어서와 Java는 처음이지!

제5장 클래스와 객체

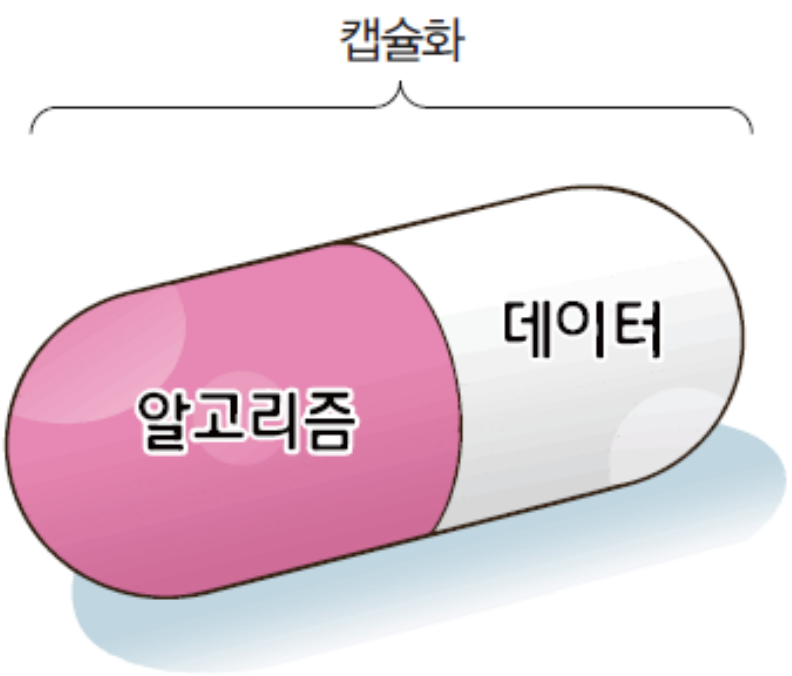
- 객체 지향 특징
- 캡슐화
- 정보 은닉
- 접근 제어
  - 접근자
  - 설정자

