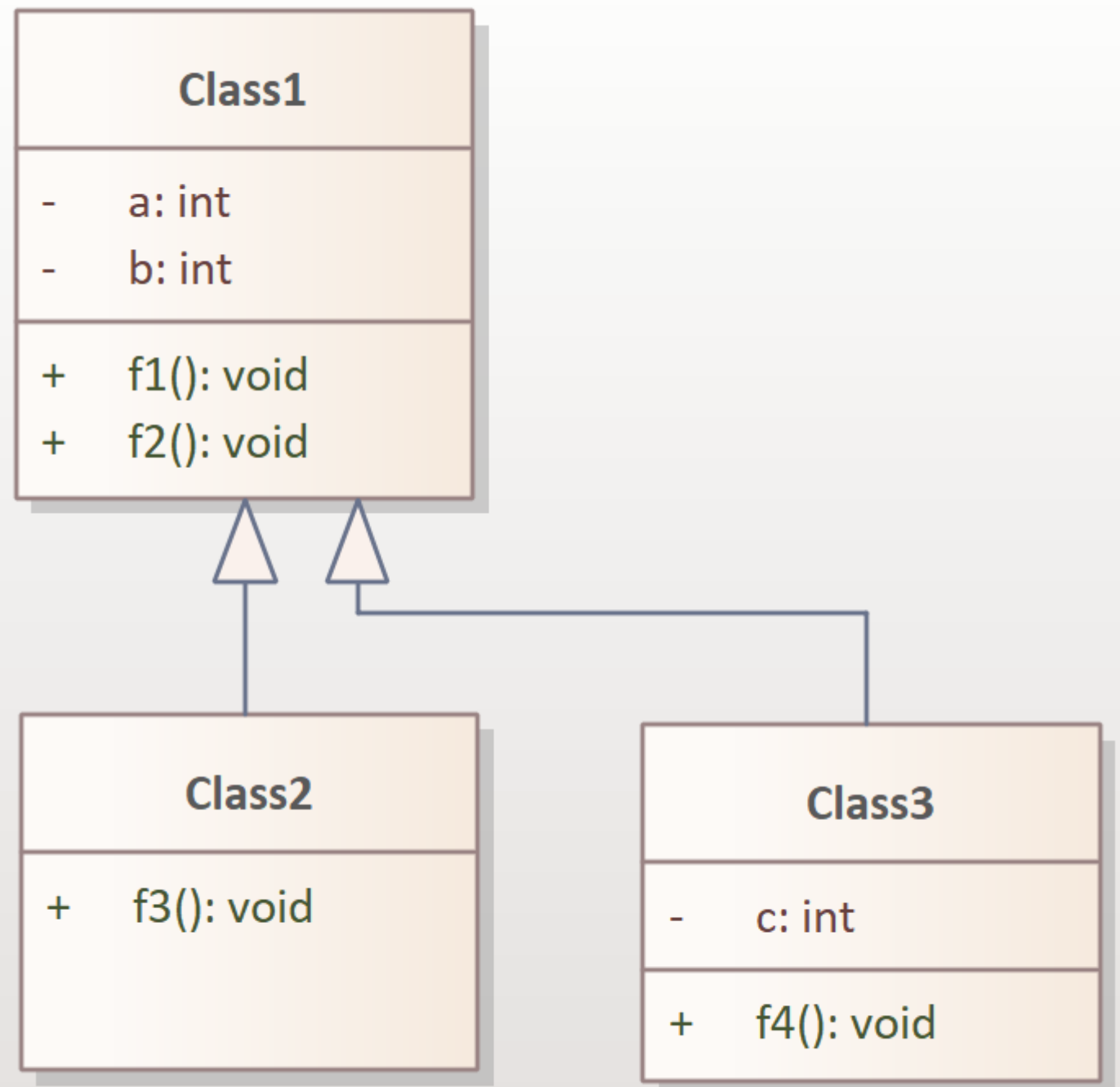
- 다음 클래스를 상속받는 클래스 작성
 - 설정 방법은 유스케이스 다이어그램과 유사



