올인원
스프링 프레임워크

Chapter 02
스프링의 이해

목차

학습목표

- 스프링 프레임워크가 무엇인지 살펴봅니다.
- 자바와 이클립스를 설치합니다.
- 스프링의 DI와 IoC 기능에 대해서 이해합니다.

Section 01

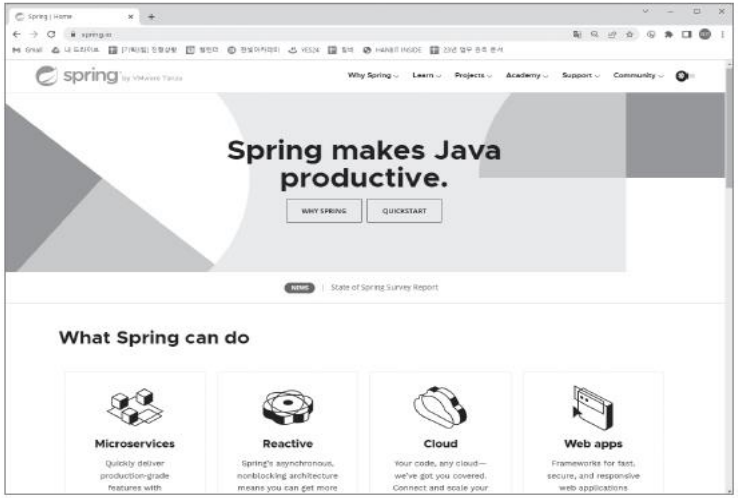
스프링 프레임워크란?

스프링 프레임워크

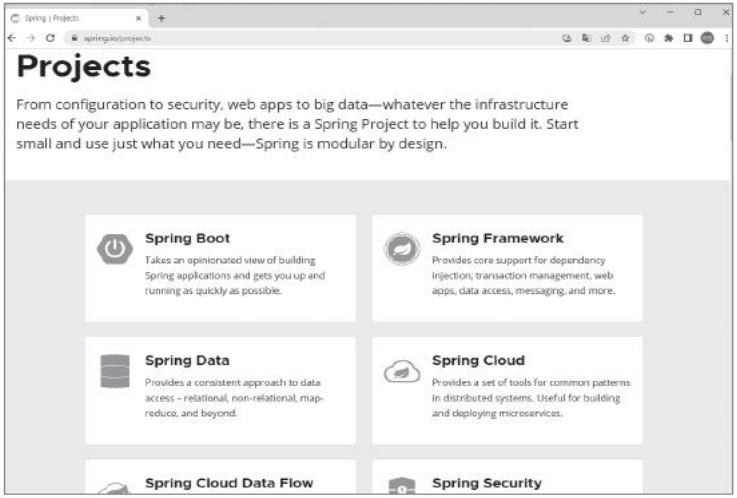
- 자바 기반의 애플리케이션을 개발하기 위한 오프소스 프레임워크로 일반적으로 줄여서 스프링이라고 함
- 웹 애플리케이션 제작에 뛰어난 강점이 있어 여러 산업 분야에서 널리 사용됨

스프링 MVC

- 스프링을 기반으로 하는 하위 프레임워크
- 웹 애플리케이션 개발에 최적화된 프레임워크



(a) 스프링 공식 홈페이지



(b) 다양한 스프링 프로젝트

그림 2-1 스프링에서 진행하는 다양한 프로젝트

스프링 프레임워크의 모듈

- 스프링은 다양한 모듈로 이루어져 있음
- 모듈의 종류는 방대하기 때문에 필요할 때 사용할 모듈의 라이브러리를 개발 프로젝트에 가져와 사용하면 됨

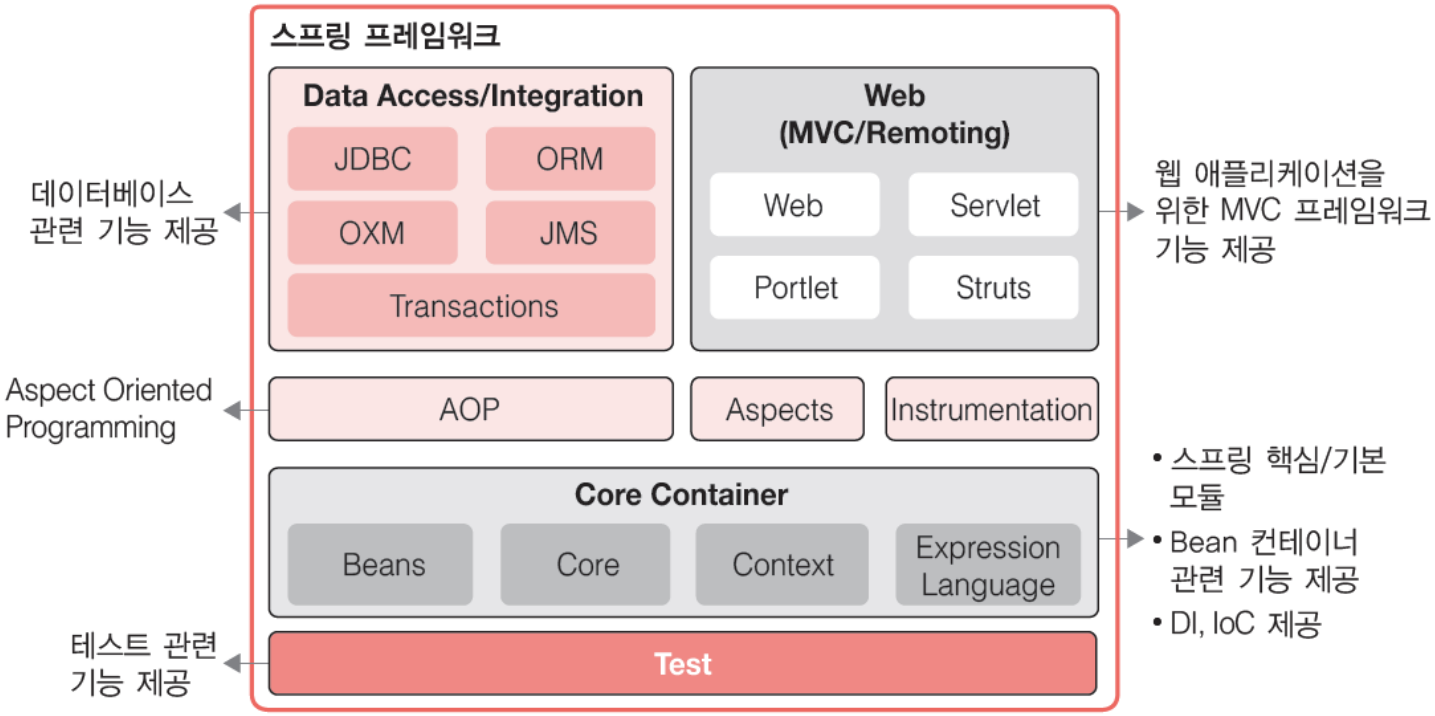


그림 2-2 필요에 따라 특정 모듈만 이용할 수 있는 경량 컨테이너인 스프링

■스프링에서 제공하는 대표적인 모듈

- 이러한 모듈을 사용하려면 프로젝트에 모듈에 대한 '의존 설정'을 해야 함

표 2-1 스프링 모듈

스프링 모듈명	기능
spring-core	스프링의 핵심인 DI Dependency Injection와 IoC Inversion of Control를 제공
spring-webmvc	스프링에서 제공하는 컨트롤러 Controller와 뷰 View를 이용한 스프링 MVC 구현 기능 제공
spring-jdbc	데이터베이스를 쉽게(적은 양의 코드) 다룰 수 있는 기능 제공
spring-tx	스프링에서 제공하는 트랜잭션 관련 기능 제공
spring-security	애플리케이션 보안을 담당하는 기능 제공

Section 02

스프링 개발 환경 구축

■본 교재의 실습환경

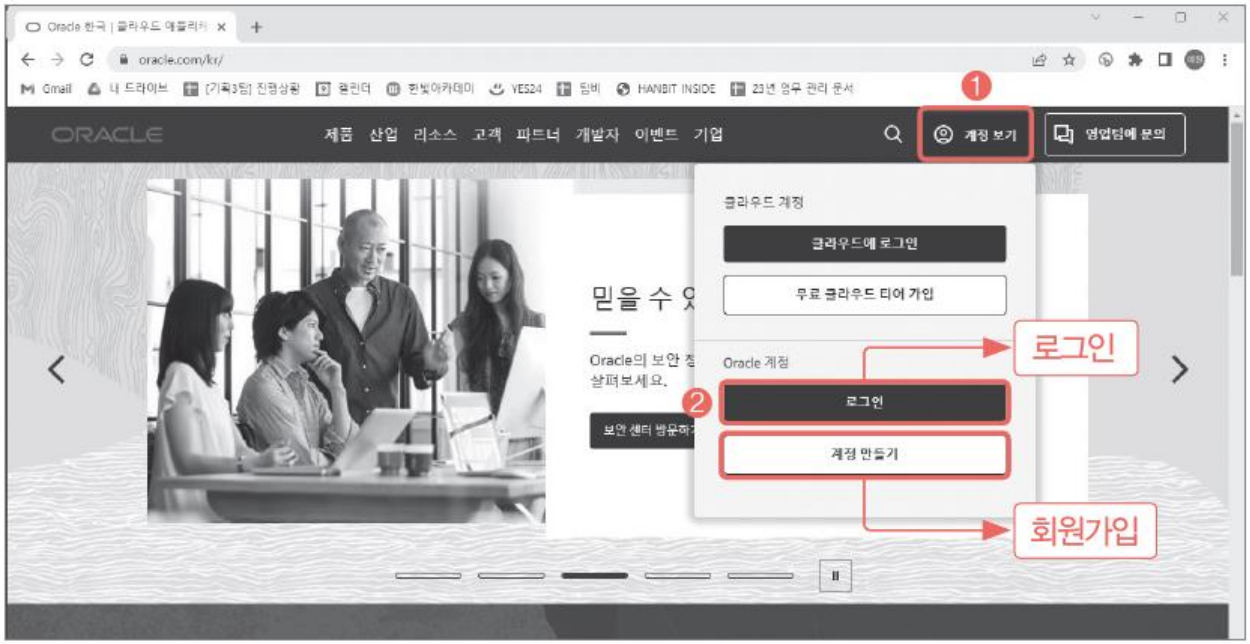
항목	프로그램	이 책의 버전
자바 개발도구	JDK	JDK 11
통합개발환경	이클립스(2~8장)	이클립스 2022-12-R
	STS(9~14장)	STS3
서블릿 컨테이너	아파치 톰캣	Apache Tomcat 9
데이터베이스	MariaDB	MariaDB Server 10

- **JDK11을 사용하는 이유**
  - ✓ **만약 컴퓨터에 JDK11이 아닌 다른 버전이 설치되어 있다면 '프로그램 추가/제거'를 이용해서 제거 후 JDK11을 설치해야 함**
  - ✓ **9장부터 사용하는 STS3 IDE에서 JDK11을 기반으로 실습이 진행하는데, 만약 JDK11이 아닌 다른 버전을 사용하면 실습에 문제가 발생할 수 있기 때문임**

1. JDK 설치하기

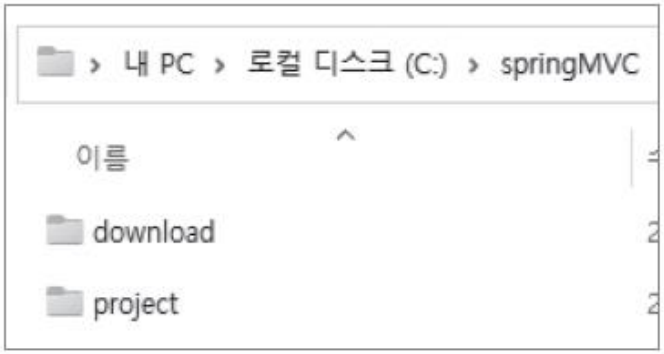
■ 1. 웹 브라우저에서 오라클 홈페이지(https://www.oracle.com/kr/index.html)에 접속하기

- 설치 파일을 내려받기 위해서는 오라클 계정이 있어야 함
- 오른쪽 상단의 '계정 보기'를 클릭하여 계정이 있다면 <로그인>을, 계정이 없다면 <계정 만들기>를 클릭