객체 지향의 3대 특징

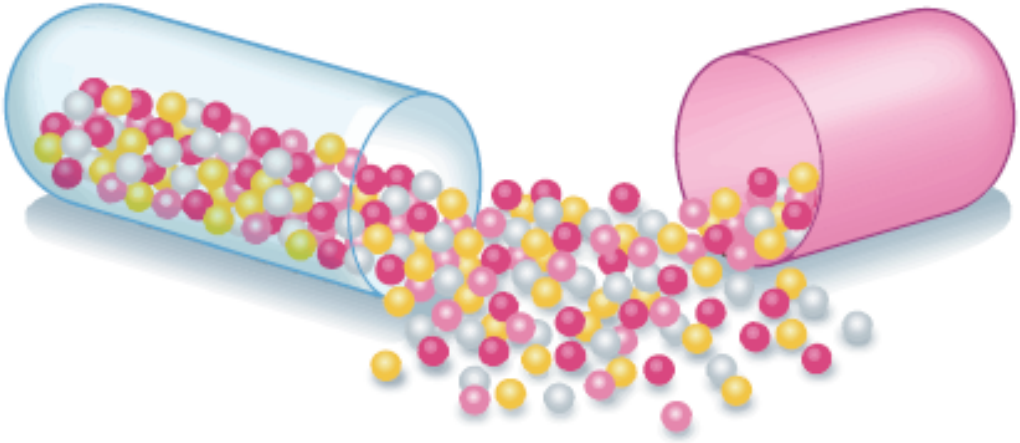
- 캡슐화
- 다형성
- 상속

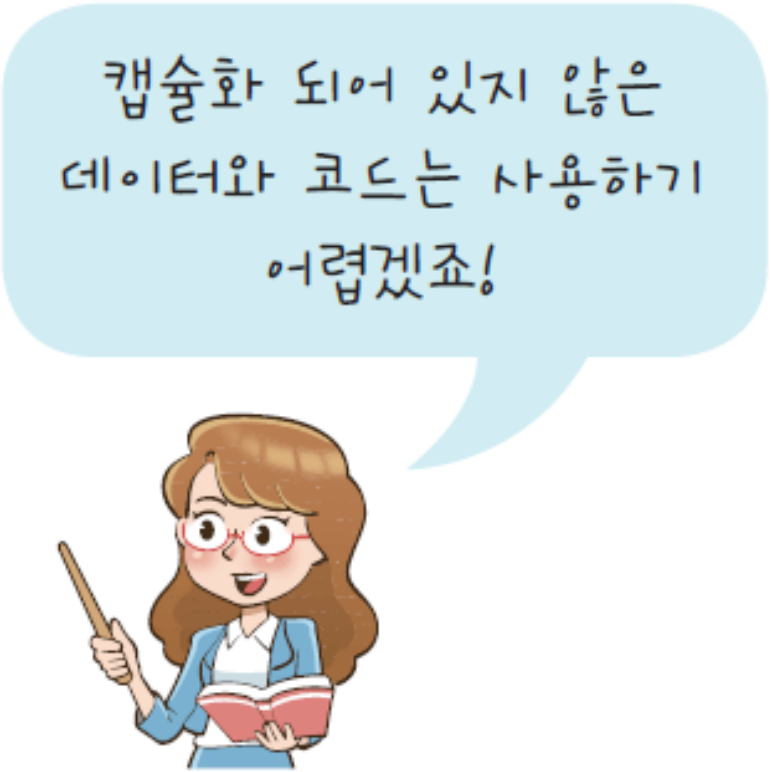
캡슐화

- **캡슐화(encapsulation)**: 관련된 데이터와 알고리즘(코드)이 하나의 묶음으로 정리되어 있는 것



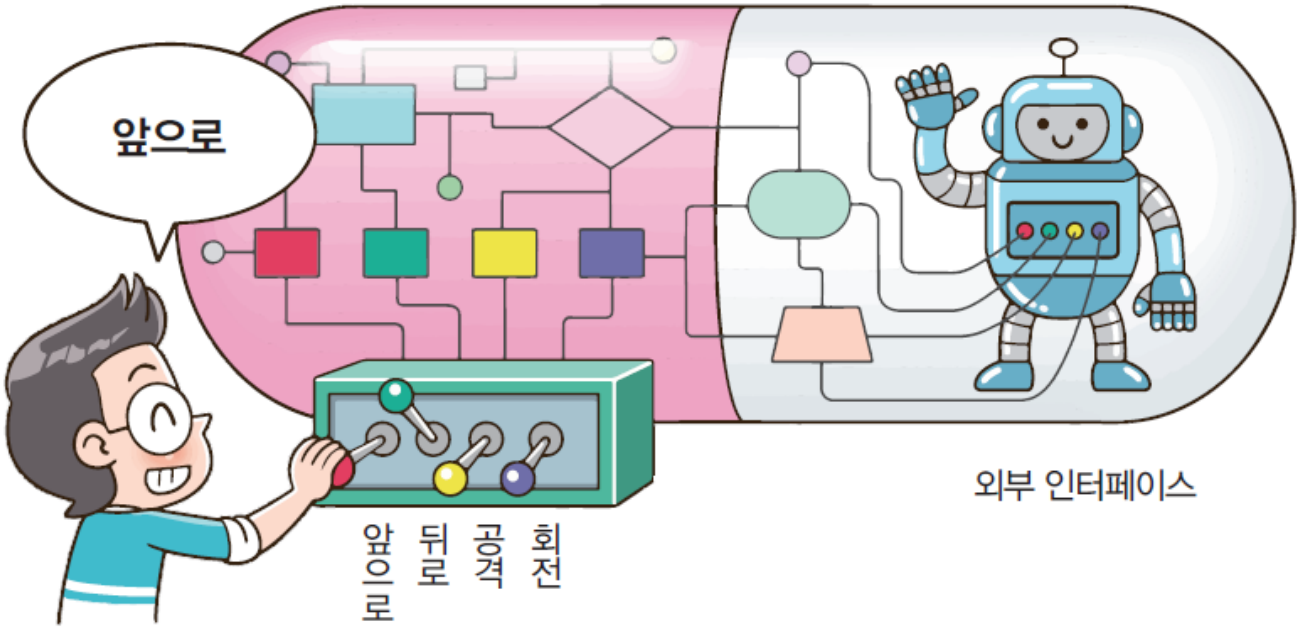
캡슐화

