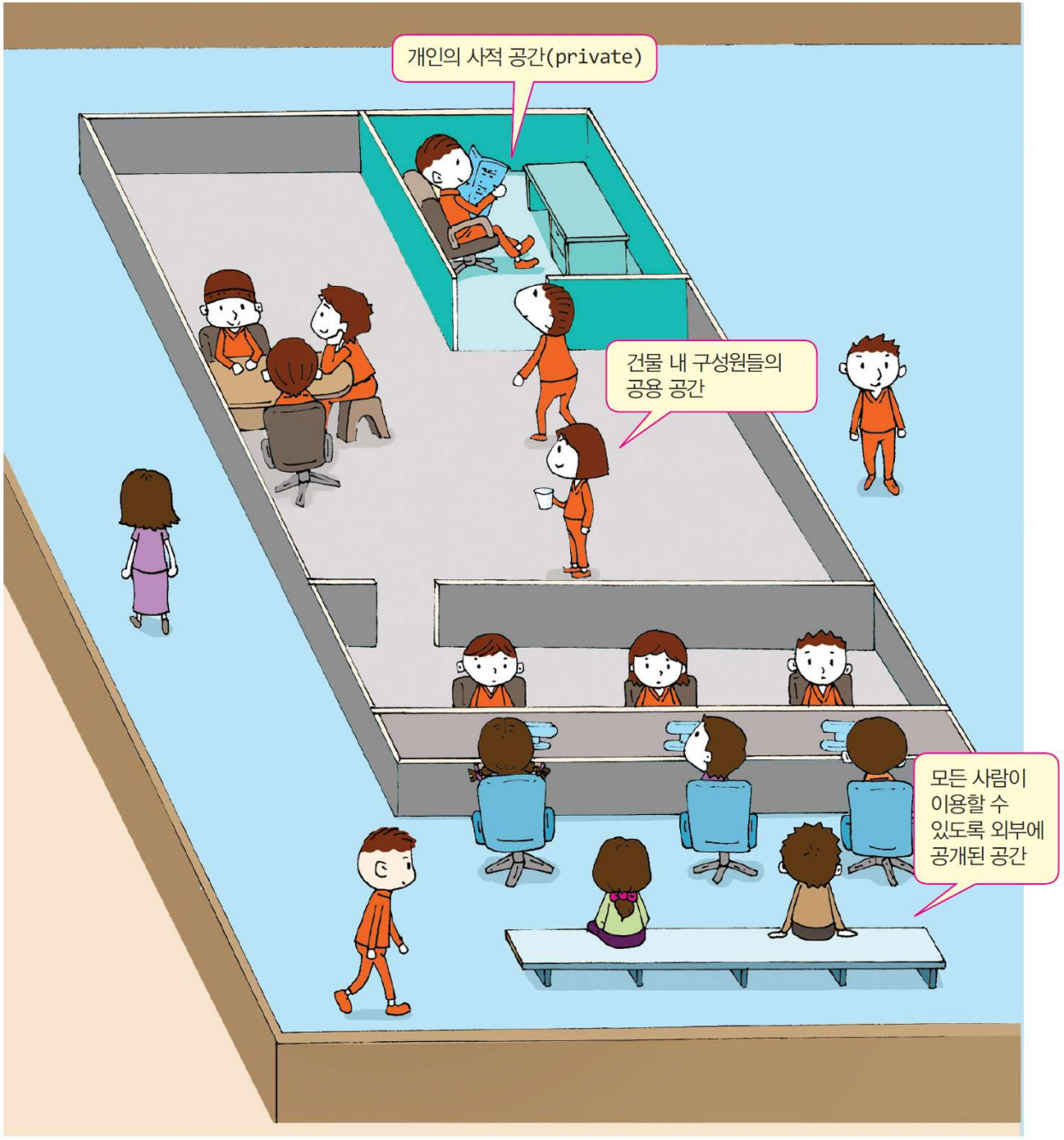
4

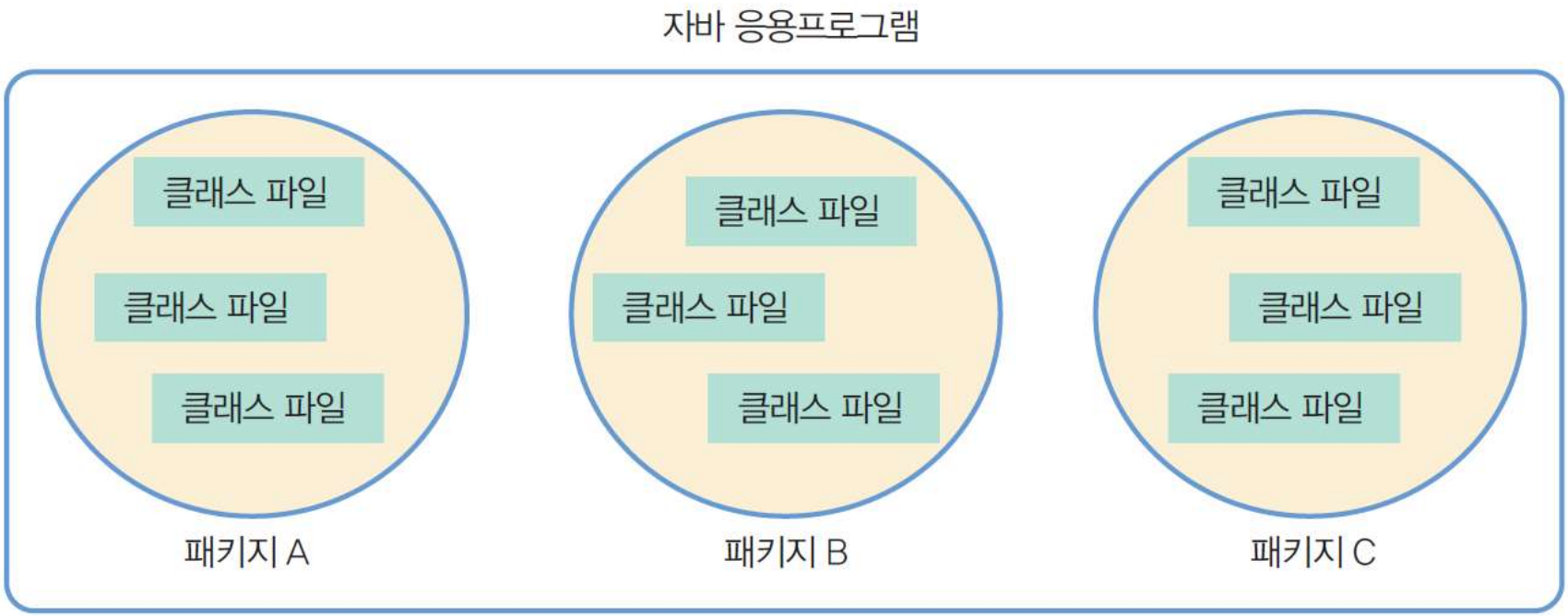
클래스와 객체 (2)

접근 지정자 이해



자바의 패키지 개념

- 패키지
  - 관련 있는 클래스 파일(컴파일된 .class)을 저장하는 디렉터리
  - 자바 응용프로그램은 하나 이상의 패키지로 구성