실습

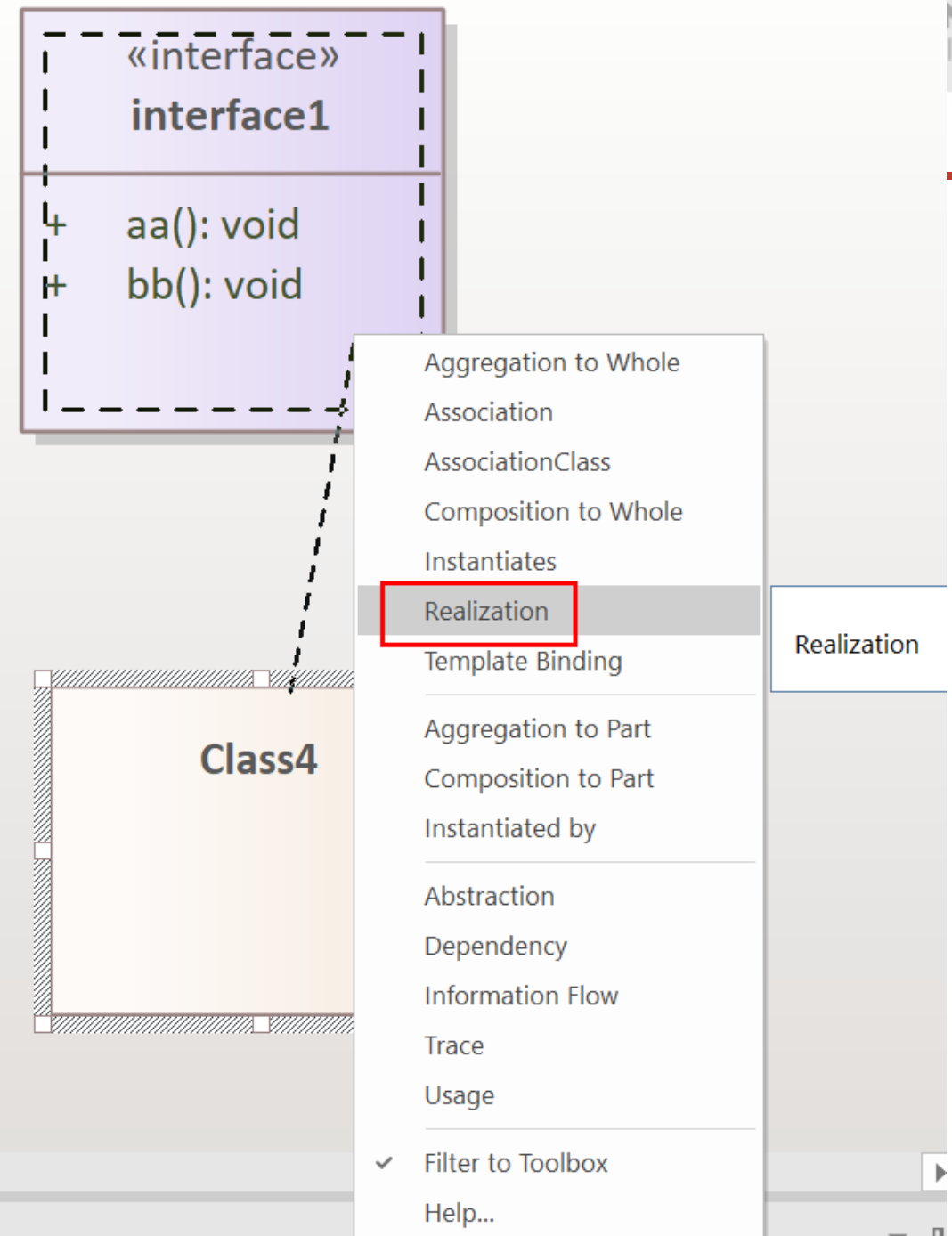
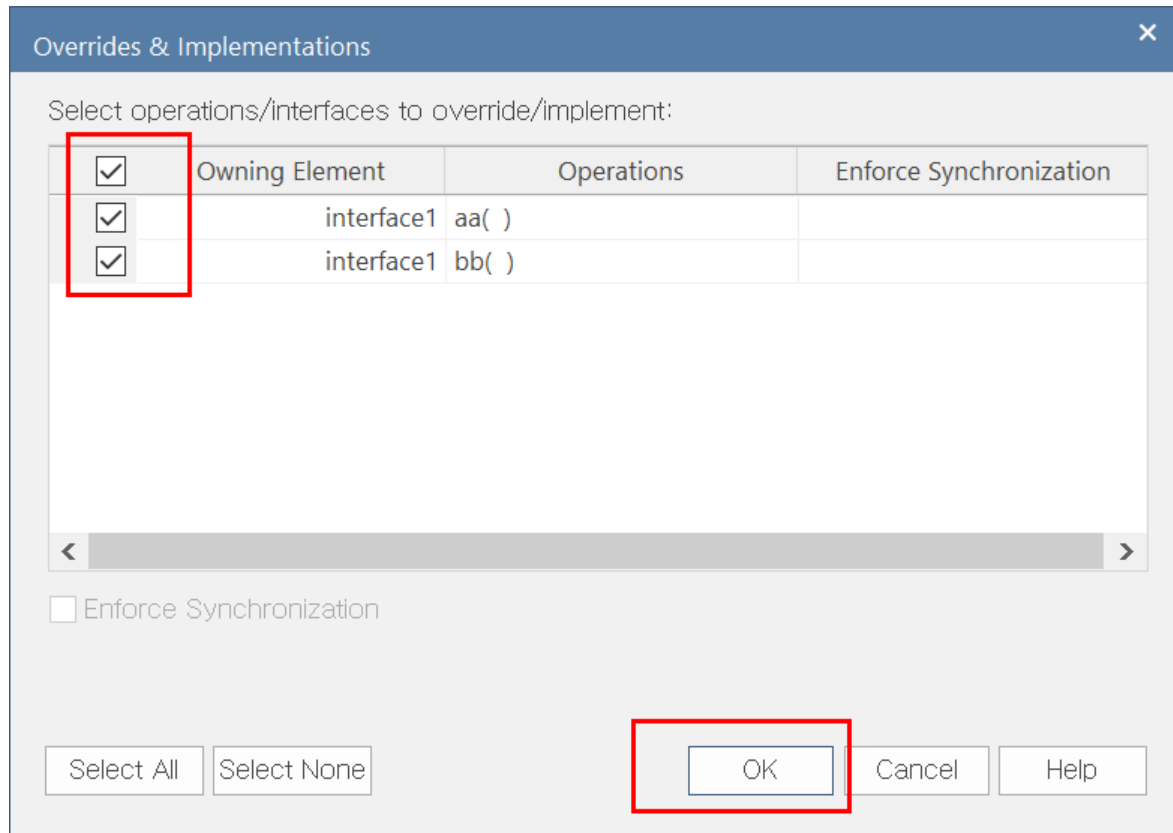
- 오른쪽과 같이 작성해 보세요.