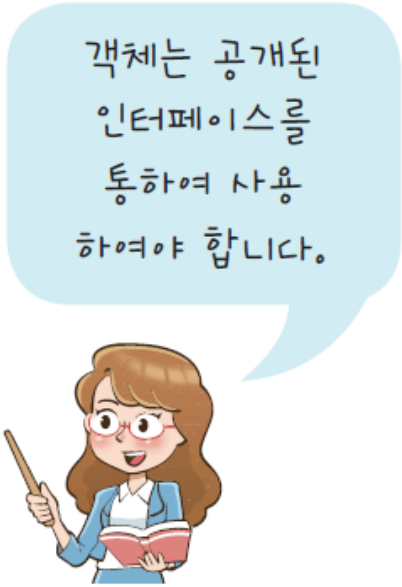




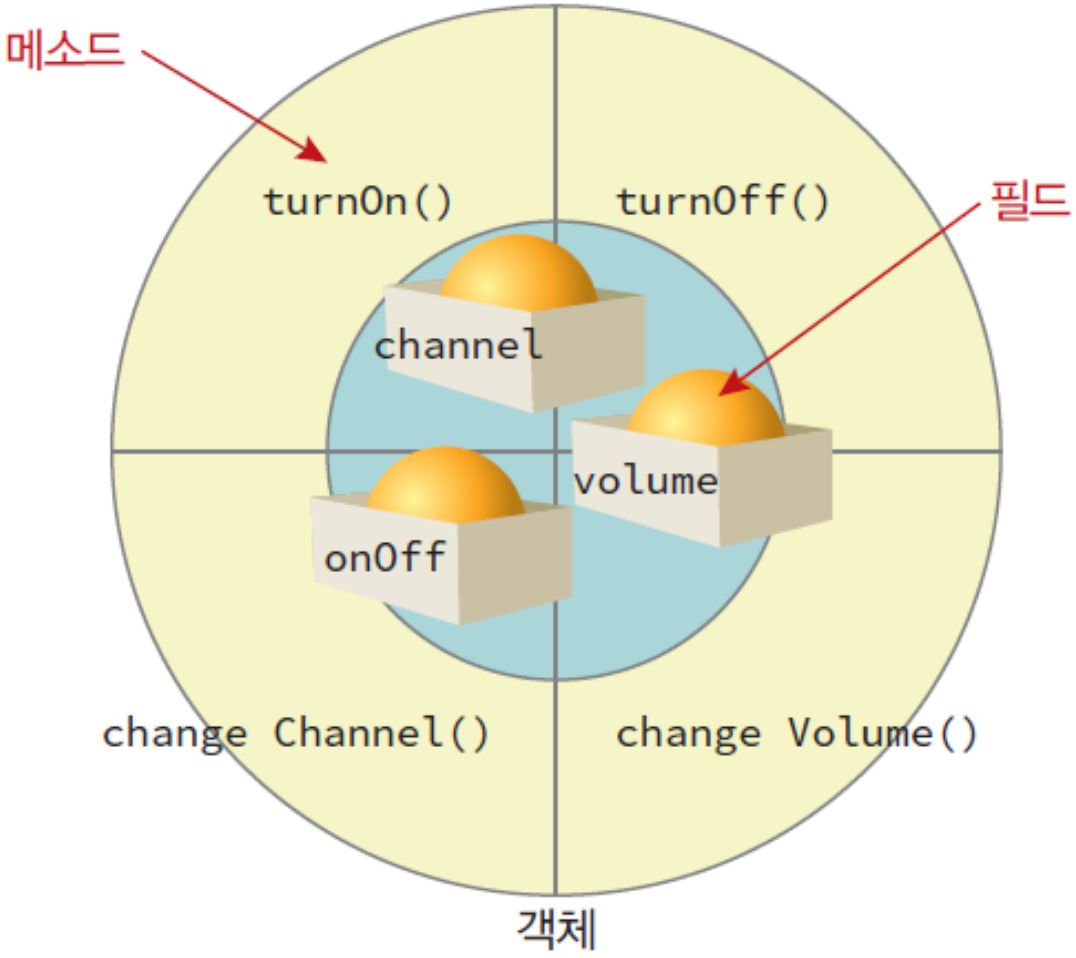
캡슐화와 정보 은닉

- 정보 은닉(information hiding)은 객체를 캡슐로 싸서 객체의 내부를 보호하는 하는 것
- 객체의 실제 구현 내용을 외부에 감추는 것





캡슐화와 정보 은닉

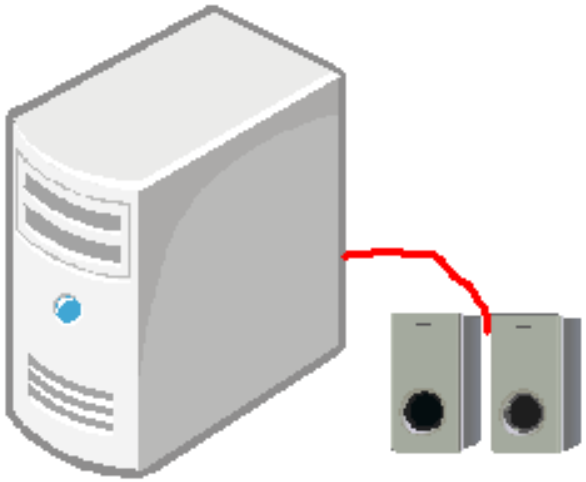