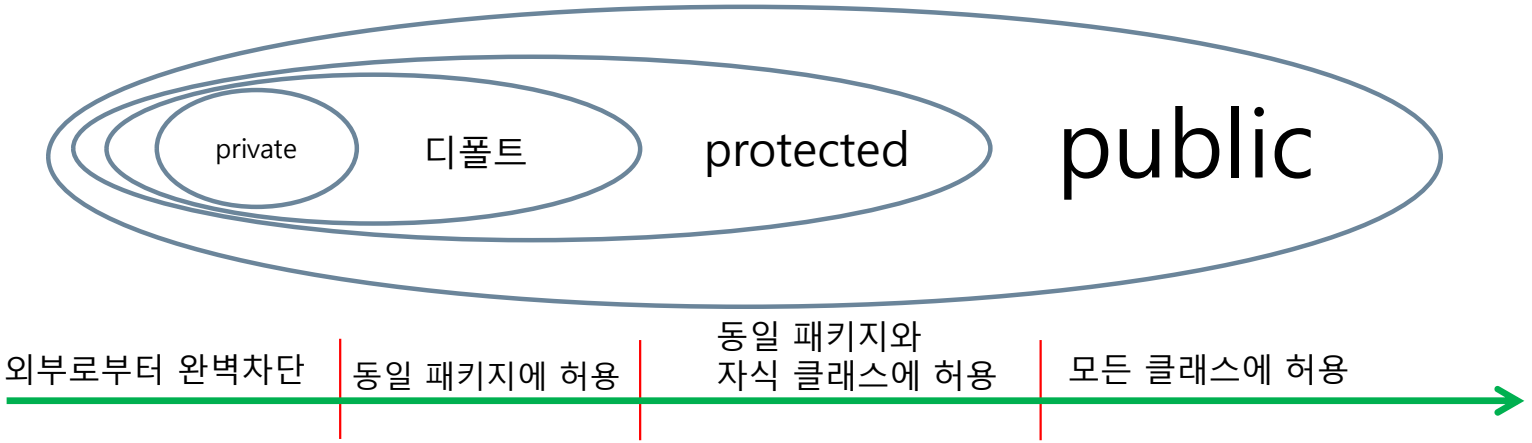


접근 지정자

- 자바의 접근 지정자
  - 4가지
    - private, protected, public, 디폴트(접근지정자 생략)

- 접근 지정자의 목적
  - 클래스나 일부 멤버를 공개하여 다른 클래스에서 접근하도록 허용
  - 객체 지향 언어의 캡슐화 정책은 멤버를 보호하는 것
    - 접근 지정은 캡슐화에 묶인 보호를 일부 해제할 목적