선을 꺾는 방법: shift 클릭할 때마다
한번씩 꺾임



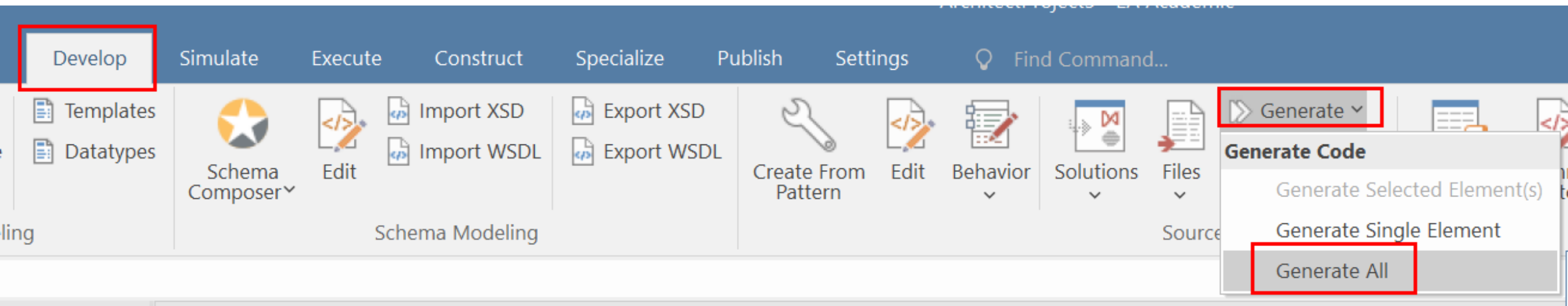
실습

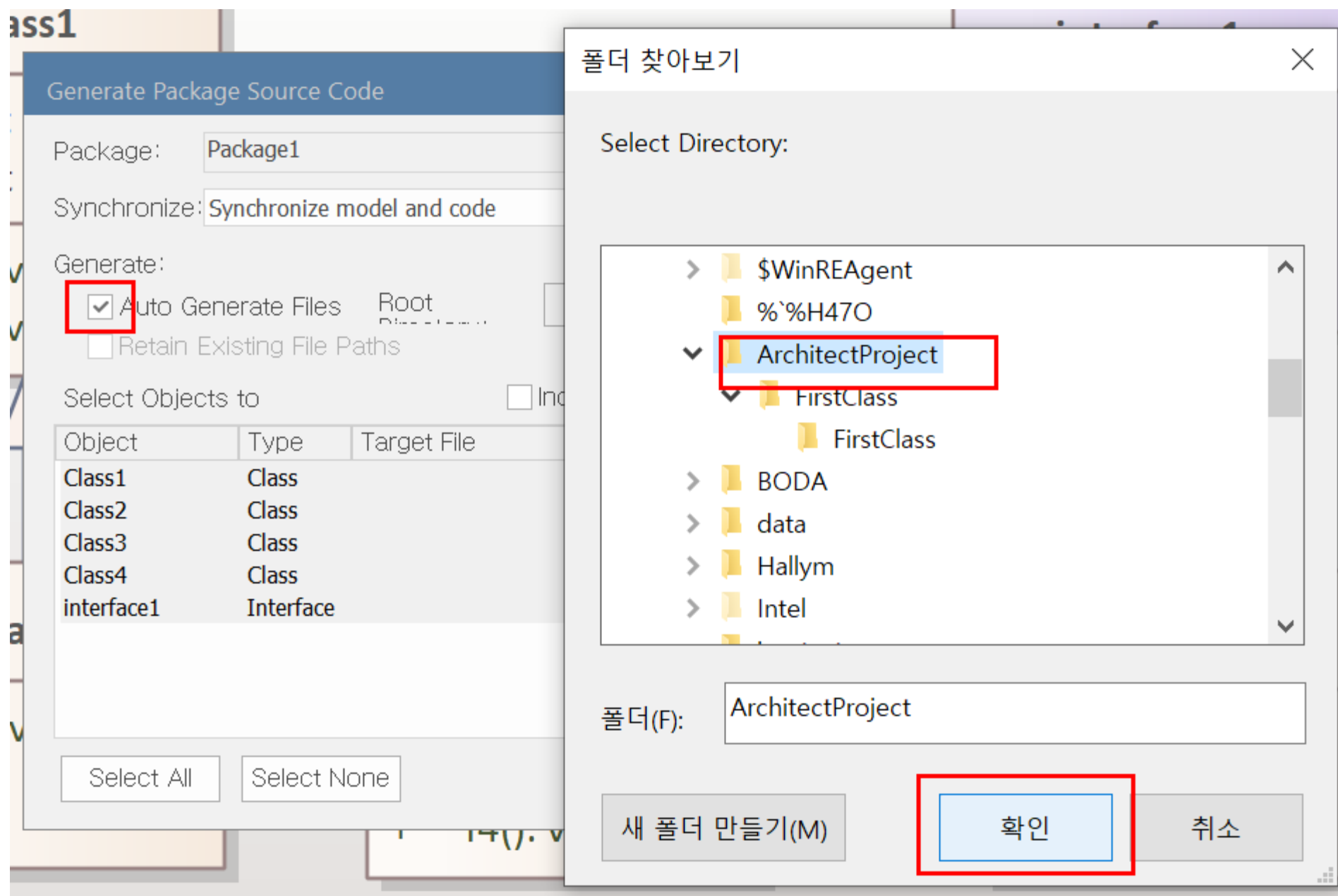
- 인터페이스를 구현하는 클래스 작성
 - 유스케이스 다이어그램과 유사



실습


- 소스코드 생성
 - 다음 메뉴 선택





Generate Package Source Code

Package: Package1



Generate

Cancel

Synchronize: Synchronize model and code

Generate:

☒ Auto Generate Files

Root

☐ Retain Existing File Paths

Select Objects to

Object	Type	Target File
Class1	Class	C:\ArchitectProject\Package1\Class1.java
Class2	Class	C:\ArchitectProject\Package1\Class2.java
Class3	Class	C:\ArchitectProject\Package1\Class3.java
Class4	Class	C:\ArchitectProject\Package1\Class4.java
interface1	Interface	C:\ArchitectProject\Package1\interface1.java

Select All

Select None

Batch generation

Current Action

Generate selected objects:

---generating C:\ArchitectProject\Package1\Class1.java

---generating C:\ArchitectProject\Package1\Class2.java

---generating C:\ArchitectProject\Package1\Class3.java

---generating C:\ArchitectProject\Package1\Class4.java

---generating C:\ArchitectProject\Package1\interface1.java

Generation complete!

Cancel Generation

Close

실습

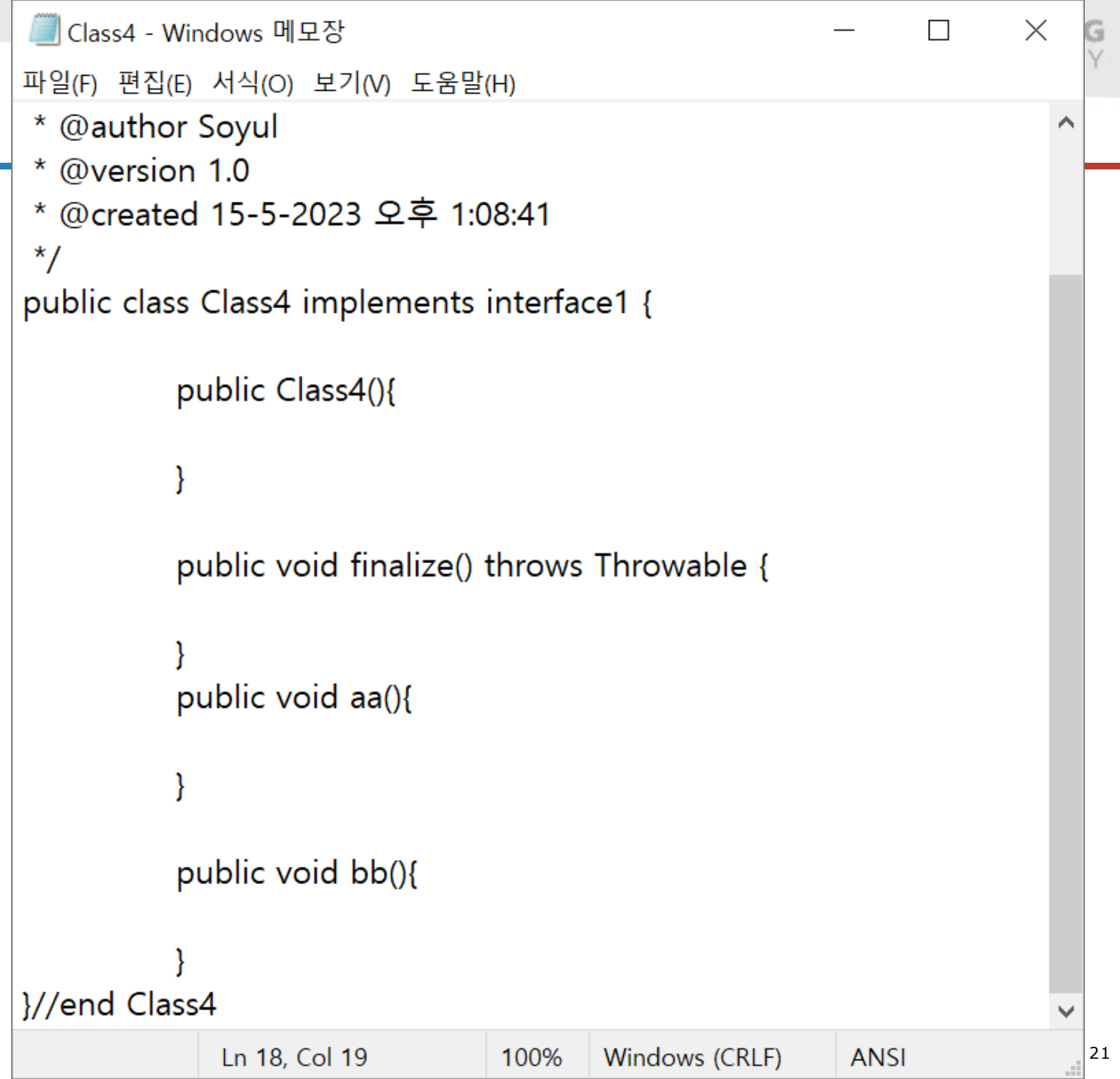
- 해당 폴더에 가보면 다음과 같이 자바 소스코드 생성 확인
 - 메모장 등으로 열어서 확인 또는 이클립스 등에서 open