보통은 데이터들은 공개되지 않고 몇 개의 메소드만이 외부로 공개됩니다.

업그레이드가 쉽다.

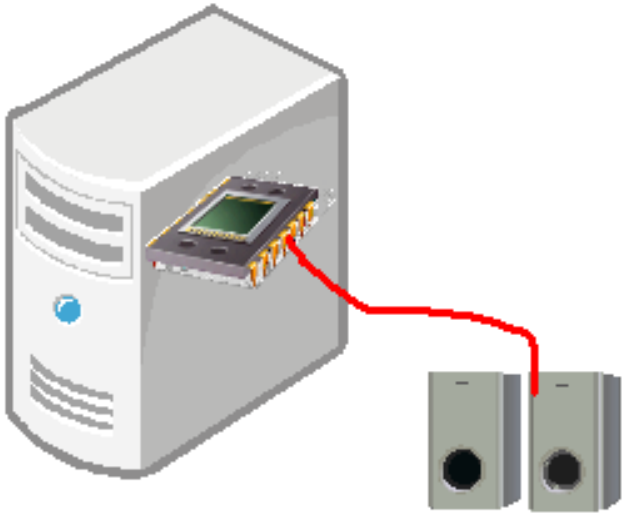
- 라이브러리가 업그레이드되면 쉽게 바꿀 수 있음
- 정보 은닉이 가능하기 때문에 업그레이드 가능

O



만약 외부의 표준 오디오 단자를 이용하였으면 내부의 사운드 카드를 변경할 수 있다.

X



만약 내부의 오디오 제어칩의 단자에 연결하였으면 내부의 사운드 카드를 변경할 수 없다.