- 접근 지정자에 따른 클래스나 멤버의 공개 범위

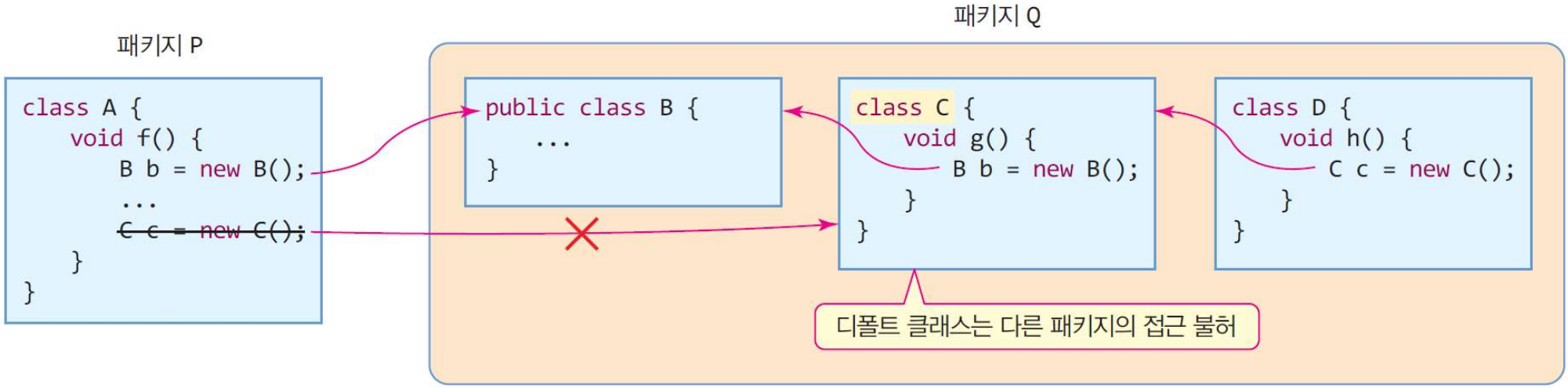


클래스 접근 지정

- 클래스 접근지정
  - 다른 클래스에서 사용하도록 허용할 지 지정
  - public 클래스
    - 다른 모든 클래스에게 접근 허용
  - 디폴트 클래스(접근지정자 생략)
    - package-private라고도 함
    - 같은 패키지의 클래스에만 접근 허용

```
public class World { // public 클래스
.....
}
```

```
class Local { // 디폴트 클래스
.....
}
```



public 클래스와 디폴트 클래스의 접근 사례

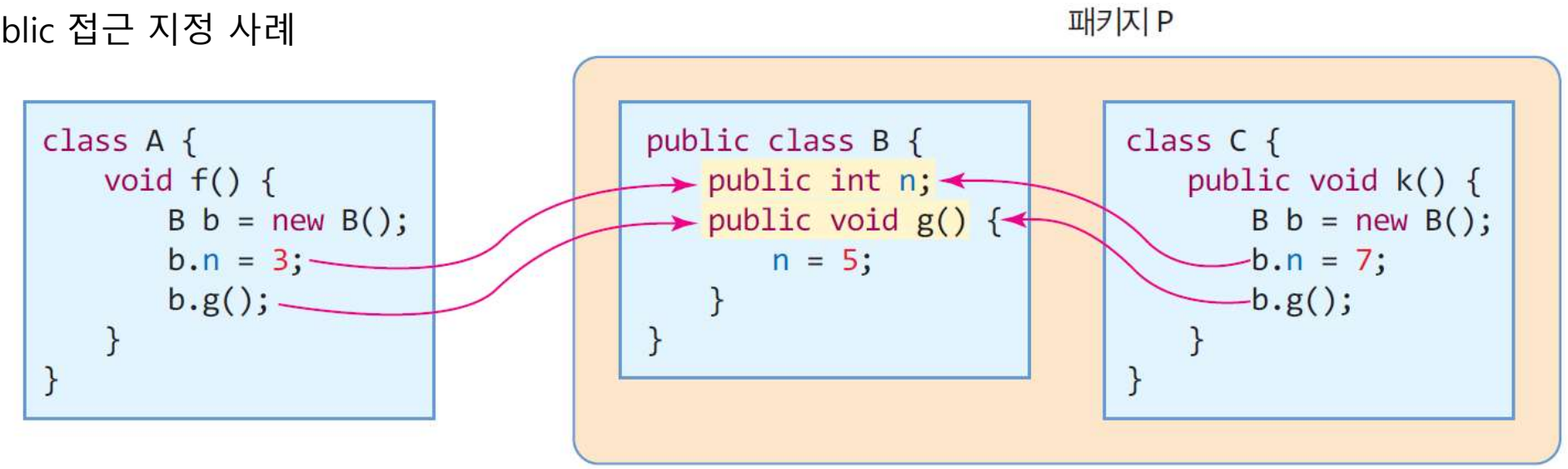
멤버 접근 지정

- public 멤버
  - 패키지에 관계 없이 모든 클래스에게 접근 허용
- private 멤버
  - 동일 클래스 내에만 접근 허용
  - 상속 받은 서브 클래스에서 접근 불가
- protected 멤버
  - 같은 패키지 내의 다른 모든 클래스에게 접근 허용
  - 상속 받은 서브 클래스는 다른 패키지에 있어도 접근 가능
- 디폴트(default) 멤버
  - 같은 패키지 내의 다른 클래스에게 접근 허용