내 PC > 로컬 디스크 (C:) > ArchitectProject > Package1

^		
	이름	유형
📌	Class1	JAVA 파일
📌	Class2	JAVA 파일
📌	Class3	JAVA 파일
📌	Class4	JAVA 파일
📌	interface1	JAVA 파일

실습

- 인터페이스를 구현하는
클래스 생성 확인
 - 생성자와 소멸자도 자동 생성



The screenshot shows a Windows Notepad window titled "Class4 - Windows 메모장". The menu bar includes "파일(F)", "편집(E)", "서식(O)", "보기(V)", and "도움말(H)". The text area contains the following Java code:

```
* @author Soyul
* @version 1.0
* @created 15-5-2023 오후 1:08:41
*/
public class Class4 implements interface1 {

    public Class4(){

    }

    public void finalize() throws Throwable {

    }
    public void aa(){

    }

    public void bb(){

    }

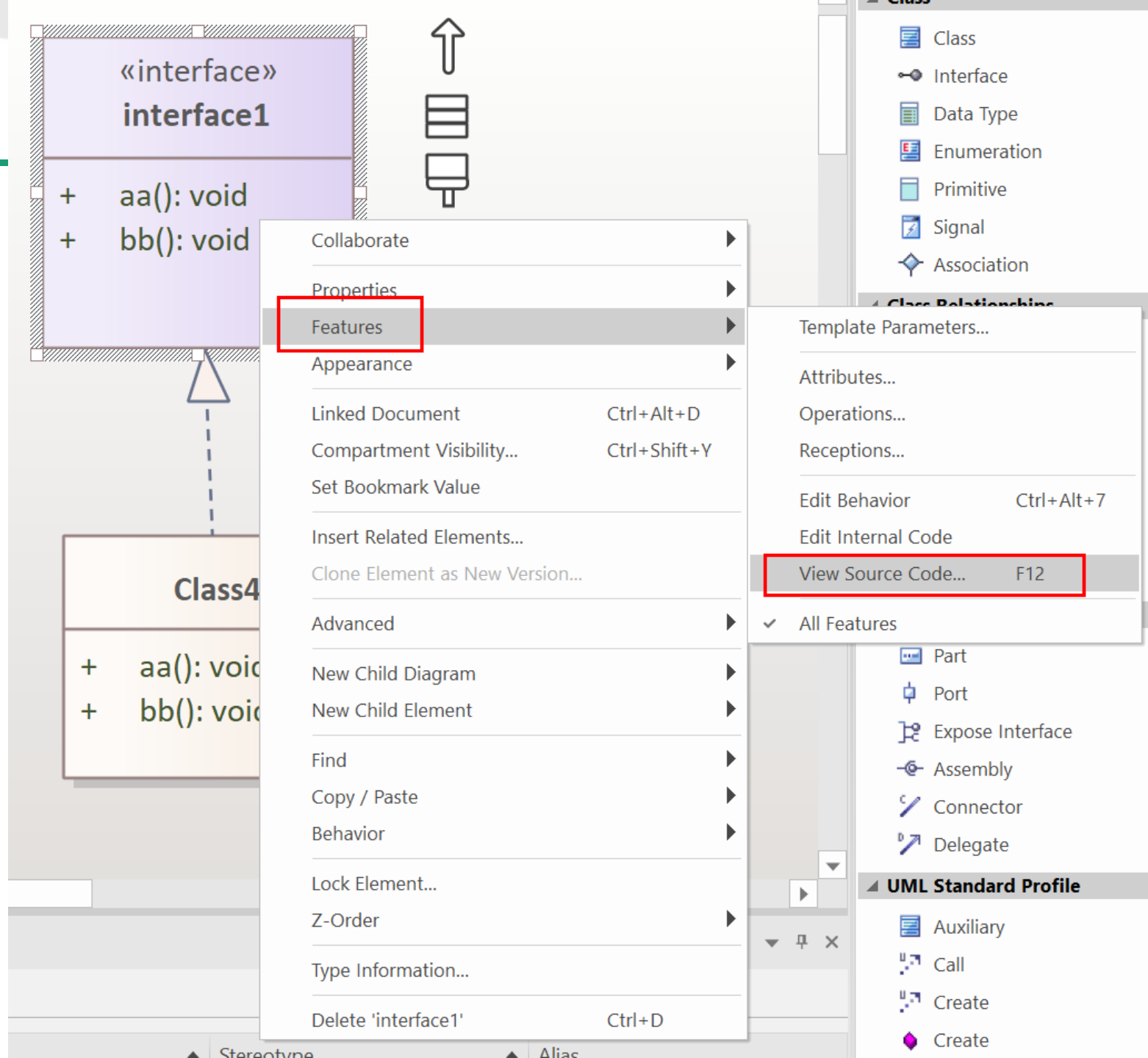
}
} //end Class4
```

The status bar at the bottom indicates "Ln 18, Col 19", "100%", "Windows (CRLF)", and "ANSI".

