접근 제한자

접근 제한자

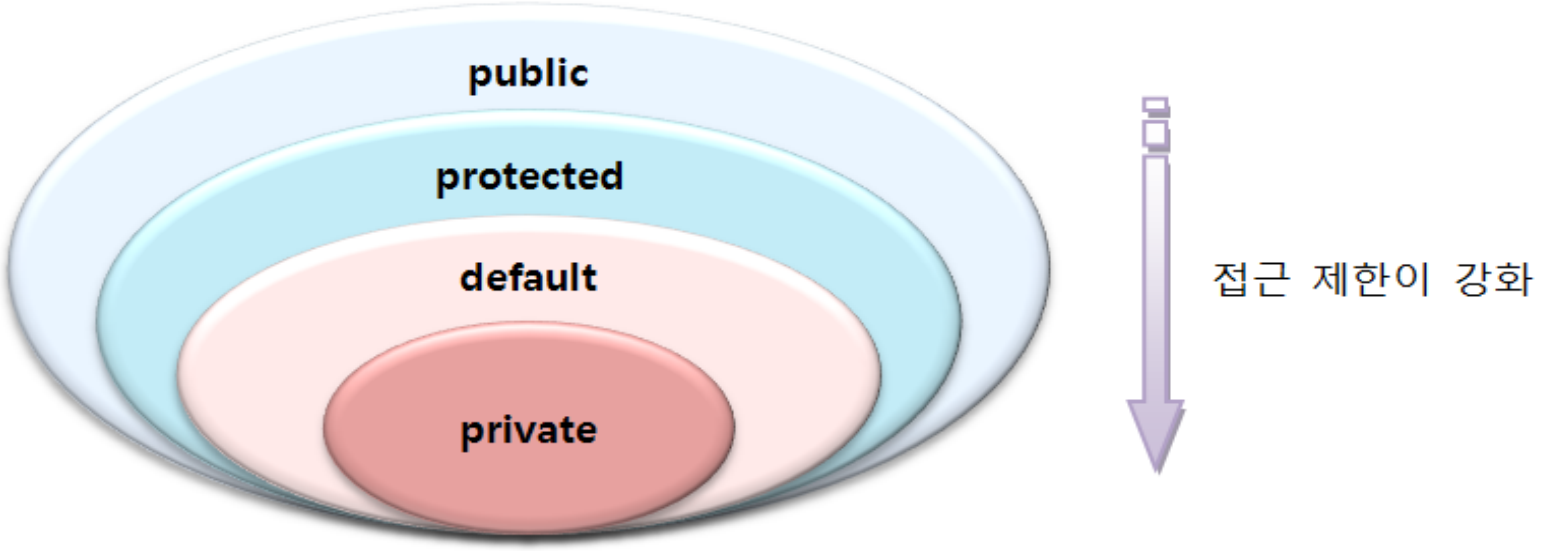
❖ 접근 제한자(Access Modifier)

- ㅇ 클래스 및 클래스의 구성 멤버에 대한 접근을 제한하는 역할
  - 다른 패키지에서 클래스를 사용하지 못하도록 (클래스 제한)
  - 클래스로부터 객체를 생성하지 못하도록 (생성자 제한)
  - 특정 필드와 메소드를 숨김 처리 (필드와 메소드 제한)

접근 제한자

❖ 접근 제한자(Access Modifier)