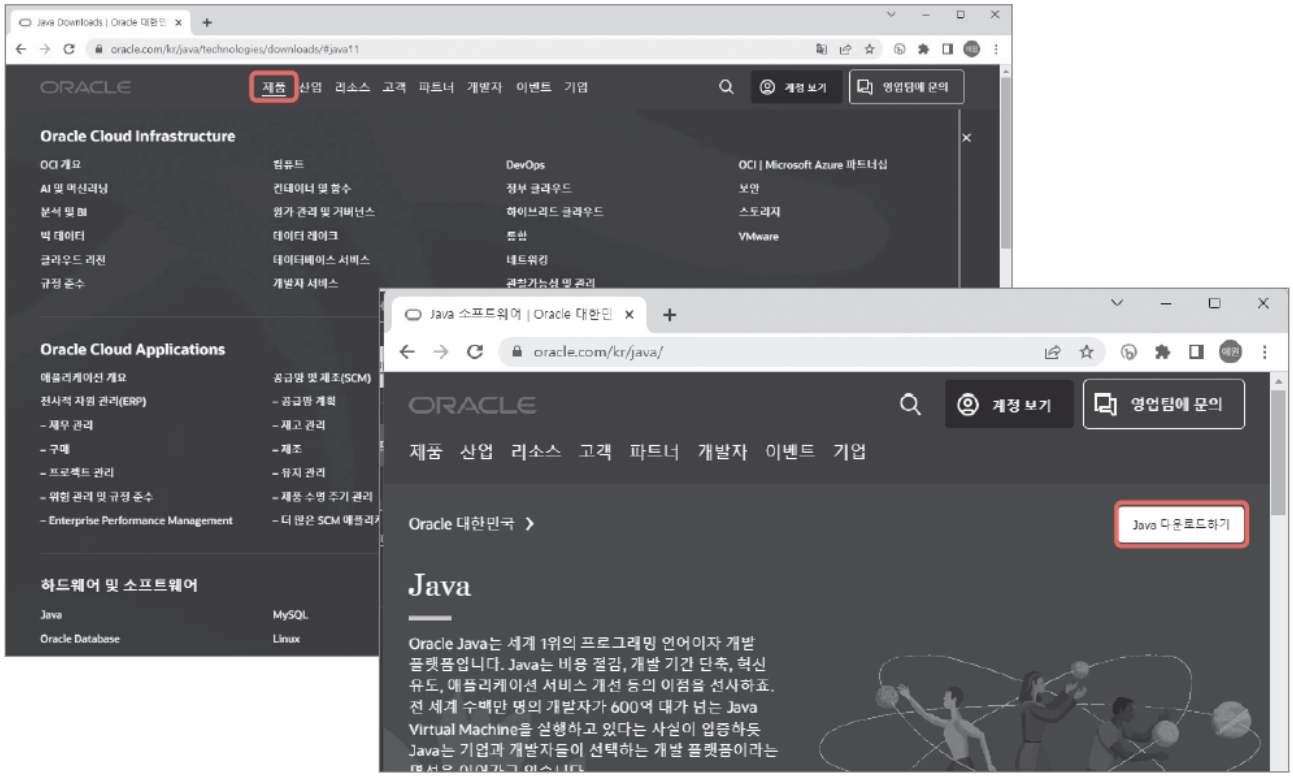
■ 2. 다운로드한 파일과 예제를 저장하기 위한 디렉터리 만들기

- C 드라이브에 [springMVC] 폴더 생성
- [springMVC] 폴더 내부에 [download], [project] 폴더 생성
  - ✓ [download] 폴더: 앞으로 다운로드할 자바, 이클립스, 톰캣, 데이터베이스 등의 설치 파일을 저장하는 폴더
  - ✓ [project] 폴더: 앞으로 진행할 예제를 저장할 폴더



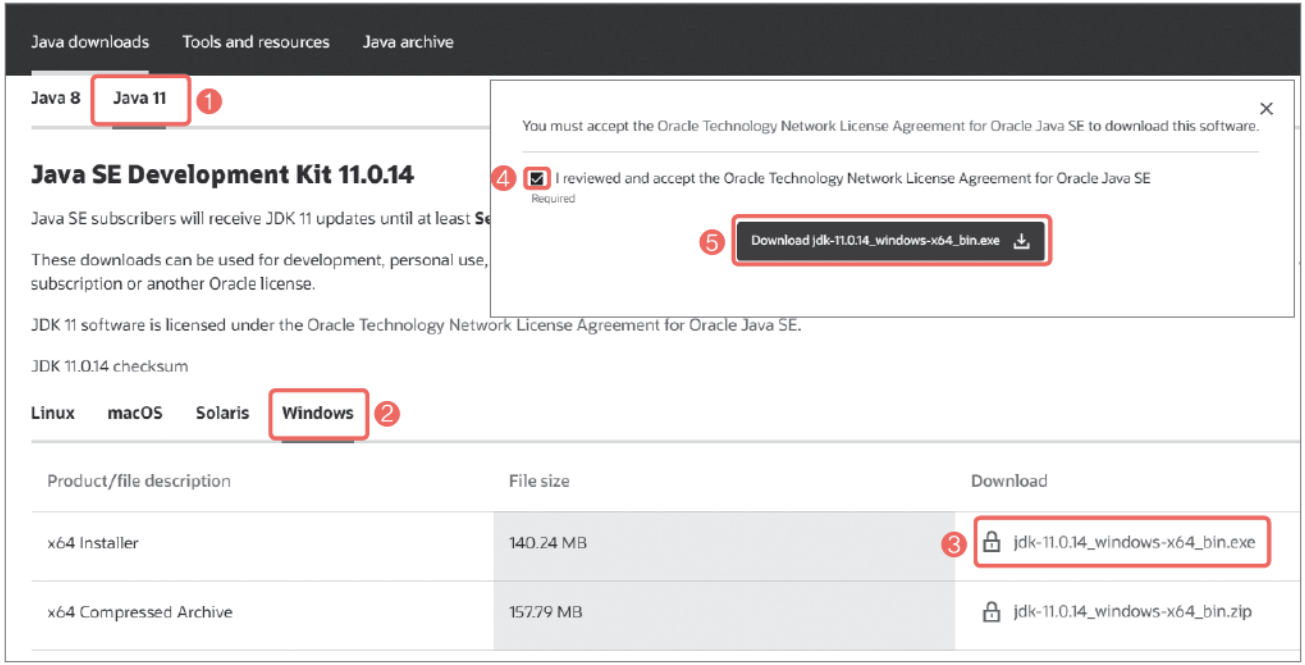
1. JDK 설치하기

■ 3. 오라클 홈페이지의 상단에서 [제품]-[Java]를 클릭하고, 그다음 페이지에서 [Java 다운로드하기]를 클릭



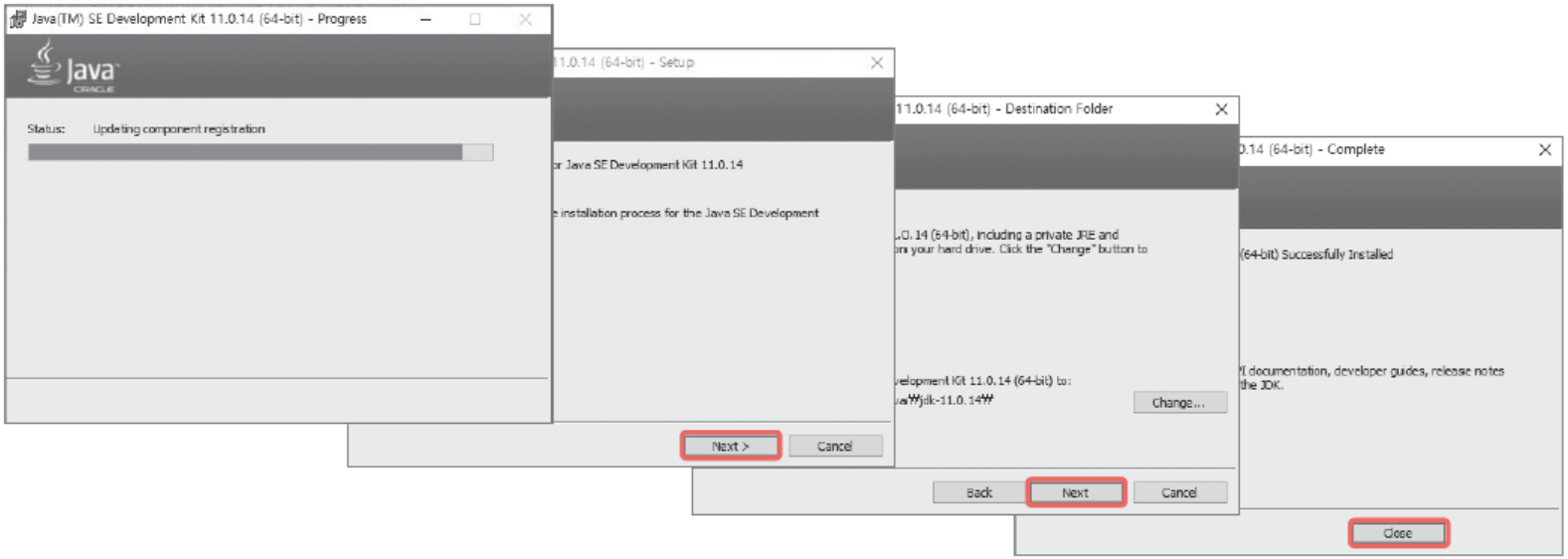
4. JDK11을 내려받기 위해 스크롤하여 화면을 내림

- 페이지 하단의 Java 11을 선택하고 Windows를 선택한 뒤 jdk-11.0.14_windows-x64_bin.exe를 클릭
- 라이선스에 동의하고 <Download jdk-11.0.14_windows-x64_bin.exe> 버튼을 클릭하여 다운로드



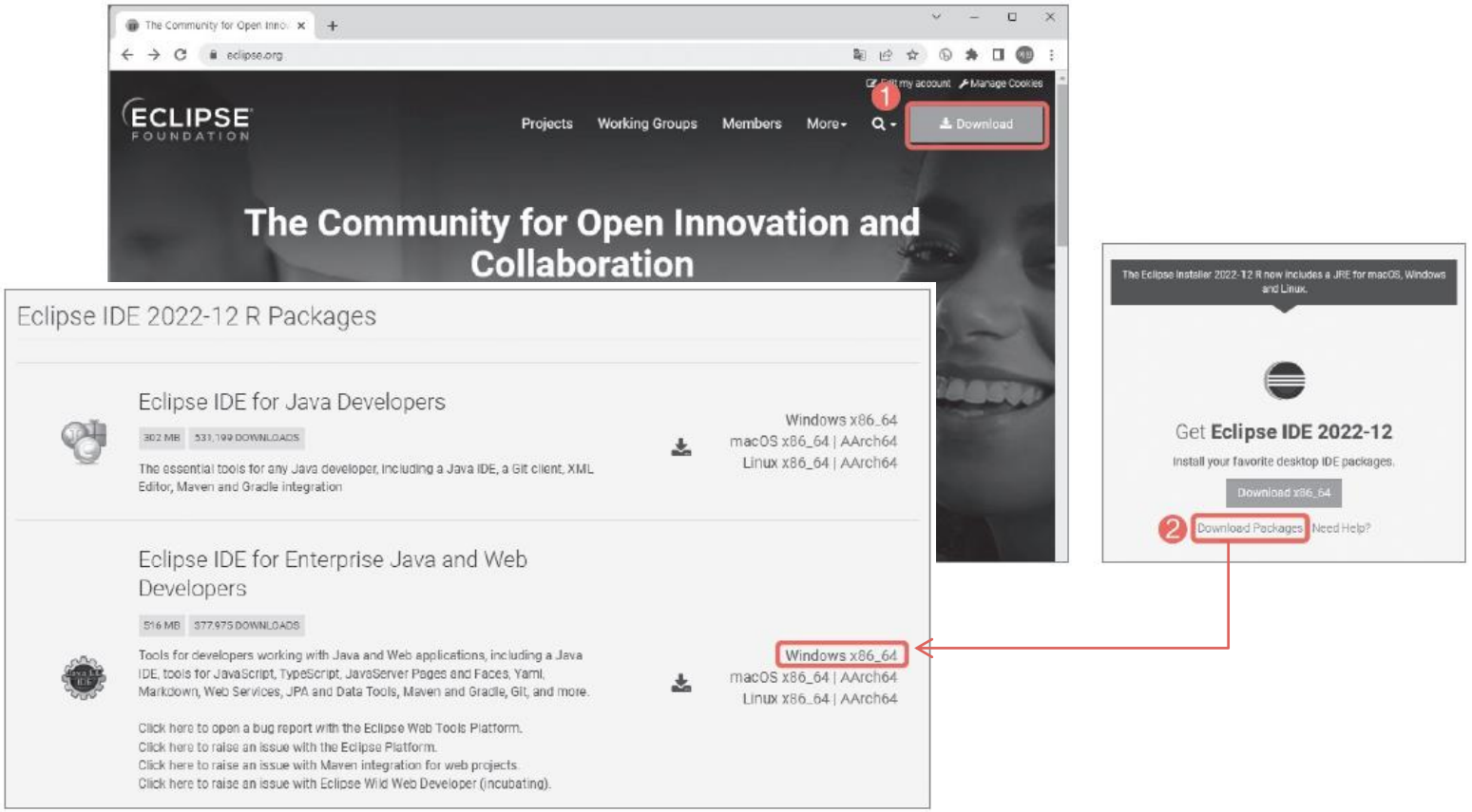
1. JDK 설치하기

- 5. 자바 설치 파일(jdk-11.0.14_windows-x64_bin.exe)을 [download] 폴더로 옮긴 후 더블클릭하여 설치를 시작
  - <Next> 버튼을 클릭하다가 마지막으로 <close> 버튼을 클릭해 설치 마무리