실습

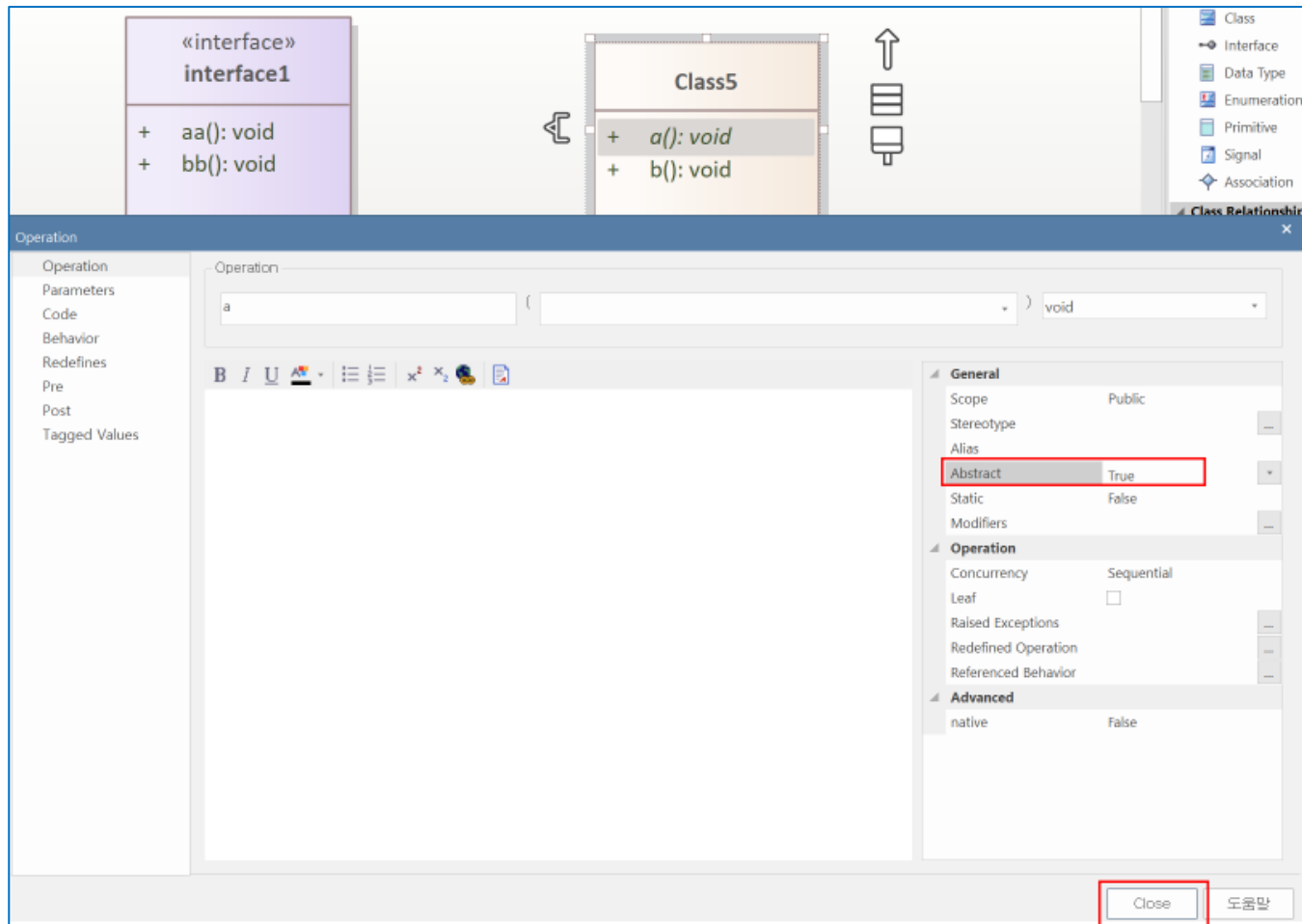
- 클래스를 클릭 후 오른쪽과 같이 메뉴를 선택하면 EA에서 확인 가능
- 단, 수정이 되면 다시 생성해야 EA 반영(당연)

```
1 package Package1;
2
3
4 /**
5  * @author Soyul
6  * @version 1.0
7  * @created 15-5-2023 오후 1:08:41
8  */
9 public interface interface1 {
10     public void aa();
11     public void bb();
12 }
13
14
15 }
```

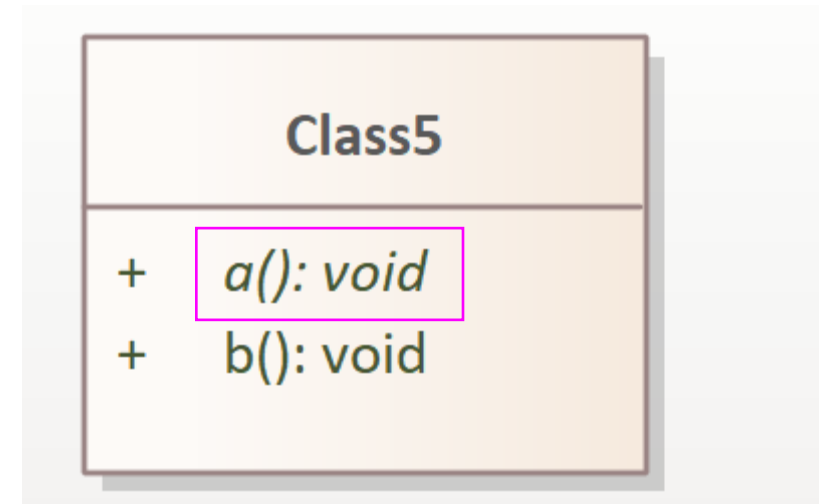


실습

- 추상 메소드(추상 클래스) 만들기 - 다이어그램에서 '해당 함수를 한번 클릭한 상태에서 다시 연속 두번 클릭' 또는 '하단 Features 창의 해당 함수에서 마우스 우클릭 properties'