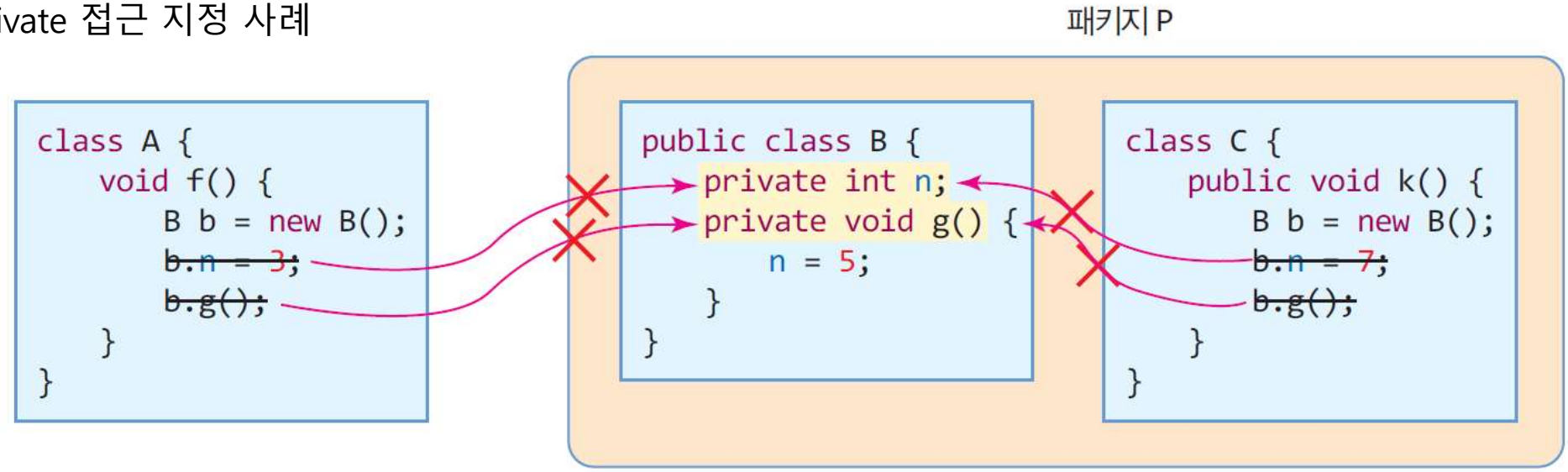
멤버에 접근하는 클래스	멤버의 접근 지정자			
	private	디폴트 접근 지정	protected	public
같은 패키지의 클래스	×	○	○	○
다른 패키지의 클래스	×	×	×	○
접근 가능 영역	클래스 내	동일 패키지 내	동일 패키지와 자식 클래스	모든 클래스