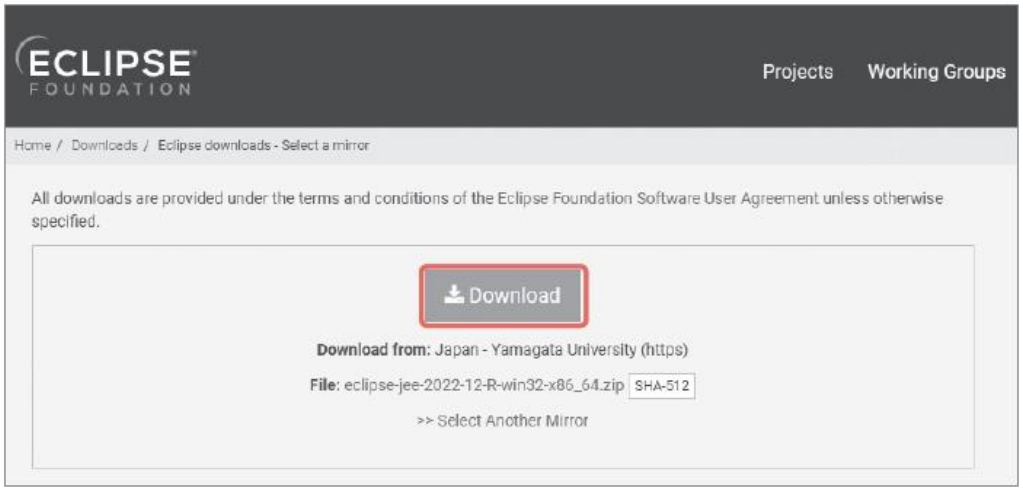
2. 이클립스 설치하기

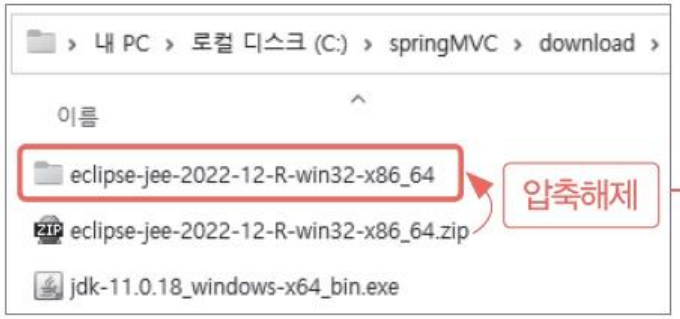
- 1. 웹 브라우저에서 이클립스(https://www.eclipse.org/) 홈페이지에 접속하여 우측 상단의 <Download> 버튼을 클릭하기
  - 이어지는 화면에서 <Download Package>를 클릭하고 설치 진행
- 2. 2022-12 R Packages의 'Eclipse IDE for Enterprise Java and Web Developers'를 선택하고, 'Windows x86_64'를 클릭하여 다운로드

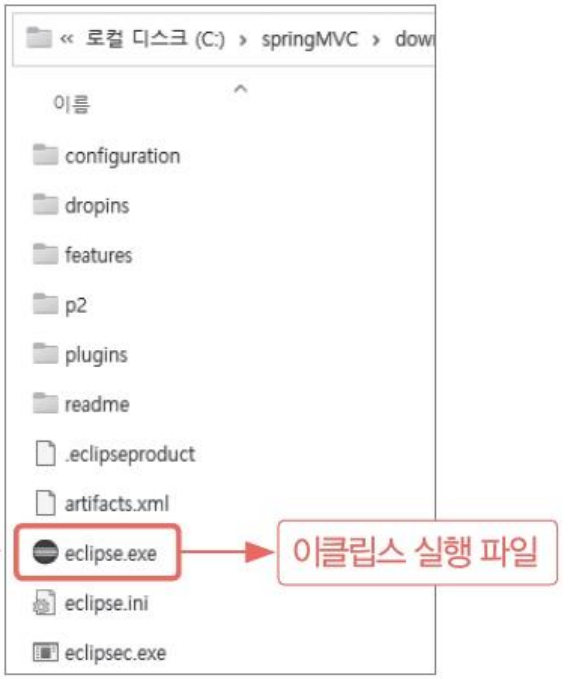


2. 이클립스 설치하기

- 3. <Download> 버튼을 클릭하여 eclipse-jee-2022-12-R-win32-x86_64.zip 파일 다운로드하기
  - 다운로드 위치: `C:\springMVC\download`
- 4. 다운로드한 이클립스는 압축파일로, 압축 해제하여 이클립스 실행 파일(eclipse.exe)을 더블클릭해서 이클립스를 실행하기

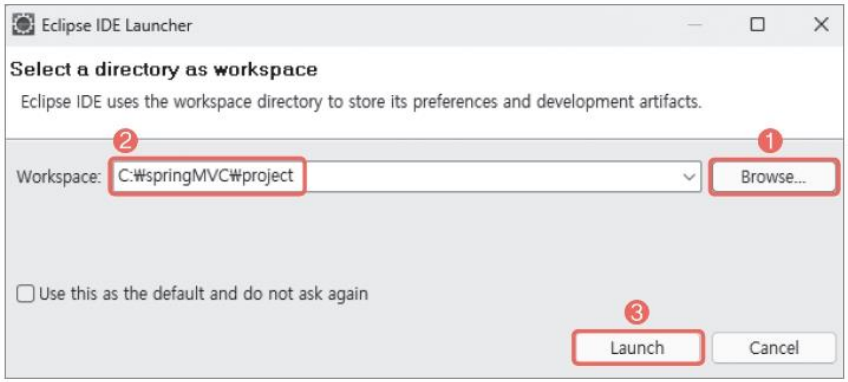


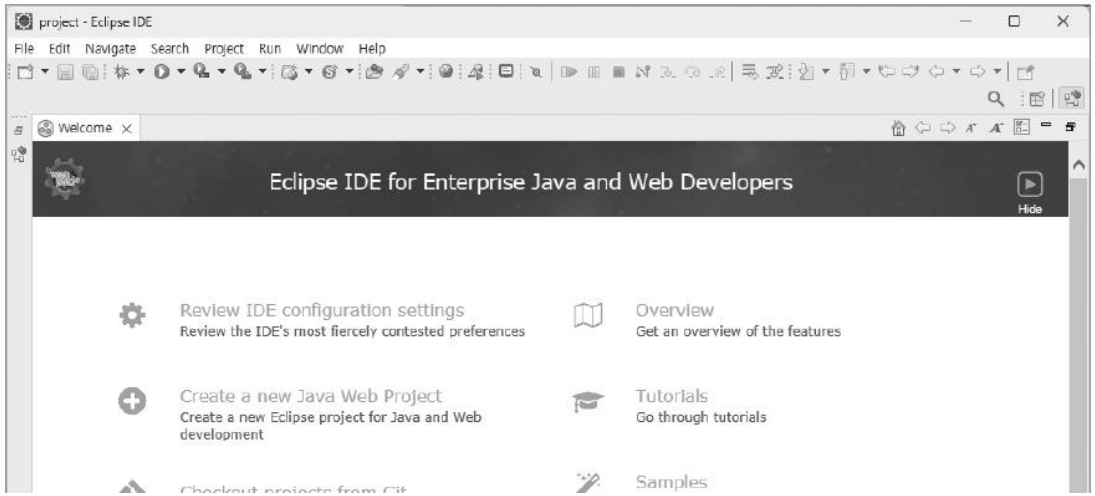




2. 이클립스 설치하기

- 5. 이클립스 로고 창이 잠시 보이고, Workspace를 설정하는 창이 뜨면 <Browse...>를 클릭하여 Workspace 경로를 변경하고 <Launch> 클릭하기
  - Workspace 경로: ‘C:\springMVC\project’
- 6. Welcome 페이지가 뜨면 이클립스가 정상적으로 실행된 것임





■ 이클립스의 자바 버전(JDK) 설정하기

- 자바 JDK11과 이클립스 2022-12-R을 설치했음
- 그런데 이클립스 2022-12-R에는 자바의 최신 버전인 17이 내장되어 있기 때문에 이클립스의 자바 버전을 JDK17이 아닌 JDK11로 변경해야 함

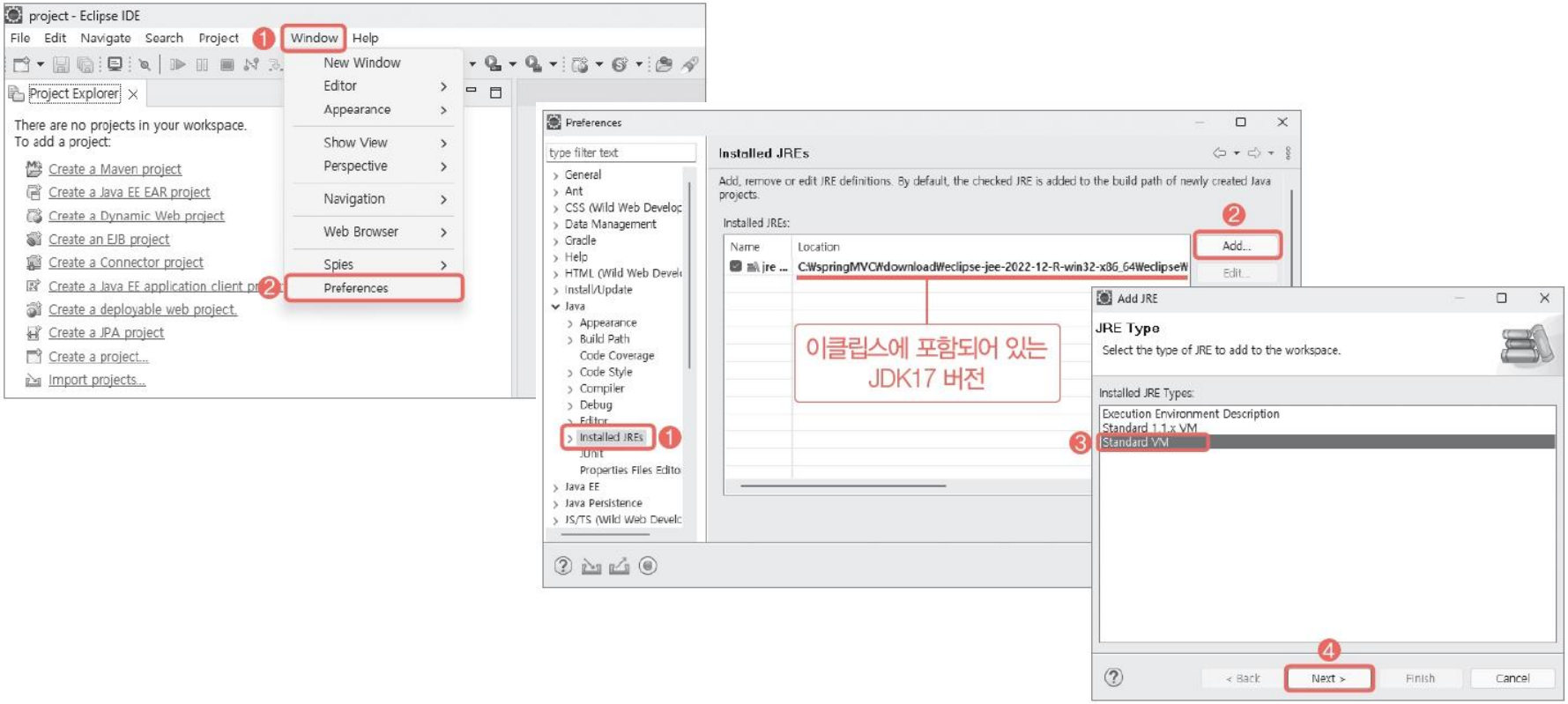
<여기서 잠깐!> 이클립스에서 자바를 JDK11로 변경하지 않고 최신 버전(JDK17)을 사용하면 안 되나요?