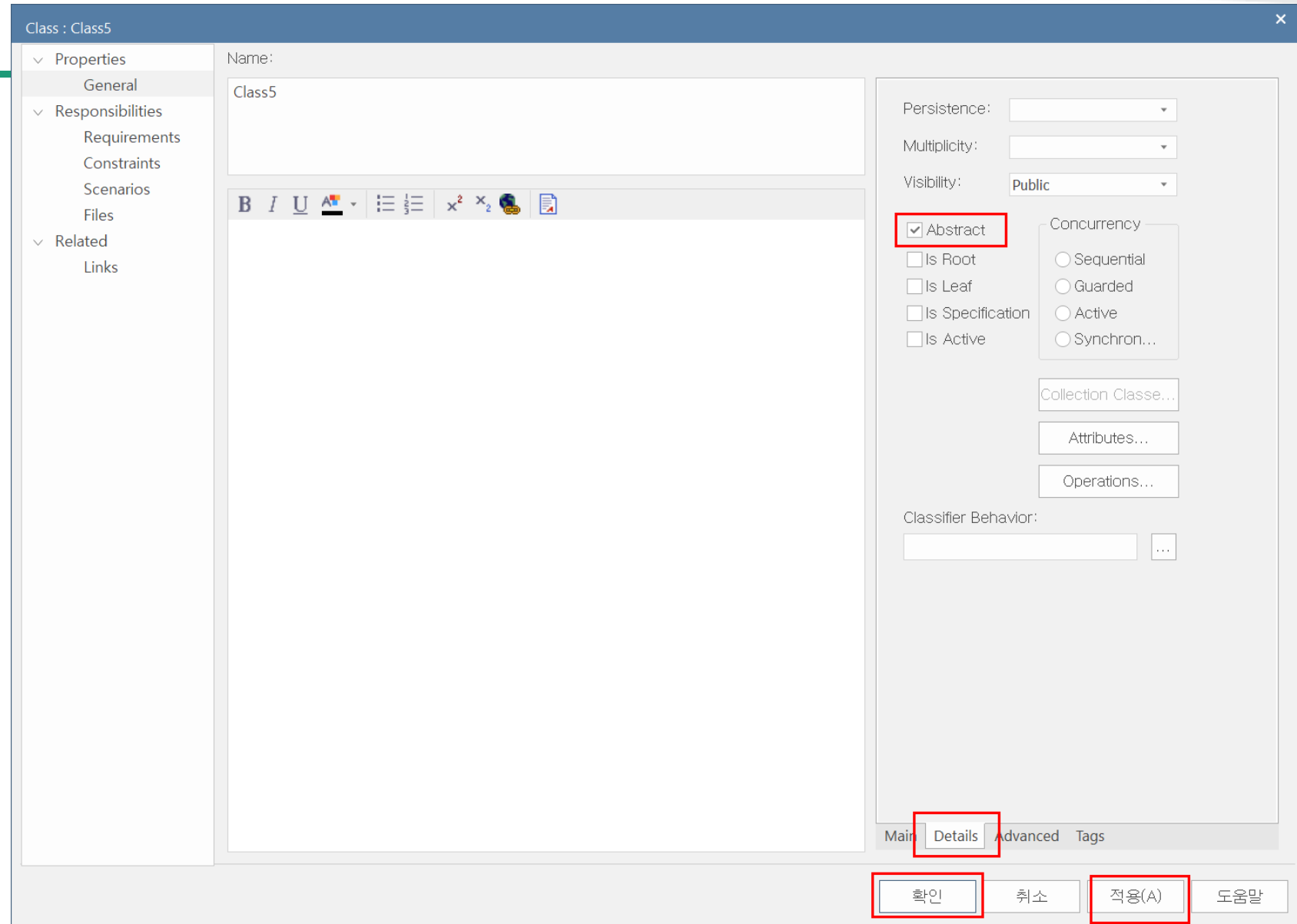
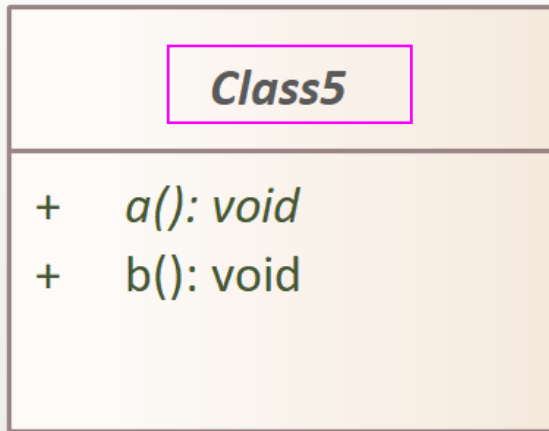
다시 소스 생성하여 결과 확인