접근 제어

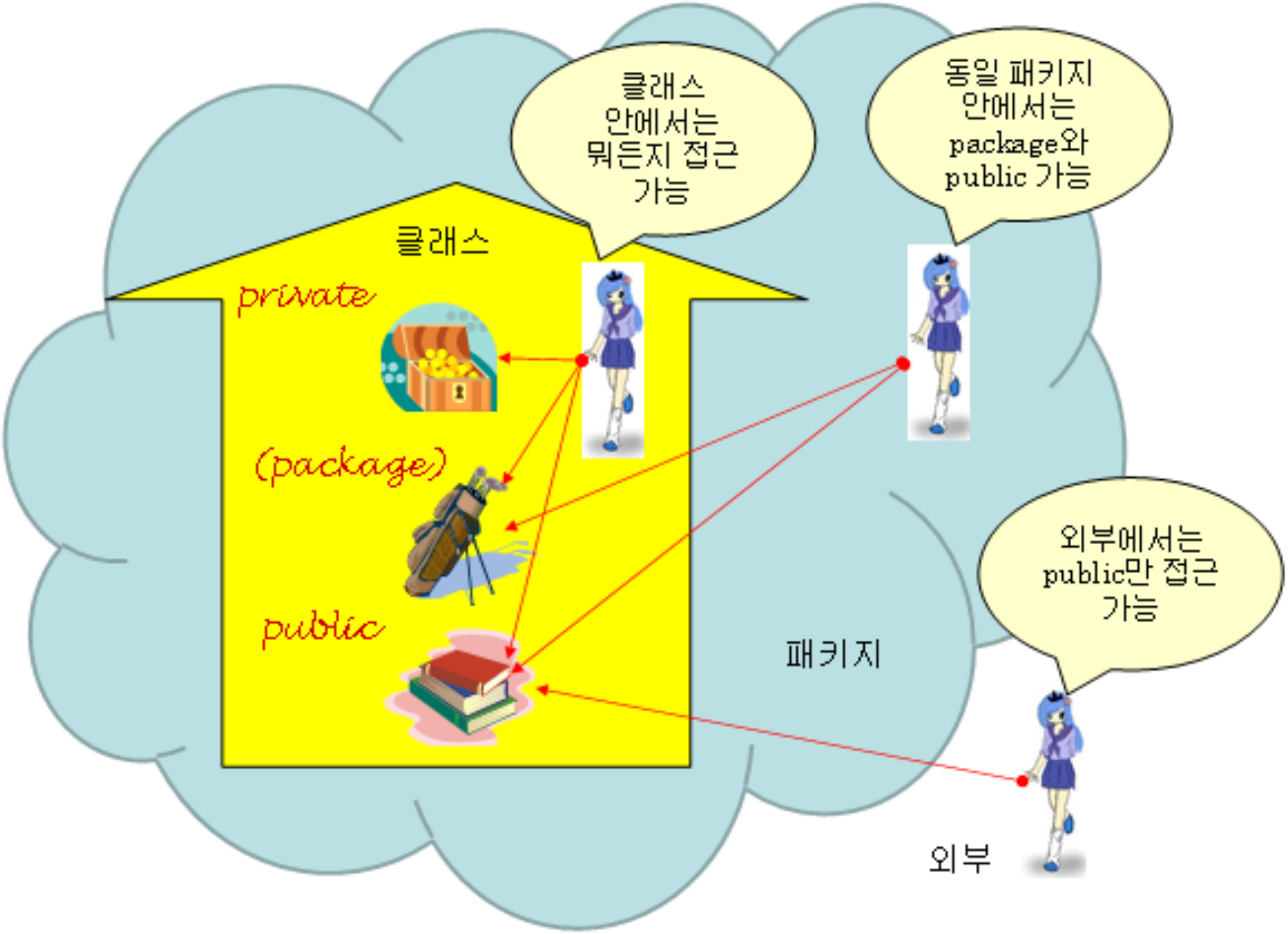
- 클래스 안에 변수나 메소드들을 누구나 사용할 수 있게 하면 어떻게 될까?

-> 많은 문제가 발생할 것이다.

- (예) 국가 기밀 서류를 누구나 보도록 방치하면 어떻게 될까?