이 책에서는 JDK11 환경에서 모든 실습을 진행합니다. 따라서 JDK17을 사용하다가 실습이 막히면 학습에 어려움이 있을 수 있습니다. 또한 실무에서 이클립스의 자바 버전을 변경해야 하는 경우가 종종 발생하기도 합니다. 그런 경우를 대비해서 자바 버전을 변경하는 방법에 대해서 이번 기회에 알아두면 좋기 때문입니다.

2. 이클립스 설치하기

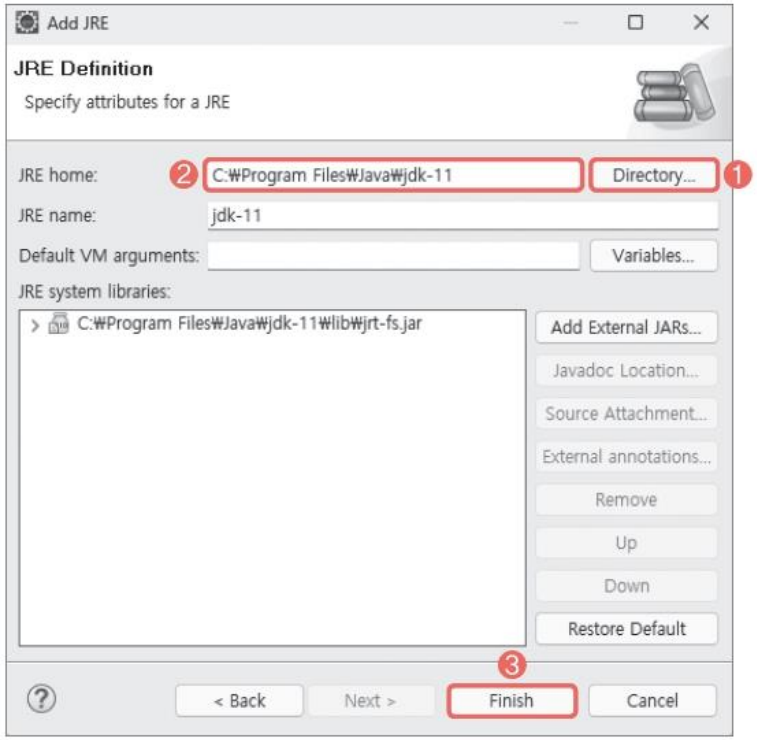
■ 이클립스의 자바 버전(JDK) 설정하기

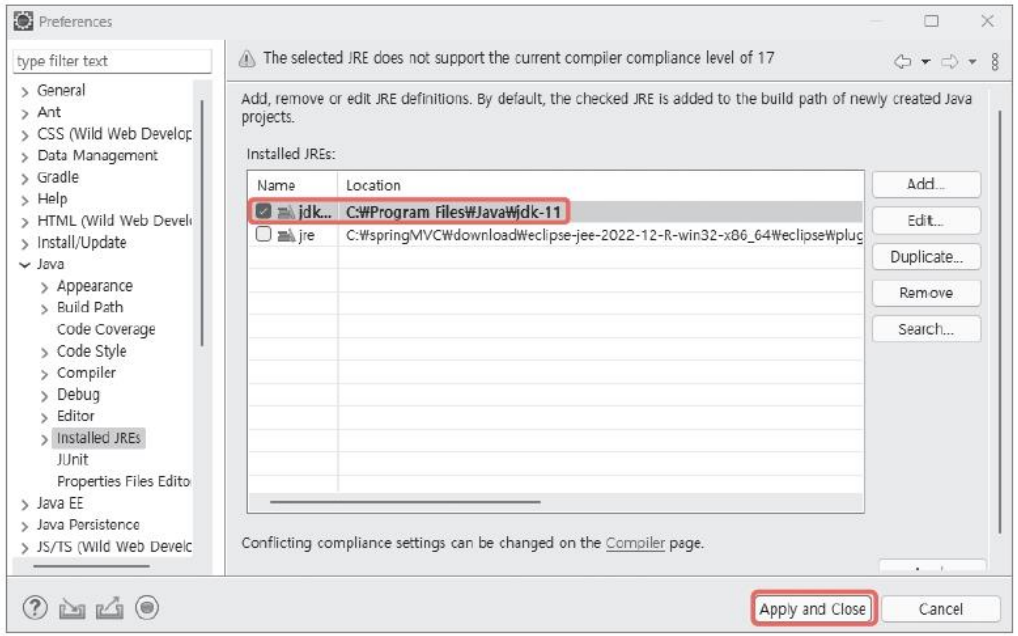
- 1. 이클립스 상단 메뉴에서 [Window]-[Preferences]를 선택
- 2. [Preferences] 창 왼쪽 메뉴에서 [Java]-[Installed JREs]를 클릭해 이클립스에 기본적으로 포함되어 있는 JDK17의 경로를 확인하기
  - ✓ <Add...>를 클릭하고, [Add JRE] 창에서 Standard VM을 선택한 후 <Next> 클릭



■ 이클립스의 자바 버전(JDK) 설정하기

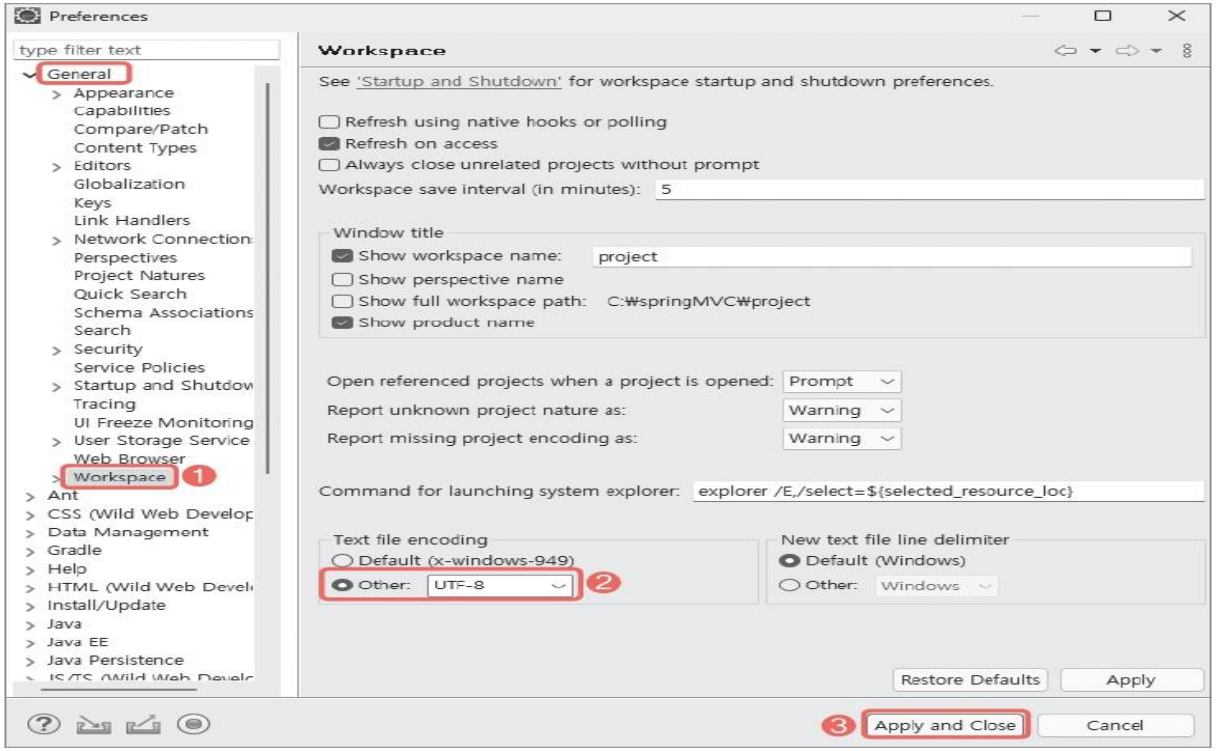
- 3. [JRE Definition] 창에서 <Directory>를 클릭하여 JRE home을 앞에서 설치한 JDK11 경로 설정하고 <Finish> 클릭
  - ✓ JDK11 경로: `C:\Program Files\Java\jdk-11.0.14`
- 4. 추가된 jdk-11의 체크박스에 체크하여 default로 설정하고, <Apply and Close>를 클릭해서 설정 종료





■인코딩 설정하기(UTF-8)

- **이클립스는 기본적으로 인코딩이 MS-949로 되어 있음. 이것을 UTF-8로 변경하기**
- 1. 이클립스의 [Window]-[Preferences]를 클릭
- 2. [Preferences] 창의 왼쪽 메뉴에서 [General]-[Workspace] 선택
- 3. 화면 하단의 Text file encoding에서 MS-949를 UTF-8로 변경하고, <Apply and Close>를 클릭해서 인코딩 설정을 종료하기



Section 03

스프링 DI와 IoC

■ 스프링 DI와 IoC 학습을 위한 간단한 계산기 자바 프로젝트

- 계산기 자바 프로젝트의 구조
  - ✓ MainClass의 main()에서 프로그램이 시작되면 MyCalculator를 생성
  - ✓ MyCalculator는 덧셈, 뺄셈, 곱셈, 나눗셈을 위한 각각의 객체(CalAdd, CalSub, CalMul, CalDiv)를 생성
  - ✓ 생성된 객체는 내부에서 사칙연산을 실행