멤버 접근 지정자의 이해

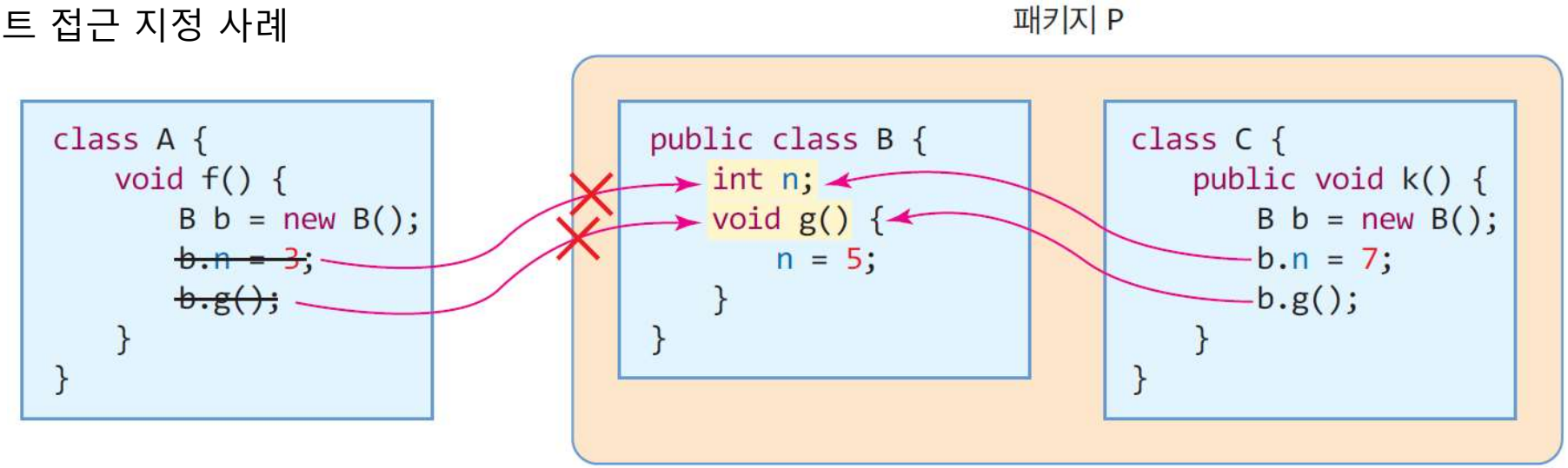
public 접근 지정 사례



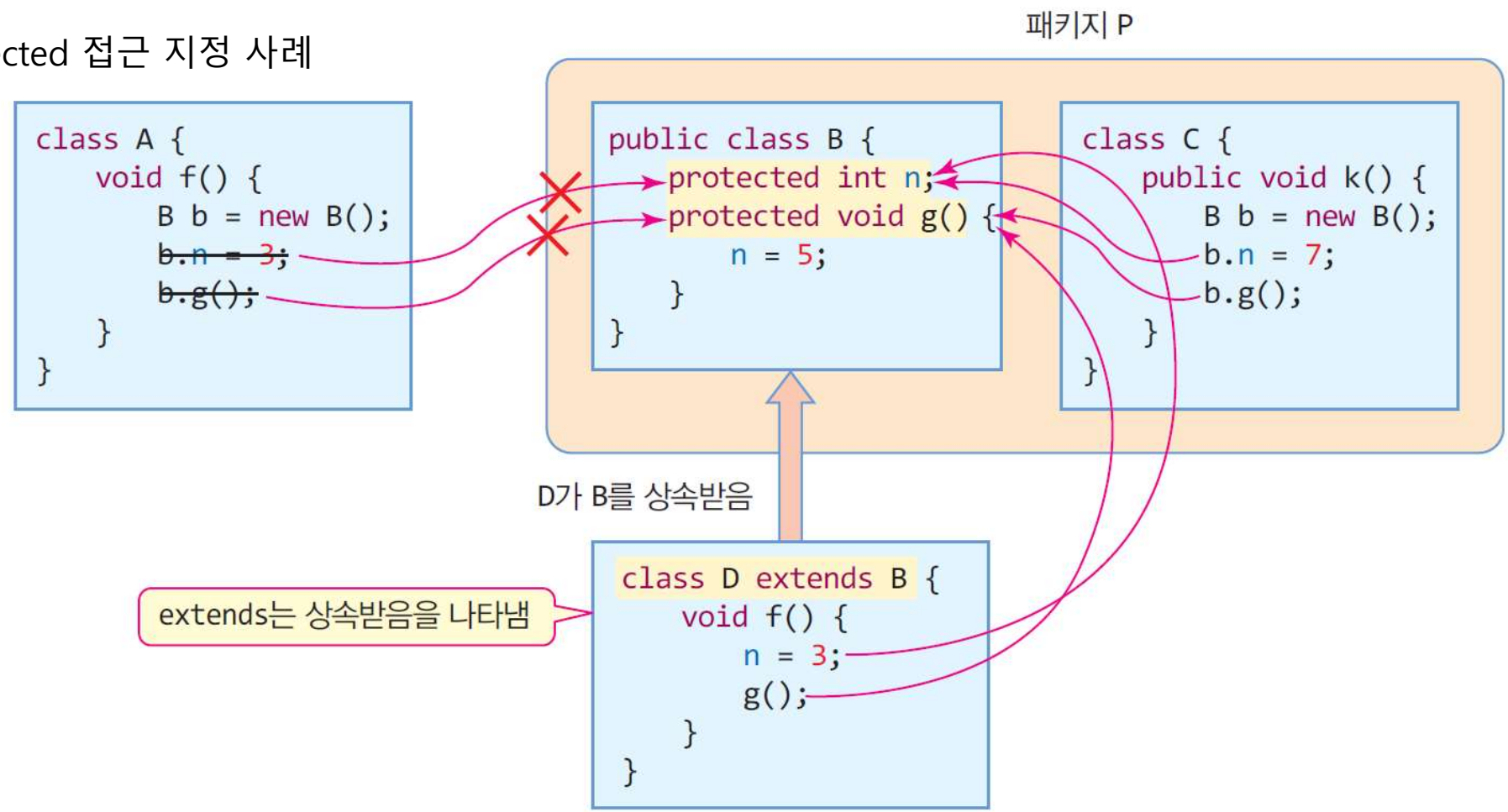
private 접근 지정 사례



디폴트 접근 지정 사례



protected 접근 지정 사례



예제 4-10 : 멤버의 접근 지정자

다음 코드의 두 클래스 Sample과 AccessEx 클래스는 동일한 패키지에 저장된다.
컴파일 오류를 찾아 내고 이유를 설명하라.

```
class Sample {
   public int a;
   private int b;
   int c;
}

public class AccessEx {
   public static void main(String[] args) {
      Sample aClass = new Sample();
      aClass.a = 10;
      aClass.b = 10;
      aClass.c = 10;
   }
}
```

- Sample 클래스의 a와 c는 각각 public, default 지정자로 선언이 되었으므로, 같은 패키지에 속한 AccessEx 클래스에서 접근 가능
- b는 private으로 선언이 되었으므로 AccessEx 클래스에서 접근 불가능