실습

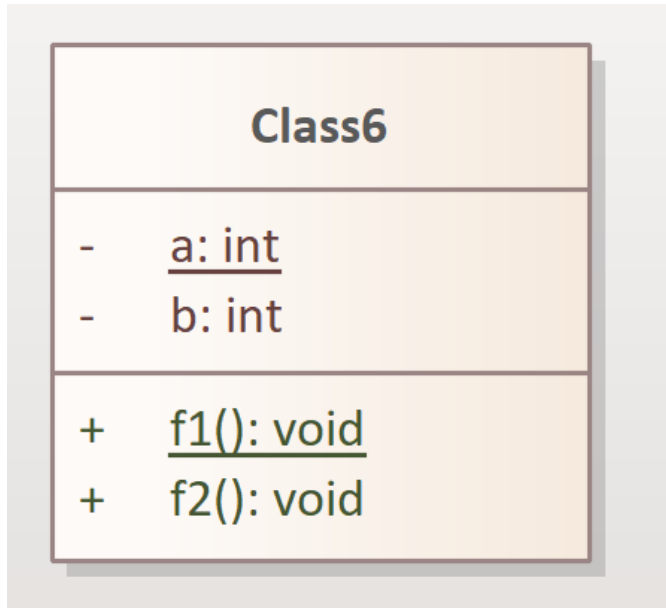
- 클래스 우 클릭 후 properties

다시 소스 생성하여 결과 확인



실습

- static 멤버 생성 - a와 f1()을 static 멤버로 바꾸고, 다시 소스를 생성하여 확인



The screenshot shows the 'Attribute' editor for the attribute 'a' of type 'int'. The 'Static' checkbox is checked, indicating it is a static member. The 'General' tab is selected, showing the attribute's scope as 'Private'. The 'Details' section shows 'Static' is 'True'. The 'Multiplicity' section shows 'Multiplicity' is '[1]'. The 'Advanced' section shows various flags like 'Transient', 'Derived', etc., all set to 'False'.

Attribute	Type
a	int

General

- Scope: Private
- Stereotype: ...
- Alias: ...
- Initial Value: ...

Attribute

- Leaf: ☐
- Redefined Property: ...
- Subsetted Property: ...

Details

- Static: True
- Const: False
- Property: ...

Multiplicity

- Multiplicity: [1]
- Containment: Not Specified
- Container Type: ...
- Is Collection: False

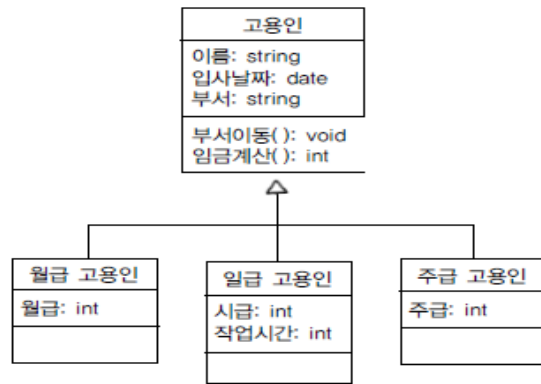
Advanced

- Transient: False
- Derived: False
- Derived Union: False
- Is ID: False
- Is Literal: False

Buttons: Close, 도움말

해보기

- 수업시간 교재의 일반화 및 실현(실체화) 관계를 작성 후 제출



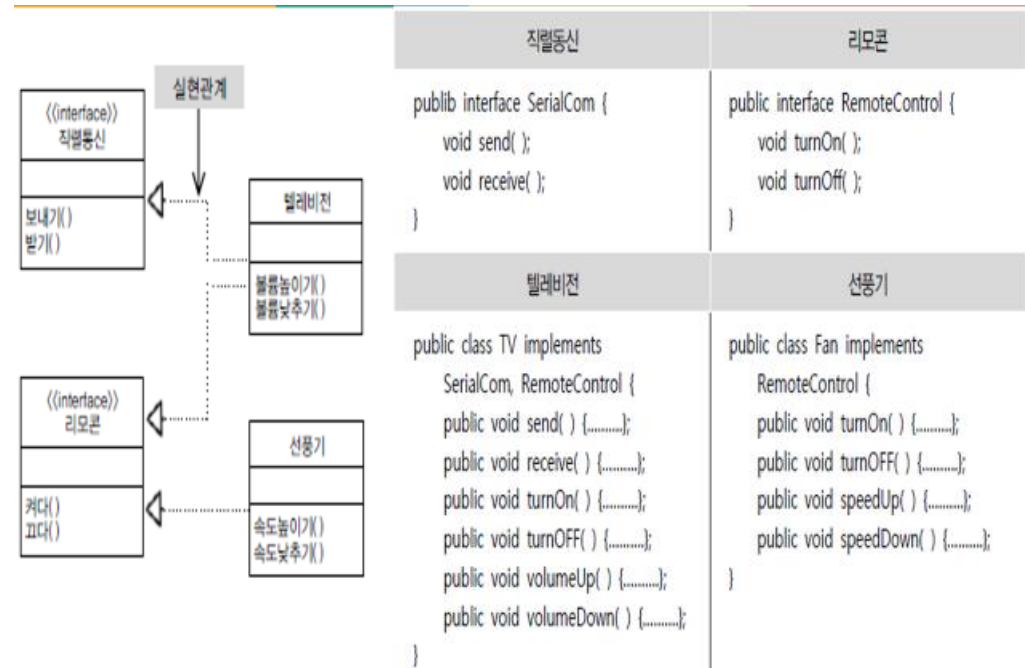
```

class 고용인 {
    String 이름;
    Date 입사날짜;
    String 부서;
    void 부서이동( );
    int 임금계산( );
}

class 월급고용인 extends 고용인 {
    int 월급;
}

class 일급고용인 extends 고용인 {
    int 시급;
    int 작업시간;
}

class 주급고용인 extends 고용인 {
    int 주급;
}
    
```





 **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!