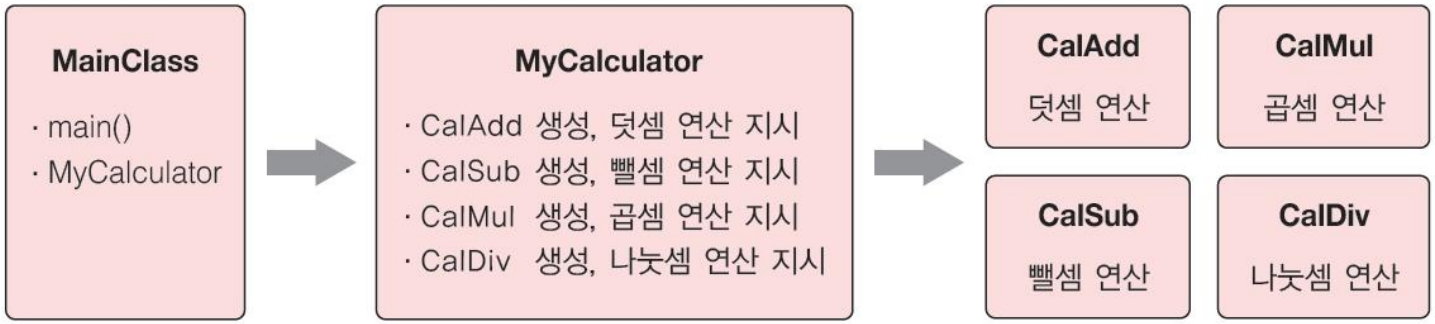
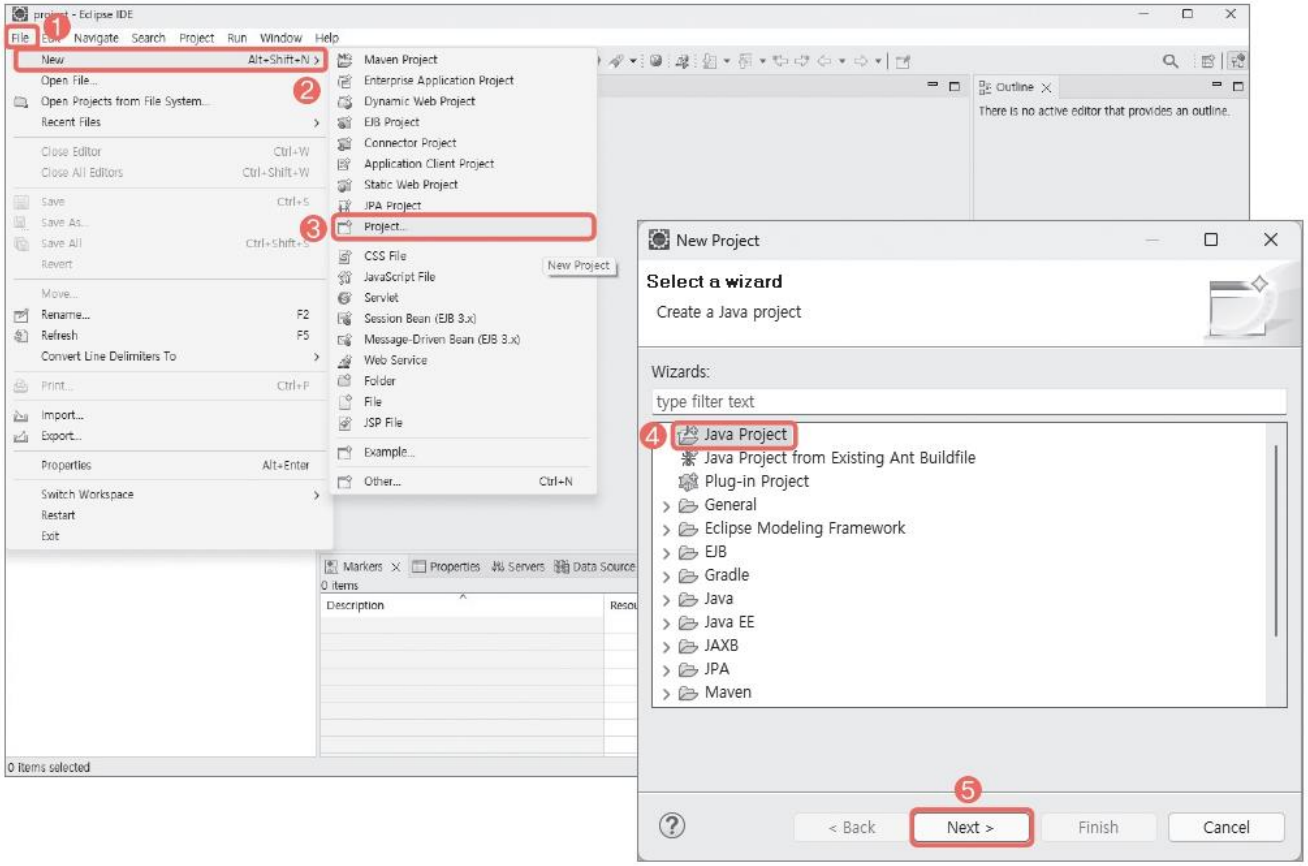


그림 2-3 계산기 프로젝트(MyCalculator)의 구성

ch02_pjt_01 프로젝트 생성하기

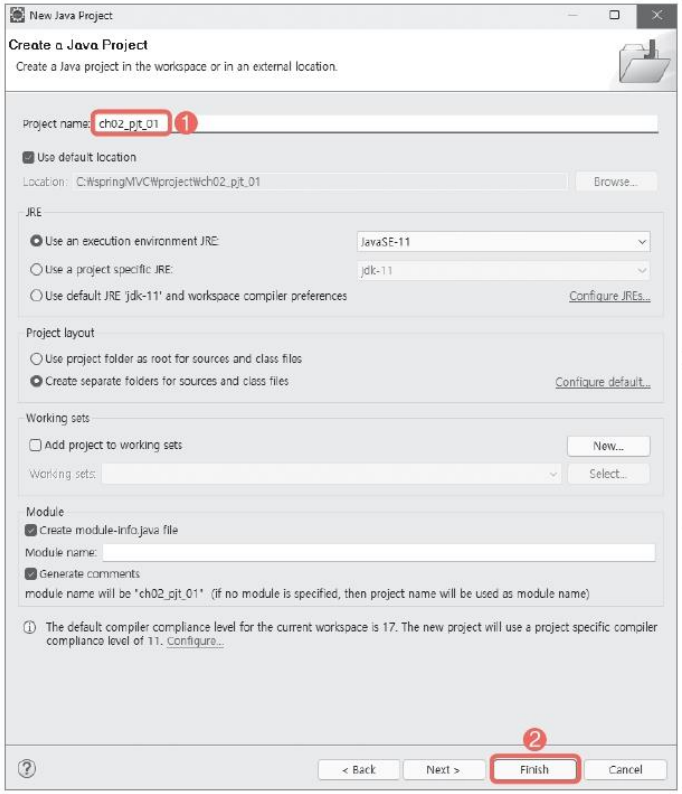
- 1. [File]-[New]-[Project]를 클릭하고, [New Project] 창에서 'Java Project'를 선택하고 <Next> 클릭

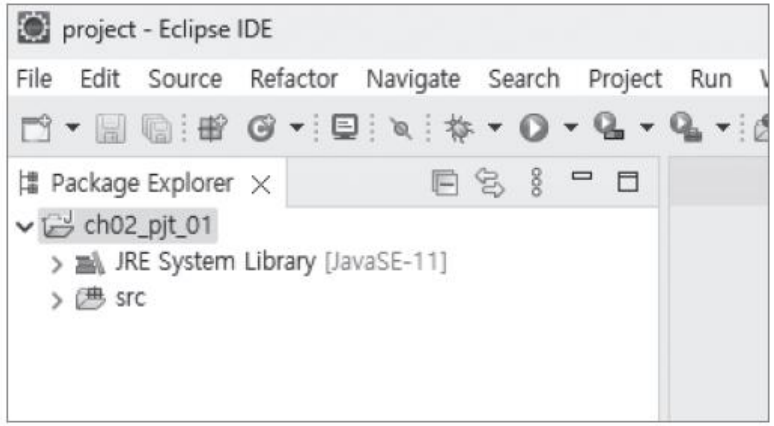


1. 계산기 프로젝트 만들기

■ ch02_pjt_01 프로젝트 생성하기

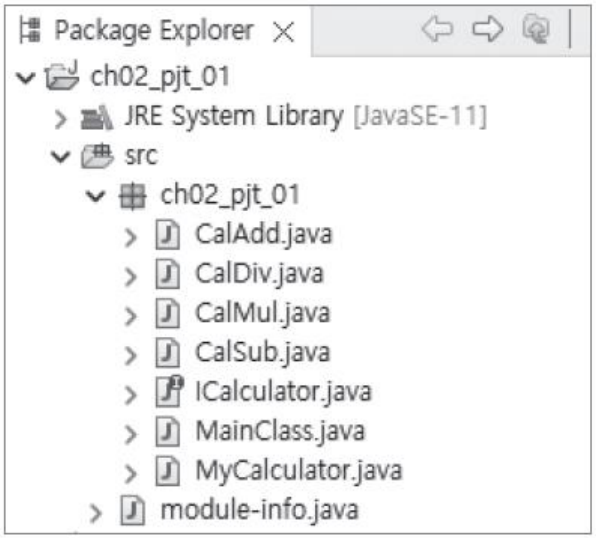
- 2. [New Java Project] 창에서 Project name을 ch02_pjt_01로 하고 <Finish> 클릭
- 3. [New module-info.java] 창이 나오면 <Don't Create>를 클릭
  - ✓ 만약 [Open Associated Perspective?] 창이 나오면 <Open Perspective>를 클릭
- 4. 프로젝트가 정상적으로 생성된 것을 확인하기





■ 클래스와 인터페이스 만들기

- 생성된 프로젝트에 다음과 같이 클래스와 인터페이스 생성하기



(a) 생성 완료된 전체 프로젝트 구조

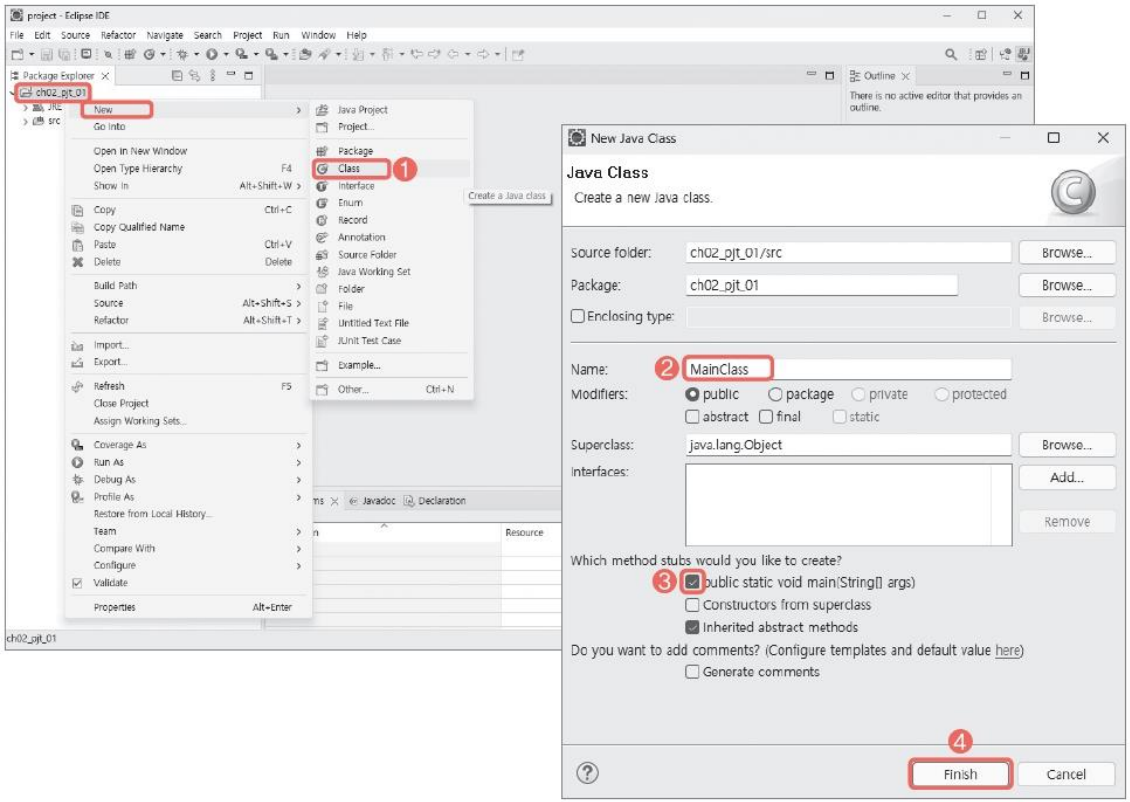
클래스	인터페이스
MainClass.java	ICalculator.java
MyCalculator.java	
CalAdd.java	
CalSub.java	
CalMul.java	
CalDiv.java	