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
   The field Sample.b is not visible
   at AccessEx.main(AccessEx.java:11)
```

static 이해를 위한 그림

눈은 각 사람마다 있고 공기는 모든 사람이 소유(공유)한다

사람은 모두 각각 눈을 가지고 태어난다.

세상에는 이미 공기가 있으며 태어난 사람은
모두 공기를 공유한다.
그리고 공기 역시 각 사람의 것이다.

static 멤버와 non-static 멤버

- non-static 멤버의 특성 (객체에 종속)
  - 공간적 특성 – 멤버들은 객체마다 독립적으로 별도 존재
    - 인스턴스 멤버라고도 부름
  - 시간적 특성 – 필드와 메소드는 객체 생성 후 비로소 사용 가능
  - 비공유 특성 - 멤버들은 다른 객체에 의해 공유되지 않고 배타적

- static 멤버 (클래스에 종속)
  - 객체마다 생기는 것이 아님
  - 클래스당 하나만 생성됨
    - 클래스 멤버라고도 부름
  - 객체를 생성하지 않고 사용가능
  - 특성
    - 공간적 특성 - static 멤버들은 클래스 당 하나만 생성
    - 시간적 특성 - static 멤버들은 클래스가 로딩될 때 공간 할당.
    - 공유의 특성 - static 멤버들은 동일한 클래스의 모든 객체에 의해 공유

```
class StaticSample {
   int n;                  // non-static 필드
   void g() {...}          // non-static 메소드

   static int m;           // static 필드
   static void f() {...}   // static 메소드
}
```

non-static 멤버와 static 멤버의 차이

	non-static 멤버	static 멤버

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

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

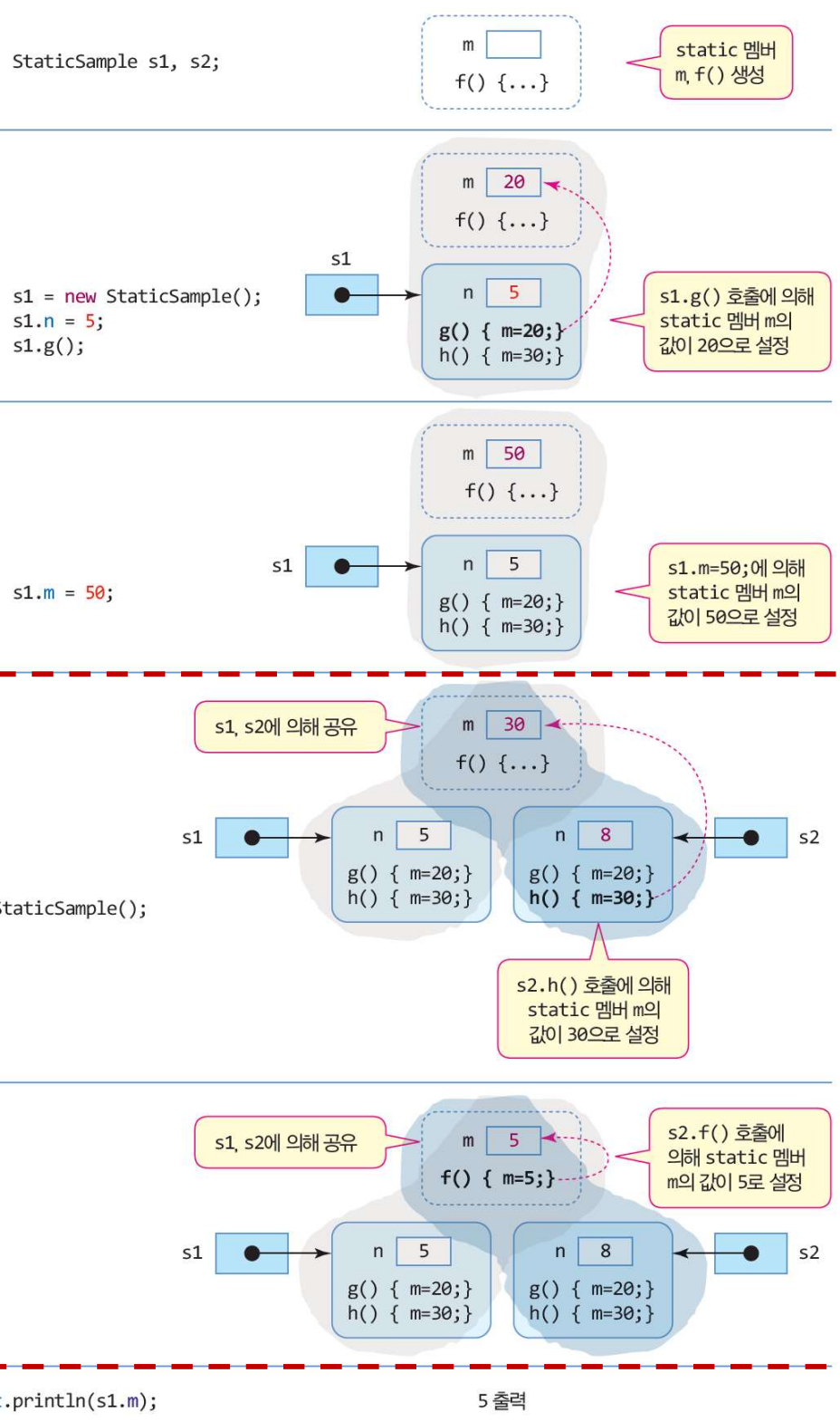
static 멤버를 객체의 멤버로 접근하는 사례

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

public class Ex {
    public static void main(String[] args) {
        StaticSample s1, s2;
        s1 = new StaticSample();
        s1.n = 5;
        s1.g();
        s1.m = 50; // static
        s2 = new StaticSample();
        s2.n = 8;
        s2.h();
        s2.f(); // static
        System.out.println(s1.m);
    }
}
```