- ㅇ 접근 제한자의 종류

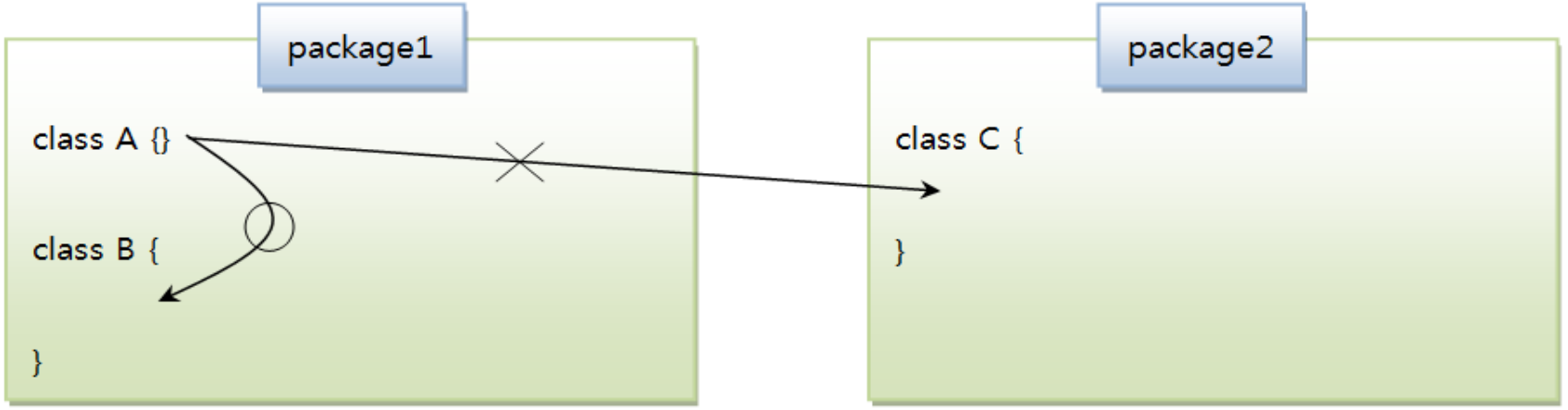


접근 제한	적용 대상	접근할 수 없는 클래스
public	클래스, 필드, 생성자, 메소드	없음
protected	필드, 생성자, 메소드	자식 클래스가 아닌 다른 패키지에 소속된 클래스
default	클래스, 필드, 생성자, 메소드	다른 패키지에 소속된 클래스
private	필드, 생성자, 메소드	모든 외부 클래스

접근 제한자

❖ 클래스의 접근 제한

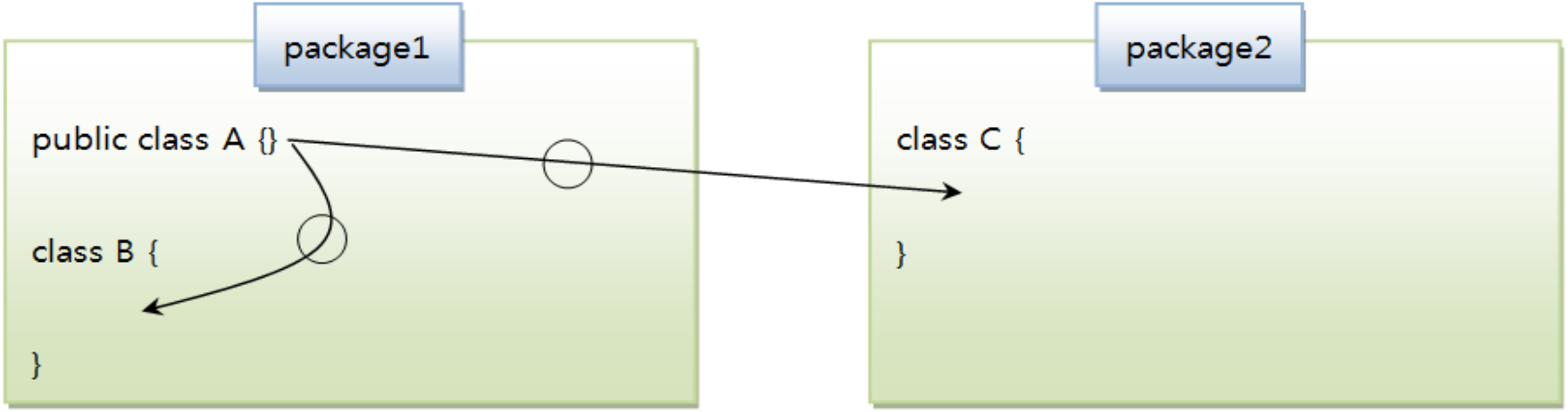
- default
  - 클래스 선언할 때 public 생략한 경우
  - 다른 패키지에서는 사용 불가



접근 제한자

❖ 클래스의 접근 제한

- public
  - 다른 개발자가 사용할 수 있도록 라이브러리 클래스로 만들 때 유용



접근 제한자

❖ 클래스의 접근 제한

【 A.java 】 클래스의 접근 제한

```
01  package sec13.exam01_class_access.package1;
02
03  class A {} ←····· default 접근 제한
```