(b) 생성할 클래스와 인터페이스

그림 2-4 계산기 프로젝트에 필요한 클래스와 인터페이스

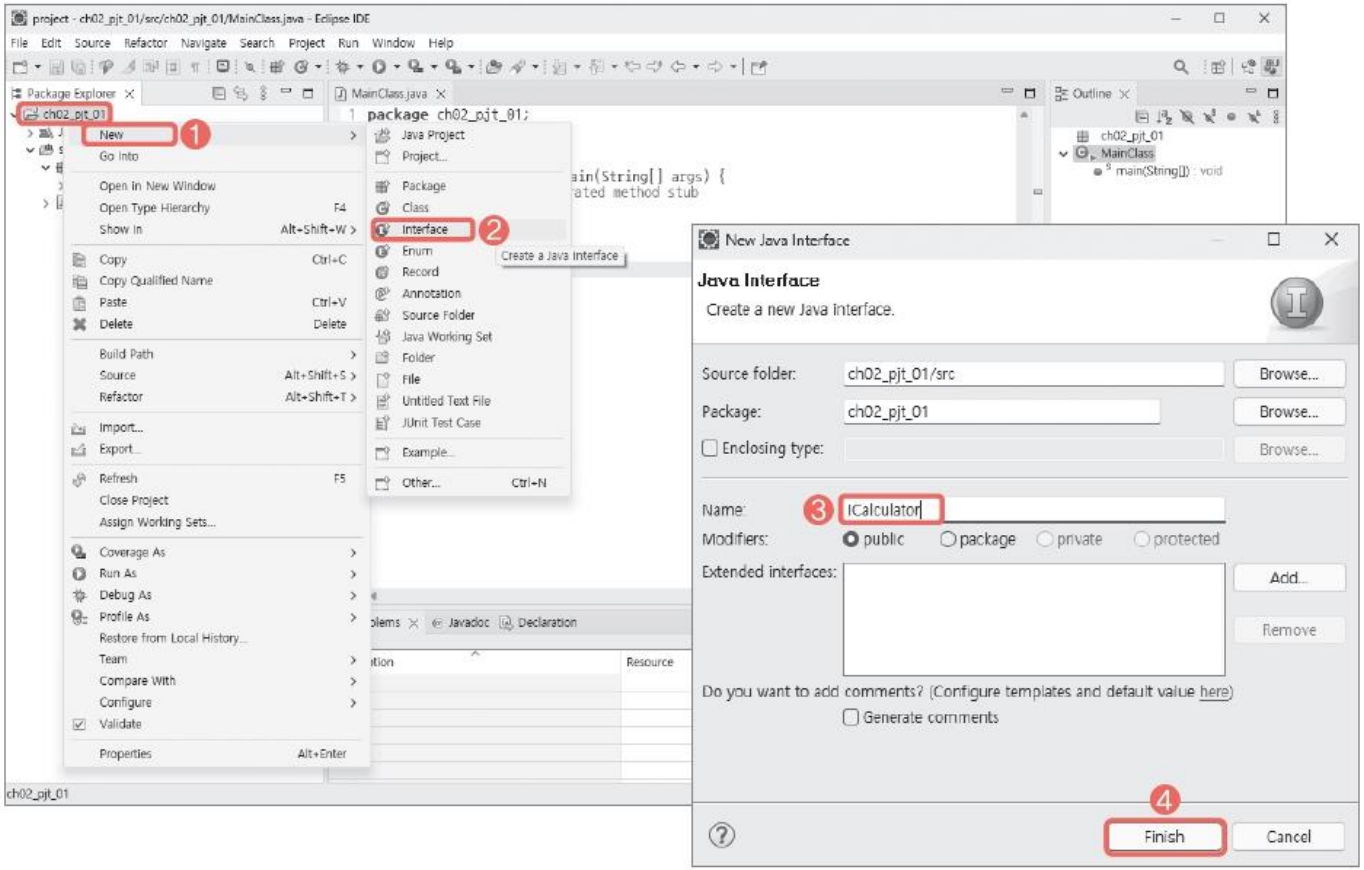
■ 클래스 생성

- ch02_pjt_01에서 마우스 오른쪽 버튼을 클릭하고 [New]-[Class]를 선택하기. Name에 생성할 클래스 이름 입력
  - ✓ MainClass를 생성할 때 public static void main (String[] args)에 체크하면 main() 메서드를 자동으로 생성해줌
  - ✓ 그 외의 클래스를 생성할 때는 체크하지 않음



■ 인터페이스 생성

- ch02_pjt_01에서 마우스 오른쪽 버튼을 클릭하고 [New]-[Interface]를 선택하고, Name에 ICalculator 입력



MainClass 클래스

코드 2-1 ch02_pjt_01\src\ch02_pjt_01\MainClass.java

```
01 package ch02_pjt_01;
02 public class MainClass {
03     public static void main(String[] args) {
04
05         MyCalculator calculator = new MyCalculator();
06         calculator.calAdd(10, 5);
07         calculator.calSub(10, 5);
08         calculator.calMul(10, 5);
09         calculator.calDiv(10, 5);
10
11     }
12 }
```

MyCalculator 클래스

코드 2-2 ch02_pjt_01\src\ch02_pjt_01\MyCalculator.java

```
01 package ch02_pjt_01;
02 public class MyCalculator {
03
04     public void calAdd(int fNum, int sNum) {
05         ICalculator calculator = new CalAdd();          // CalAdd 객체 생성
06         int value = calculator.doOperation(fNum, sNum); // 덧셈 실행
07         System.out.println("result : " + value);
08     }
09
10     public void calSub(int fNum, int sNum) {
11         ICalculator calculator = new CalSub();          // CalSub 객체 생성
12         int value = calculator.doOperation(fNum, sNum); // 뺄셈 실행
13         System.out.println("result : " + value);
14     }
15
16     public void calMul(int fNum, int sNum) {
17         ICalculator calculator = new CalMul();          // CalMul 객체 생성
18         int value = calculator.doOperation(fNum, sNum); // 곱셈 실행
19         System.out.println("result : " + value);
20     }
21
22     public void calDiv(int fNum, int sNum) {
23         ICalculator calculator = new CalDiv();          // CalDiv 객체 생성
24         int value = calculator.doOperation(fNum, sNum); // 나눗셈 실행
25         System.out.println("result : " + value);
26     }
27 }
```

ICalculator 인터페이스

코드 2-3 ch02_pjt_01\src\ch02_pjt_01\ICalculator.java

```
01 package ch02_pjt_01;
02 public interface ICalculator {
03     public int doOperation(int firstNum, int secondNum);
04 }
```

CalAdd 클래스

코드 2-4 ch02_pjt_01\src\ch02_pjt_01\CalAdd.java

```
01 package ch02_pjt_01;
02 public class CalAdd implements ICalculator {
03     @Override
04     public int doOperation(int firstNum, int secondNum) {
05         return firstNum + secondNum;
06     }
07 }
```

CalSub 클래스

코드 2-5 ch02_pjt_01\src\ch02_pjt_01\CalSub.java

```
01 package ch02_pjt_01;
02 public class CalSub implements ICalculator {
03     @Override
04     public int doOperation(int firstNum, int secondNum) {
05         return firstNum - secondNum;
06     }
07 }
```

CalMul 클래스

코드 2-6 ch02_pjt_01\src\ch02_pjt_01\CalMul.java

```
01 package ch02_pjt_01;
02 public class CalMul implements ICalculator {
03     @Override
04     public int doOperation(int firstNum, int secondNum) {
05         return firstNum * secondNum;
06     }
07 }
```

CalDiv 클래스

코드 2-7 ch02_pjt_01\src\ch02_pjt_01\CalDiv.java

```
01 package ch02_pjt_01;
02 public class CalDiv implements ICalculator {
03     @Override
04     public int doOperation(int firstNum, int secondNum) {
05         return secondNum != 0 ? (firstNum / secondNum) : 0;
06     }
07 }
```

실행 결과

```
result : 15
result : 5
result : 50
result : 2
```

DI(Dependency Injection)

- 의존성 주입이라고 함
- 의존성 주입이란 필요한(의존하는) 객체를 직접 생성하지 않고 외부에서 주입하는 방식을 의미함

의존의 개념

- 계산기 프로그램을 실행하면 main()([코드 2-2])에서 MyCalculator를 생성하고 calAdd(), calSub(), calMul(), calDiv()를 호출함
- 그리고 이 메서드들은 동일하게 연산에 필요한 객체를 직접 생성함
- MyCalculator에서 연산에 필요한 객체를 생성하는 것을 'MyCalculator는 CalAdd, CalSub, CalMul, CalDiv 객체를 이용한다'라고 함
- 즉 MyCalculator는 자신이 직접 연산하지 않고, 각각의 연산 객체들(CalAdd, CalSub, CalMul, CalDiv)에게 연산 업무를 전달함

■의존의 개념

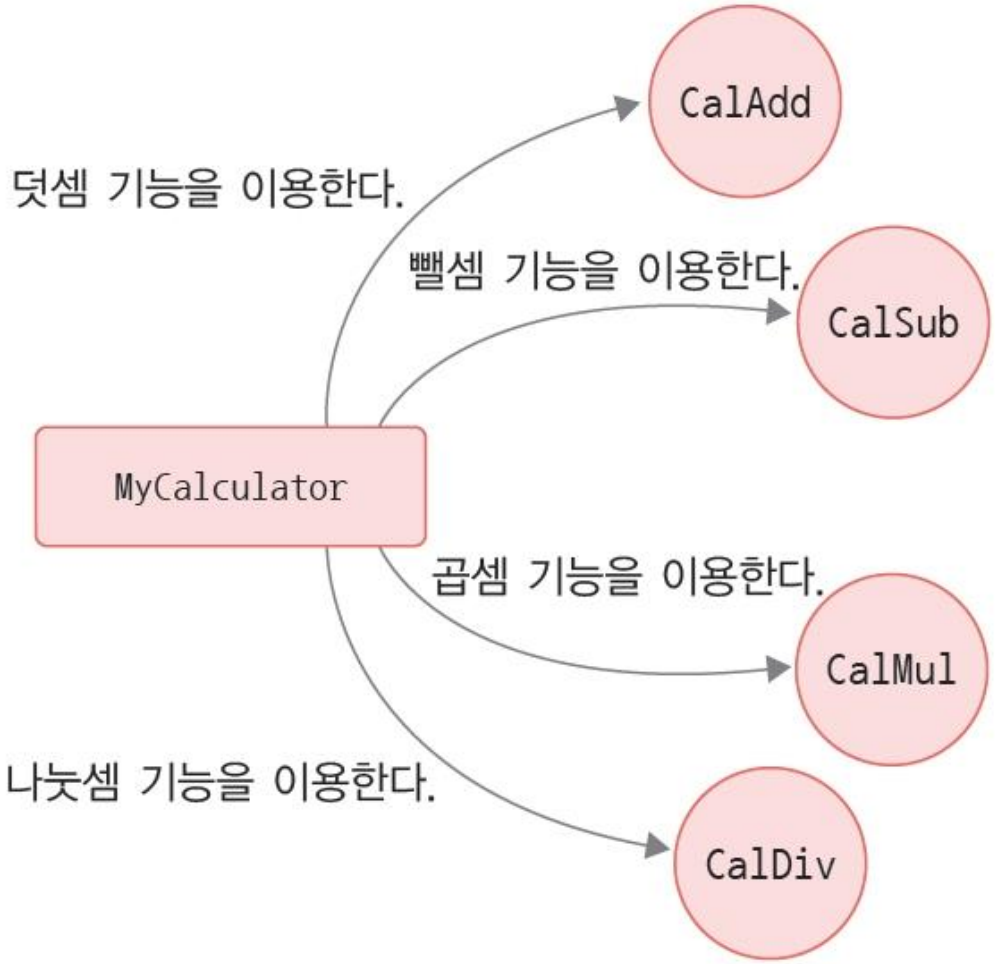


그림 2-5 MyCalculator에서 연산 업무를 전달하는 형태

의존의 개념

- MyCalculator가 다른 객체들을 이용한다'는 것을 다르게 표현해보기
- CalAdd 입장에서는 main() 함수가 MyCalculator에 지시한 덧셈 연산 업무를 MyCalculator를 대신해서 덧셈 연산을 처리하고 있음
- 즉, 'MyCalculator는 CalAdd에 의존한다'라고 할 수 있음