실행 결과

```
5
```



static 멤버를 클래스 이름으로 접근하는 사례

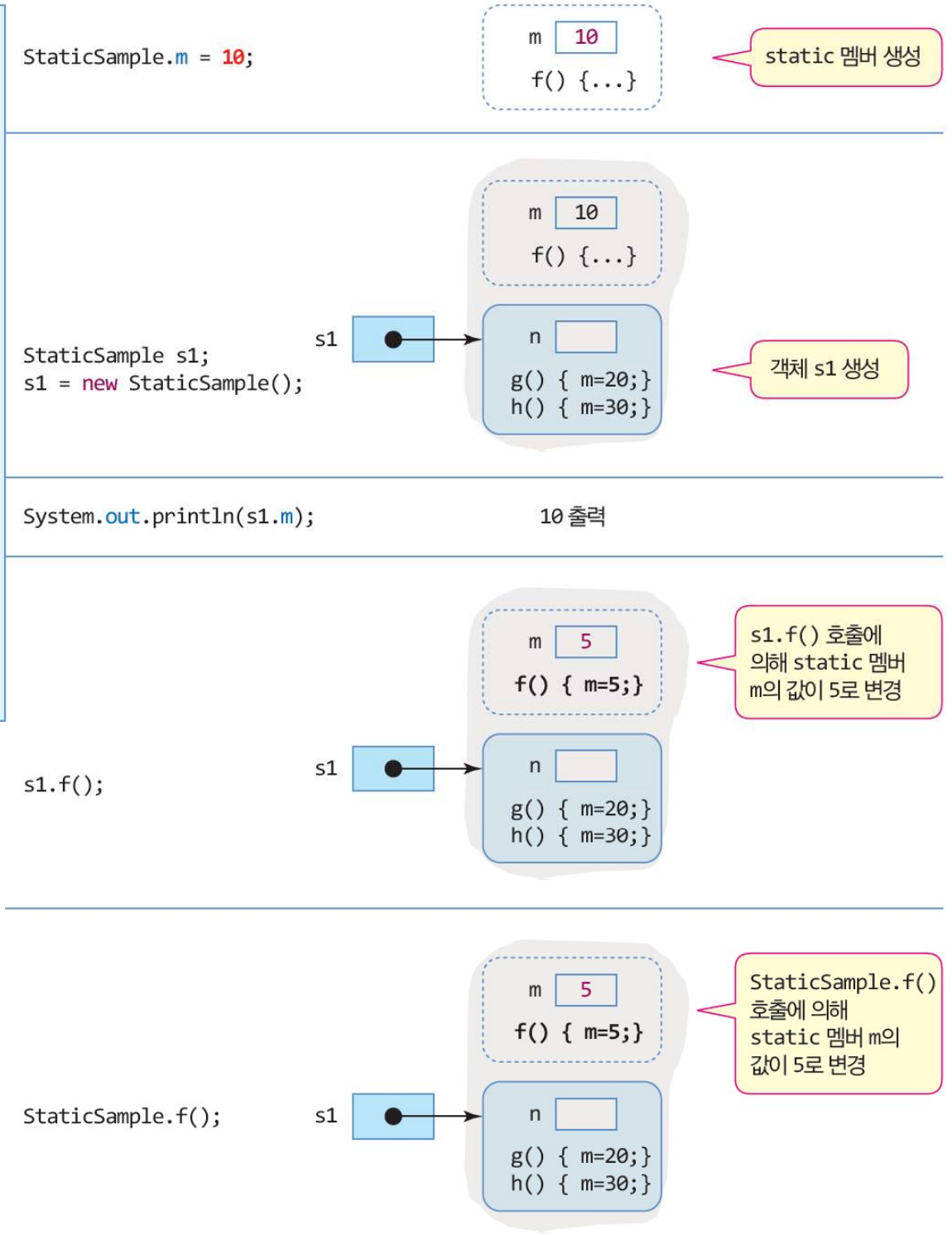
```
class StaticSample {
    public int n;
    public void g() {
        m = 20;
    }
    public void h() {
        m = 30;
    }
    public static int m;
    public static void f() {
        m = 5;
    }
}

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

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

실행 결과