【 B.java 】 클래스의 접근 제한

```
01  package sec13.exam01_class_access.package1;
02
03  public class B {
04    A a;  //(o) ←····· A 클래스 접근 가능(필드로 선언할 수 있음)
05  }
```

접근 제한자

❖ 클래스의 접근 제한

【 C.java 】 클래스의 접근 제한

```
01  package sec13.exam01_class_access.package2;  <---- 패키지가 다름
02
03  import sec13.exam01_class_access.package1.*;
04
05  public class C {
06    A a;  //(x)  <---- A 클래스 접근 불가(컴파일 에러)
07    B b;  //(o)
08  }
```

접근 제한자

❖ 생성자 접근 제한

- 생성자가 가지는 접근 제한에 따라 호출 여부 결정

```
public class ClassName {
  //public 접근 제한
   public ClassName(…) { … }

   //protected 접근 제한
   protected ClassName(…) { … }

   //default 접근 제한
   ClassName(…) { … }

   //private 접근 제한
   private ClassName(…) { … }
}
```

접근 제한자

❖ 생성자 접근 제한

접근 제한자	생성자	설명
public	클래스(…)	public 접근 제한은 모든 패키지에서 아무런 제한 없이 생성자를 호출할 수 있도록 한다. 생성자가 public 접근 제한을 가진다면 클래스도 public 접근 제한을 가지는 것이 정상적이다. 클래스가 default 접근 제한을 가진다면 클래스 사용이 같은 패키지로 한정되므로, 비록 생성자가 public 접근 제한을 가지더라도 같은 패키지에서만 생성자를 호출할 수 있다.
protected	클래스(…)	protected 접근 제한은 default 접근 제한과 마찬가지로 같은 패키지에 속하는 클래스에서 생성자를 호출할 수 있도록 한다. 차이점은 다른 패키지에 속한 클래스가 해당 클래스의 자식(child) 클래스라면 생성자를 호출할 수 있다.
default	클래스(…)	생성자를 선언할 때 public 또는 private를 생략했다면 생성자는 default 접근 제한을 가진다. default 접근 제한은 같은 패키지에서는 아무런 제한 없이 생성자를 호출할 수 있으나, 다른 패키지에서는 생성자를 호출할 수 없도록 한다.
private	클래스(…)	private 접근 제한은 동일 패키지이건 다른 패키지이건 상관없이 생성자를 호출하지 못하도록 제한한다. 따라서 클래스 외부에서 new 연산자로 객체를 만들수 없다. 오로지 클래스 내부에서만 생성자를 호출할 수 있고, 객체를 만들 수 있다.

접근 제한자

❖ 생성자의 접근 제한: A.java

```
package sec13.exam02_constructor_access.package1;

public class A {
    // 필드
    A a1 = new A(true);
    A a2 = new A(1);
    A a3 = new A("문자열");

    // 생성자
    public A(boolean b) {  // public 접근 제한
    }

    A(int b) {             // default 접근 제한
    }

    private A(String s) {  // private 접근 제한
    }
}
```

접근 제한자

❖ 생성자의 접근 제한: B.java

```
package sec13.exam02_constructor_access.package1;   // 패키지가 동일

public class B {
    //필드
    A a1 = new A(true);        // (O)
    A a2 = new A(1);           // (O)
    A a3 = new A("문자열");    // (X)    private 생성자 접근 불가(컴파일 에러)
}
```

접근 제한자

❖ 생성자의 접근 제한 : C.java

```
package sec13.exam02_constructor_access.package2;   // 패키지가 다름

import sec13.exam02_constructor_access.package1.A;

public class C {
    A a1 = new A(true);    // (O)
    A a2 = new A(1);       // (X)    default 생성자 접근 불가(컴파일 에러)
    A a3 = new A("문자열");// (X)    private 생성자 접근 불가(컴파일 에러)
}
```

접근 제한자

❖ 필드와 메소드의 접근 제한

- 클래스 내부, 패키지 내, 패키지 상호간에 사용할 지 고려해 선언

```
//필드 선언
[ public ¦ protected ¦ private ] [static] 타입 필드;

//메소드 선언
[ public ¦ protected ¦ private ] [static] 리턴 타입 메소드(…) { … }
```