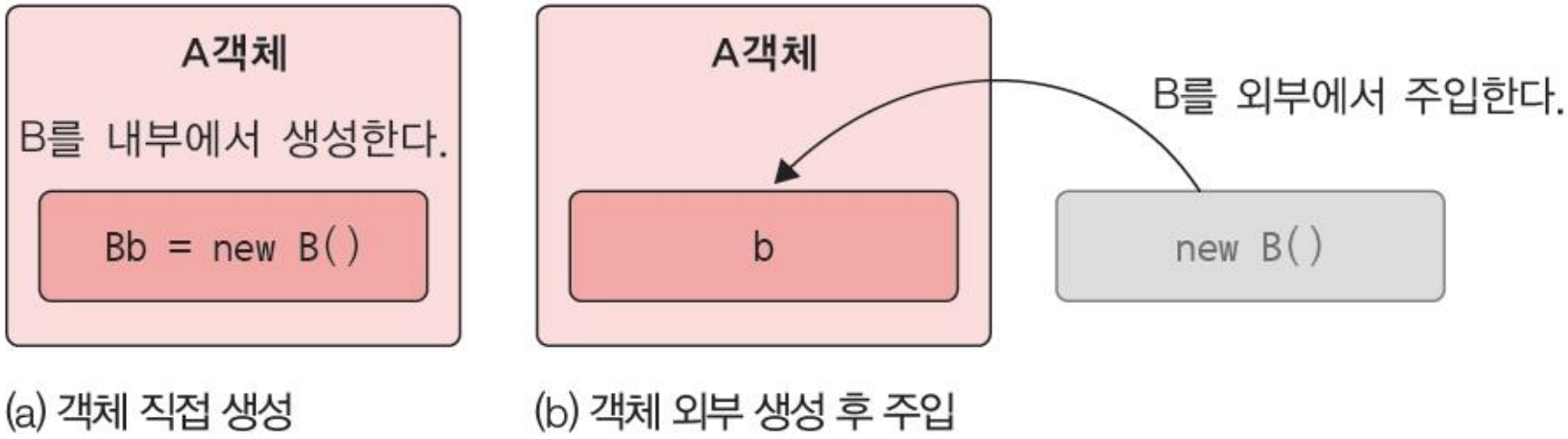


그림 2-6 필요한 객체를 내부 또는 외부에서 생성하여 주입함

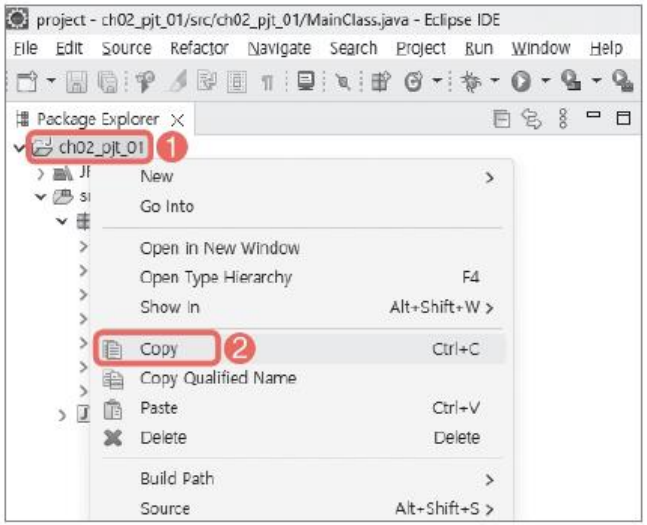
3. 계산기 프로그램을 DI 방식으로 변경하기

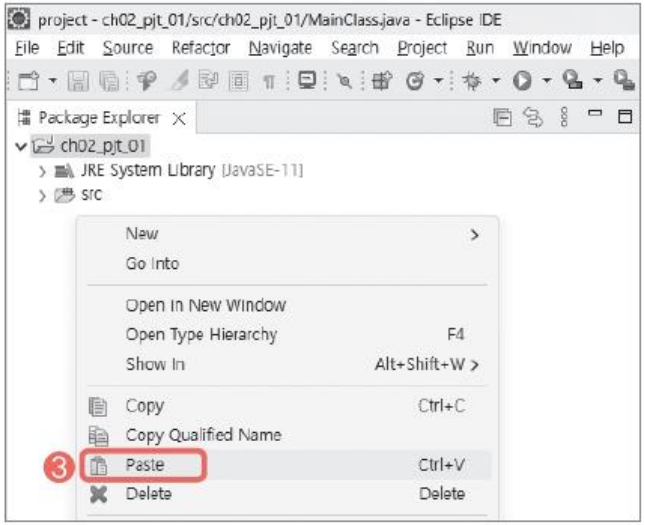
■ ch02_pjt_01 프로젝트를 외부에서 주입하는 방식으로 변경하기

- ch02_pjt_01을 ch02_pjt_02로 복사해서 CalAdd, CalSub, CalMul, CalDiv를 MyCalculator가 직접 생성하지 않고 외부에서 주입하는 방식으로 변경하기

■ ch02_pjt_02 생성하기

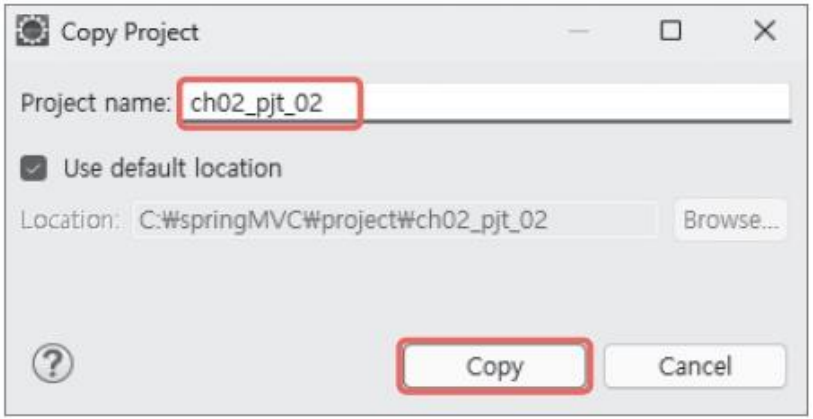
- 1. ch02_pjt_01에서 마우스 오른쪽 버튼을 클릭하고 [Copy]를 선택하고, Package Explorer의 흰색 공간에서 마우스 오른쪽 버튼을 클릭하고 [Paste]를 선택

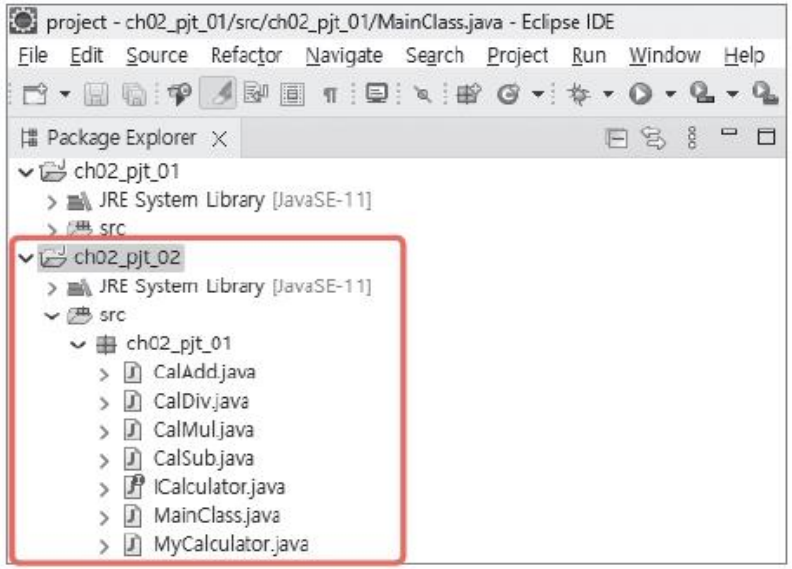




ch02_pjt_02 생성하기

- 2. [Copy Project] 창의 Project name에 ch02_pjt_02를 입력하고 <Copy> 클릭
- 3. ch02_pjt_02가 생성된 것을 확인하기





3. 계산기 프로그램을 DI 방식으로 변경하기

■ MyCalculator 클래스 수정하기

- MyCalculator가 의존하는 CalAdd, CalSub, CalMul, CalDiv를 MyCalculator가 직접 생성하지 않고 외부에서 주입하는 방식으로 변경하기 위해 수정함
- 이를 매개변수를 이용해서 연산에 필요한 객체를 외부에서 받으면 됨
- **calAdd( ), calSub(), calMul(), calDiv() 코드 수정하기**
  - ✓ MyCalculator 클래스가 CalAdd 클래스를 외부에서 주입받을 수 있도록 [코드 2-2]의 calAdd(), calSub(), calMul(), calDiv() 부분을 수정

■ 수정된 MyCalculator 클래스

코드 2-8 ch02_pjt_01\src\ch02_pjt_02\MyCalculator.java

```
01 package ch02_pjt_01;
02 public class MyCalculator {
03
04     public void calAdd(int fNum, int sNum, CalAdd calAdd) { // CalAdd 객체 주입
05         int value = calAdd.doOperation(fNum, sNum);      // 덧셈 실행
06         System.out.println("result : " + value);
07     }
08
09     public void calSub(int fNum, int sNum, CalSub calSub) { // CalSub 객체 주입
10         int value = calSub.doOperation(fNum, sNum);      // 뺄셈 실행
11         System.out.println("result : " + value);
12     }
13
14     public void calMul(int fNum, int sNum, CalMul calMul) { // CalMul 객체 주입
15         int value = calMul.doOperation(fNum, sNum);      // 곱셈 실행
16         System.out.println("result : " + value);
17     }
18
19     public void calDiv(int fNum, int sNum, CalDiv calDiv) { // CalDiv 객체 주입
20         int value = calDiv.doOperation(fNum, sNum);      // 나눗셈 실행
21         System.out.println("result : " + value);
22     }
23 }
```

MainClass 수정하기

- MyCalculator 수정이 끝났다면 MyCalculator를 사용하는 MainClass도 수정
- main()에서 MyCalculator를 사용할 때 예전과 달리 연산에 필요한 객체들(CalAdd, CalSub, CalMul, CalDiv)을 주입해야 함

코드 2-9 ch02_pjt_01\src\ch02_pjt_02\ MainClass.java

```
01 package ch02_pjt_01;
02 public class MainClass {
03     public static void main(String[] args) {
04
05         MyCalculator calculator = new MyCalculator();
06
07         calculator.calAdd(10, 5, new CalAdd());    // 객체 주입
08         calculator.calSub(10, 5, new CalSub());    // 객체 주입
09         calculator.calMul(10, 5, new CalMul());    // 객체 주입
10         calculator.calDiv(10, 5, new CalDiv());    // 객체 주입
11     }
12 }
```

■ MainClass 수정하기

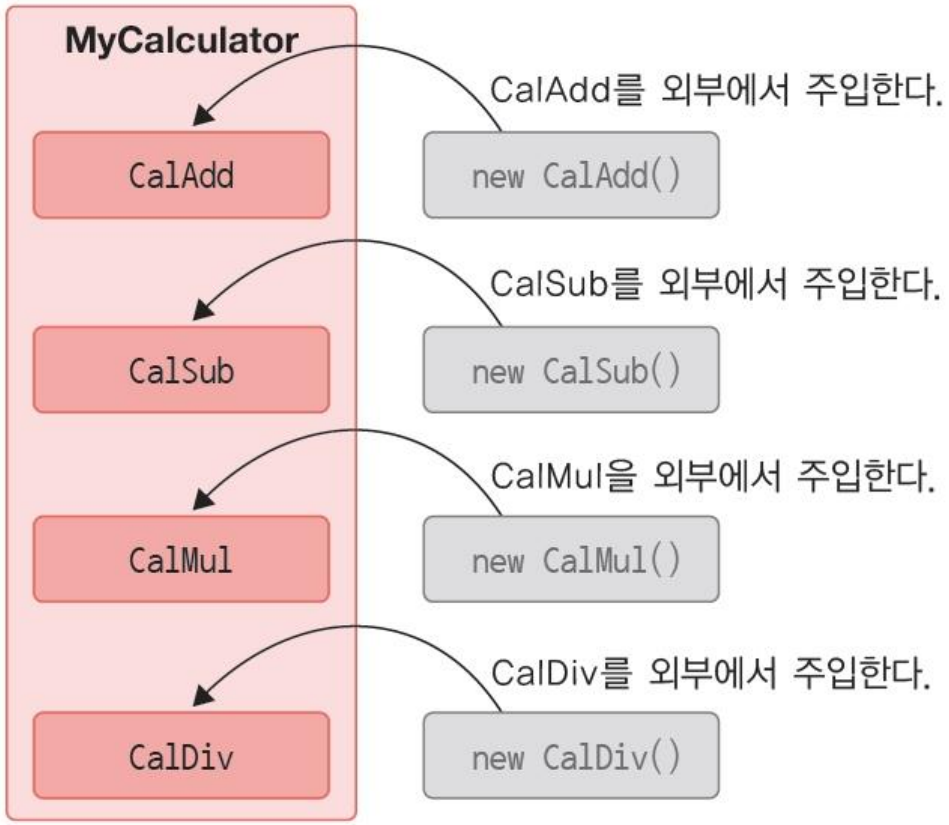


그림 2-7 연산에 필요한 객체를 외부에서 주입받는 형태의 MyCalculator 구조

■ 인터페이스를 활용하도록 수정하기

- MyCalculator가 필요한 객체를 외부에서 받는 코드는 변형하는데 인터페이스를 이용하지 않으니 동일한 코드를 계속 반복 작성해야 했음
- 인터페이스를 활용하여 MyCalculator가 CalAdd, CalSub, CalMul, CalDiv를 외부에서 받을 때 ICalculator를 이용하도록 MyCalculator와 MainClass를 최종 수정하기