```
10
```



static의 활용

1. 전역 변수와 전역 함수를 만들 때 활용

- 전역변수나 전역 함수는 static으로 클래스에 작성
- static 멤버를 가진 클래스 사례
  - Math 클래스 : java.lang.Math
    - 모든 필드와 메소드가 *public static*으로 선언
    - 다른 모든 클래스에서 사용할 수 있음

```
public class Math {
   public static int abs(int a);
   public static double cos(double a);
   public static int max(int a, int b);
   public static double random();
   ...
}
```

// 잘못된 사용법

```
Math m = new Math(); // Math() 생성자는 private
int n = m.abs(-5);
```

// 바른 사용법

```
int n = Math.abs(-5);
```

2. 공유 멤버를 작성할 때

- static 필드나 메소드는 하나만 생성. 클래스의 객체들 공유

예제 4-11 : static 멤버를 가진 Calc 클래스 작성

전역 함수로 작성하고자 하는 abs, max, min의 3개 함수를 static 메소드로 작성하고 호출하는 사례를 보여라.

```
class Calc {
   public static int abs(int a) { return a>0?a:-a; }
   public static int max(int a, int b) { return (a>b)?a:b; }
   public static int min(int a, int b) { return (a>b)?b:a; }
}

public class CalcEx {
   public static void main(String[] args) {
      System.out.println(Calc.abs(-5));
      System.out.println(Calc.max(10, 8));
      System.out.println(Calc.min(-3, -8));
   }
}
```

```
5
10
-8
```

static 메소드의 제약 조건 1

- static 메소드는 non-static 멤버 접근할 수 없음
  - 객체가 생성되지 않은 상황에서도 static 메소드는 실행될 수 있기 때문에, non-static 메소드와 필드 사용 불가
  - 반대로, non-static 메소드는 static 멤버 사용 가능

```
class StaticMethod {
    int n;
    void f1(int x) {n = x;} // 정상
    void f2(int x) {m = x;} // 정상

    static int m;
    static void s1(int x) {n = x;} // 컴파일 오류. static 메소드는 non-static 필드 사용 불가
    static void s2(int x) {f1(3);} // 컴파일 오류. static 메소드는 non-static 메소드 사용 불가

    static void s3(int x) {m = x;} // 정상. static 메소드는 static 필드 사용 가능
    static void s4(int x) {s3(3);} // 정상. static 메소드는 static 메소드 호출 가능
}
```

오류

오류

static 메소드의 제약 조건 2

- static 메소드는 this 사용불가
  - static 메소드는 객체가 생성되지 않은 상황에서도 호출이 가능하므로, 현재 객체를 가리키는 this 레퍼런스 사용할 수 없음

```
class StaticAndThis {
    int n;
    static int m;
    void f1(int x) {this.n = x;}
    void f2(int x) {this.m = x;} // non-static 메소드에서는 static 멤버 접근 가능
    static void s1(int x) {this.n = x;} // 컴파일 오류. static 메소드는 this 사용 불가
    static void s2(int x) {this.m = x;} // 컴파일 오류. static 메소드는 this 사용 불가
}
```

오류

오류

예제 4-12 : static을 이용한 환율 계산기

static 멤버를 이용하여 달러와 원화를 변환 해주는 환율 계산기를 만들어보자.

```
class CurrencyConverter {
    private static double rate; //  한국 원화에 대한 환율
    public static double toDollar(double won) {
        return won/rate; // 한국 원화를 달러로 변환
    }
    public static double toKWR(double dollar) {
        return dollar * rate; // 달러를 한국 원화로 변환
    }
    public static void setRate(double r) {
        rate = r; // 환율 설정. KWR/$1
    }
}
public class StaticMember {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("환율(1달러)>> ");
        double rate = scanner.nextDouble();
        CurrencyConverter.setRate(rate); // 미국 달러 환율 설정
        System.out.println("백만원은 $" + CurrencyConverter.toDollar(1000000) +"입니다.");
        System.out.println("$100는 " + CurrencyConverter.toKWR(100) + "원입니다.");
        scanner.close();
    }
}
```

```
환율(1달러)>> 1121
백만원은 $892.0606601248885입니다.
$100는 112100.0원입니다.
```

final 클래스와 메소드

- final 클래스 - 클래스 상속 불가

```
final class FinalClass {
        .....
}
class SubClass extends FinalClass { // 컴파일 오류. FinalClass 상속 불가
        .....
}
```

- final 메소드 - 오버라이딩 불가

```
public class SuperClass {
        protected final int finalMethod() { ... }
}

class SubClass extends SuperClass { // SubClass가 SuperClass 상속
        protected int finalMethod() { ... } // 컴파일 오류, 오버라이딩 할 수 없음
}
```

final 필드

- final 필드, 상수 선언
  - 상수를 선언할 때 사용

```
class SharedClass {
        public static final double PI = 3.14;
}
```

  - 상수 필드는 선언 시에 초기 값을 지정하여야 한다
  - 상수 필드는 실행 중에 값을 변경할 수 없다


```
public class FinalFieldClass {
        final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정

        void f() {
                int [] intArray = new int [ROWS]; // 상수 활용
                ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경할 수 없다.
        }
}
```