접근 제한자

❖ 필드와 메소드의 접근 제한

- 클래스 내부, 패키지 내, 패키지 상호간에 사용할 지 고려해 선언

접근 제한자	생성자	설명
public	필드 메소드(…)	public 접근 제한은 모든 패키지에서 아무런 제한 없이 필드와 메소드를 사용할 수 있도록 해준다. 필드와 메소드가 public 접근 제한을 가질 경우 클래스도 public 접근 제한을 가져야 한다. 클래스가 default 접근 제한을 가지게 되면 같은 패키지 안에서만 클래스가 사용되기 때문이다.
protected	필드 메소드(…)	protected 접근 제한은 default 접근 제한과 마찬가지로 같은 패키지에 속하는 클래스에서 필드와 메소드를 사용할 수 있도록 한다. 차이점은 다른 패키지에 속한 클래스가 해당 클래스의 자식 클래스라면 필드와 메소드를 사용할 수 있다.
default	필드 메소드(…)	필드와 메소드를 선언할 때 public 또는 private를 생략했다면 default 접근 제한을 가진다. default 접근 제한은 같은 패키지에서는 아무런 제한 없이 필드와 메소드를 사용할 수 있으나, 다른 패키지에서는 필드와 메소드를 사용할 수 없도록 한다.
private	필드 메소드(…)	private 접근 제한은 동일 패키지이건 다른 패키지이건 상관없이 필드와 메소드를 사용하지 못하도록 제한한다. 오로지 클래스 내부에서만 사용할 수 있다.

접근 제한자

❖ 필드와 메소드의 접근 제한 : A.java

```
package sec13.exam03_field_method_access.package1;

public class A {
    public int field1;     // public 접근 제한
    int field2;            // default 접근 제한
    private int field3;    // private 접근 제한

    public A() {        // 메서드 내부일 경우 접근 제한자의 영향을 받지 않음
        field1 = 1;
        field2 = 1;
        field3 = 1;

        method1();
        method2();
        method3();
    }

    public void method1() {   }  // public 접근 제한
    void method2() {    }        // default 접근 제한
    private void method3() {   } // private 접근 제한
}
```

접근 제한자

❖ 필드와 메소드의 접근 제한 : B.java

```
package sec13.exam03_field_method_access.package1;  // 패키지가 동일

public class B {
    public B() {
        A a = new A();
        a.field1 = 1;    // (O)
        a.field2 = 1;    // (O)
        a.field3 = 1;    // (X)    private 필드 접근 불가(컴파일 에러)

        a.method1();     // (O)
        a.method2();     // (O)
        a.method3();     // (X)    private 필드 접근 불가(컴파일 에러)
    }
}
```

접근 제한자

❖ 필드와 메서드의 접근 제한: C.java

```
package sec13.exam03_field_method_access.package2;  // 패키지가 다름

import sec13.exam03_field_method_access.package1.A;

public class C {
    public C() {
        A a = new A();
        a.field1 = 1;    // (O)
        a.field2 = 1;    // (X)    protected 필드 접근 불가(컴파일 에러)
        a.field3 = 1;    // (X)    private 필드 접근 불가(컴파일 에러)

        a.method1();     // (O)
        a.method2();     // (X)    protected 필드 접근 불가(컴파일 에러)
        a.method3();     // (X)    private 필드 접근 불가(컴파일 에러)

    }
}
```

Getter와 Setter

❖ Getter와 Setter 메서드 사용: Car.java

```
public class Car {

    private int speed;
    private boolean stop;

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        if (speed < 0) {
            this.speed = 0;
            return;
        } else {
            this.speed = speed;
        }
    }

    public boolean isStop() {
        return stop;
    }
```

❖ Getter와 Setter 메서드 사용: Car.java

```
    public void setStop(boolean stop) {
        this.stop = stop;
        this.speed = 0;
    }
}
```

Getter와 Setter

❖ Getter와 Setter 메서드 사용: CarExample.java

```
public class CarExample {
    public static void main(String[] args) {
        Car myCar = new Car();

        // 잘못된 속도 변경
        myCar.setSpeed(-50);

        System.out.println("현재 속도: " + myCar.getSpeed());

        // 올바른 속도 변경
        myCar.setSpeed(60);

        // 멈춤
        if (!myCar.isStop()) {
            myCar.setStop(true);
        }

        System.out.println("현재 속도: " + myCar.getSpeed());
    }
}
```