코드 2-10 ch02_pjt_01\src\ch02_pjt_02\MyCalculator.java

```
01 package ch02_pjt_01;
02 public class MyCalculator {
03                                                    ICalculator 객체 주입
04     public void calculate(int fNum, int sNum, ICalculator calculator) {
05         int value = calculator.doOperation(fNum, sNum);    // 연산 실행
06         System.out.println("result : " + value);
07     }
08 }
```

■ 인터페이스를 활용하도록 수정하기

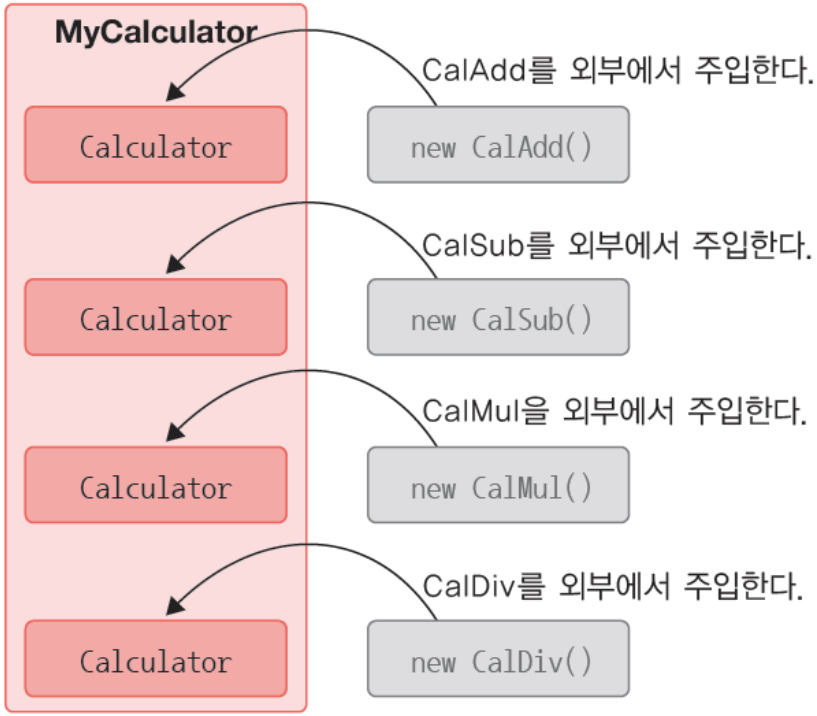


그림 2-8 ICalculator를 이용하여 유연해진 프로그램

■ 인터페이스를 활용하도록 수정하기

- MyCalculator를 사용하도록 MainClass의 main() 메서드도 수정하기

코드 2-11 ch02_pjt_01\src\ch02_pjt_02\MyCalculator.java

```
01 package ch02_pjt_01;
02 public class MainClass {
03     public static void main(String[] args) {
04 
05         MyCalculator calculator = new MyCalculator();
06         calculator.calculate(10, 5, new CalAdd());
07         calculator.calculate(10, 5, new CalSub());
08         calculator.calculate(10, 5, new CalMul());
09         calculator.calculate(10, 5, new CalDiv());
10     }
11 }
```

IoC(inversion of control)

- 제어의 역전이라고 함
- 프로그램의 제어권을 개발자가 컨트롤하는 것이 아니라 외부에서 컨트롤하는 방식을 의미함
- 말 그대로 제어의 주체가 개발자에서 스프링으로 바뀐 것
- IoC를 이해하기 위해 ch02_pjt_02를 복사해서 ch02_pjt_03 생성하기

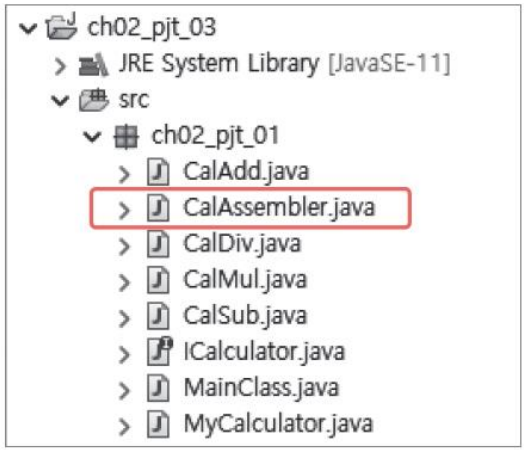
5. 계산기 프로젝트를 IoC 방식으로 변경하기

■ CalAssembler 클래스

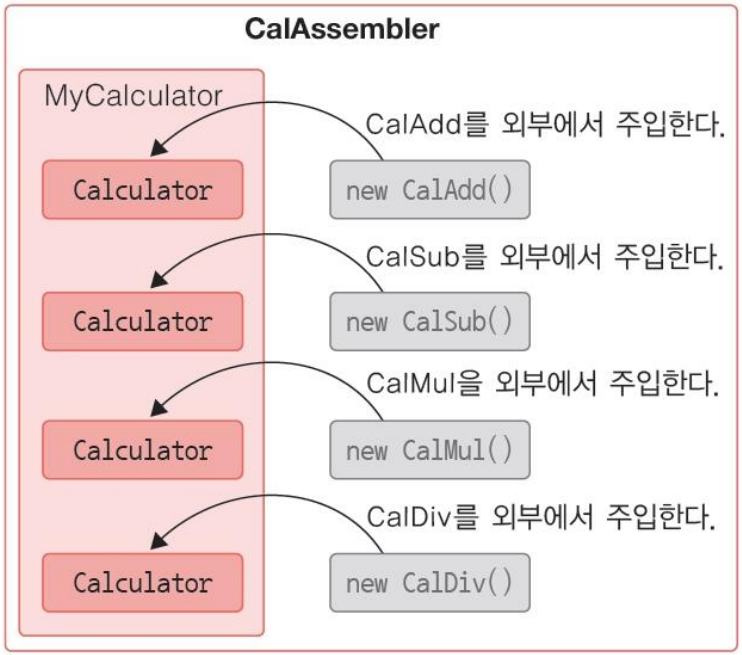
- ch02_pjt_03에서 수정할 부분은 main()임
- main()의 역할
  - ✓ 프로그램의 시작을 나타냄(JVM이 가장 먼저 찾음)
  - ✓ 따라서 지금처럼 MyCalculator, CalAdd, CalSub, CalMul, CalDiv 등과 같은 프로그램에 필요한 모든 객체를 main()에서 생성한다는 것은 main()에 과다한 업무를 부여하는 것과 같음
- 프로그램 실행에 필요한 객체는 별도의 클래스에서 생성하도록 수정함
  - ✓ CalAssembler 클래스의 역할 : 프로그램에서 사용하는 객체들을 생성하고 주입함

■ CalAssembler 클래스

- 1. CalAssembler 클래스를 만들기



(a) 프로젝트에 추가된 CalAssembler 클래스



(b) 객체를 생성하고 주입하는 CalAssembler 클래스

그림 2-9 CalAssembler 클래스의 역할

CalAssembler 클래스

- 2. 프로그램에서 사용하는 객체들을 생성하고 주입하는 CalAssembler 코딩하기

코드 2-12 ch02_pjt_01\src\ch02_pjt_03\CalAssembler.java

```
01 package ch02_pjt_01;
02 public class CalAssembler {
03     MyCalculator calculator;
04     CalAdd calAdd;
05     CalSub calSub;
06     CalMul calMul;
07     CalDiv calDiv;
08
09     public CalAssembler() {
10         calculator = new MyCalculator();
11         calAdd = new CalAdd();
12         calSub = new CalSub();
13         calMul = new CalMul();
14         calDiv = new CalDiv();
15
16         assemble();
17     }
18     public void assemble() {
19         calculator.calculate(10, 5, calAdd);
20         calculator.calculate(10, 5, calSub);
21         calculator.calculate(10, 5, calMul);
22         calculator.calculate(10, 5, calDiv);
23     }
24 }
```

CalAssembler 클래스

- 3. 이제 main() 메서드에서 하던 일을 CalAssembler에서 하게 됨
  - ✓ CalAssembler는 생성자에서 프로그램에 필요한 객체들(MyCalculator, CalAdd, CalSub, CalMul, CalDiv)을 모두 생성하고 연산을 자동으로 실행(assemble())함
  - ✓ 이에 따라 MainClass의 main() 메서드를 다음과 같이 수정하기

코드 2-13 ch02_pjt_01\src\ch02_pjt_03\MainClass.java

```
01 package ch02_pjt_01;
02 public class MainClass {
03     public static void main(String[] args) {
04         new CalAssembler();
05     }
06 }
```

CalAssembler 클래스

- [코드 2-13] 4행에서 볼 수 있듯이 main() 메서드에서는 CalAssembler 객체만 생성하도록 변경됨
- IoC 컨테이너: CalAssembler와 같이 객체를 생성하고 조립하는 특별한 공간
- 빈(Bean): IoC 컨테이너의 객체
- 다시 말해 스프링의 IoC 컨테이너는 빈을 생성하고 필요한 곳에 주입(DI)하는 특별한 공간임

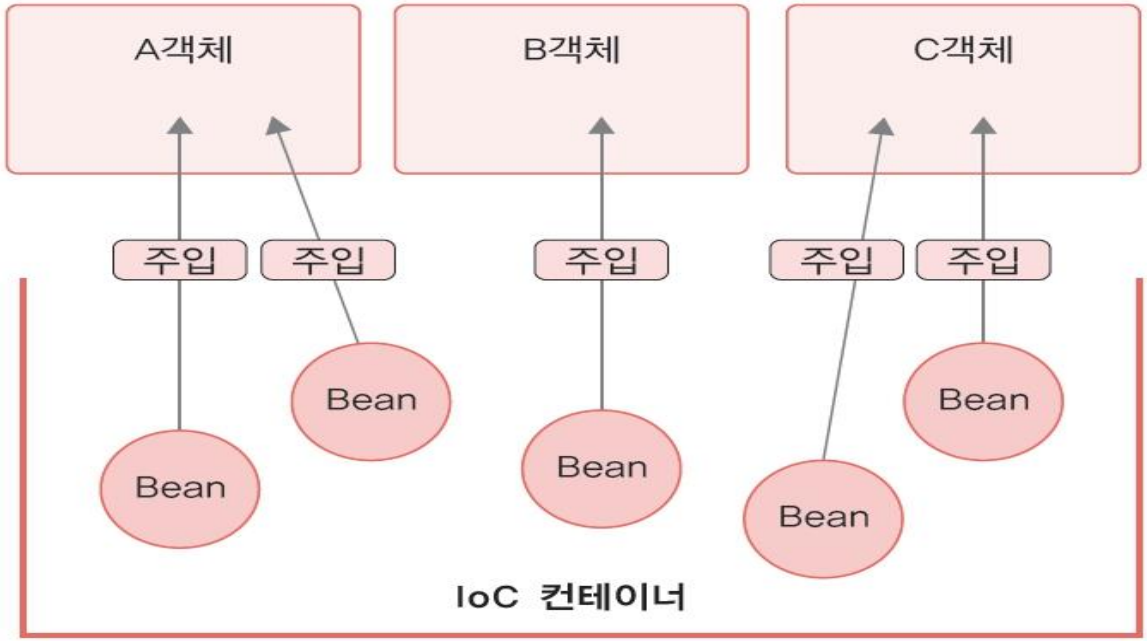


그림 2-10 빈을 생성하고 필요한 곳에 주입하는 IoC 컨테이너