접근 제어

- 접근 제어(access control):
  - 다른 클래스가 특정한 필드나 메소드에 접근하는 것을 제어하는 것


클래스
안에서는
뭐든지 접근
가능
동일 패키지
안에서는
package와
public 가능
클래스
private
(package)
public
외부에서는
public만 접근
가능
패키지
외부

멤버 수준에서의 접근 제어

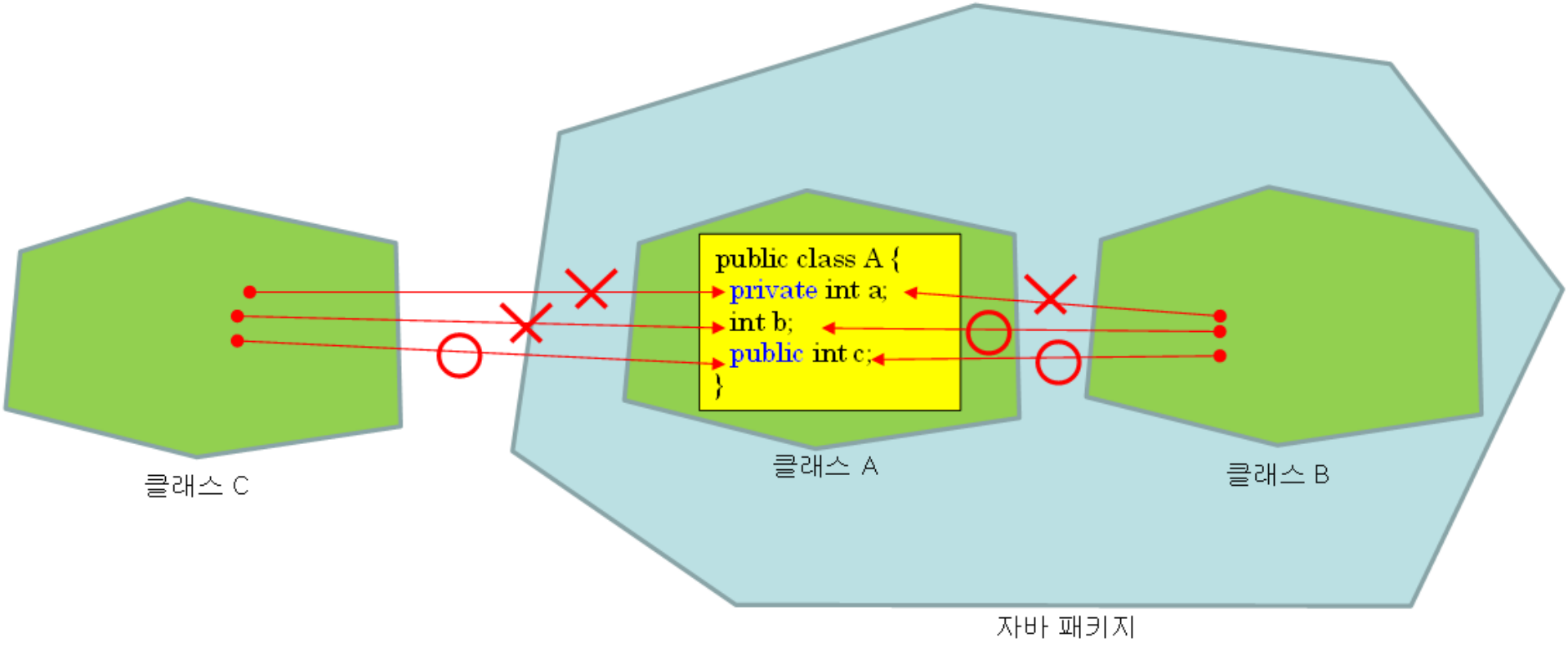
접근지정자	클래스	패키지	자식클래스	전체세계
public	O	O	O	O
protected	O	O	O	X
없음	O	O	X	X
private	O	X	X	X

예제

```
class A {
	private int a;		// 전용
	int b;			// 디폴트
	public int c;		// 공용
}

public class Test {
	public static void main(String args[]) {
		A obj = new A();	// 객체 생성
		obj.a = 10;	// 전용 멤버는 다른 클래스에서는 접근 안 됨
		obj.b = 20;	// 디폴트 멤버는 접근할 수 있음
		obj.c = 30;	// 공용 멤버는 접근할 수 있음
	}
}
```

예제


public class A {
private int a;
int b;
public int c;
}
클래스 C
클래스 A
클래스 B
자바 패키지

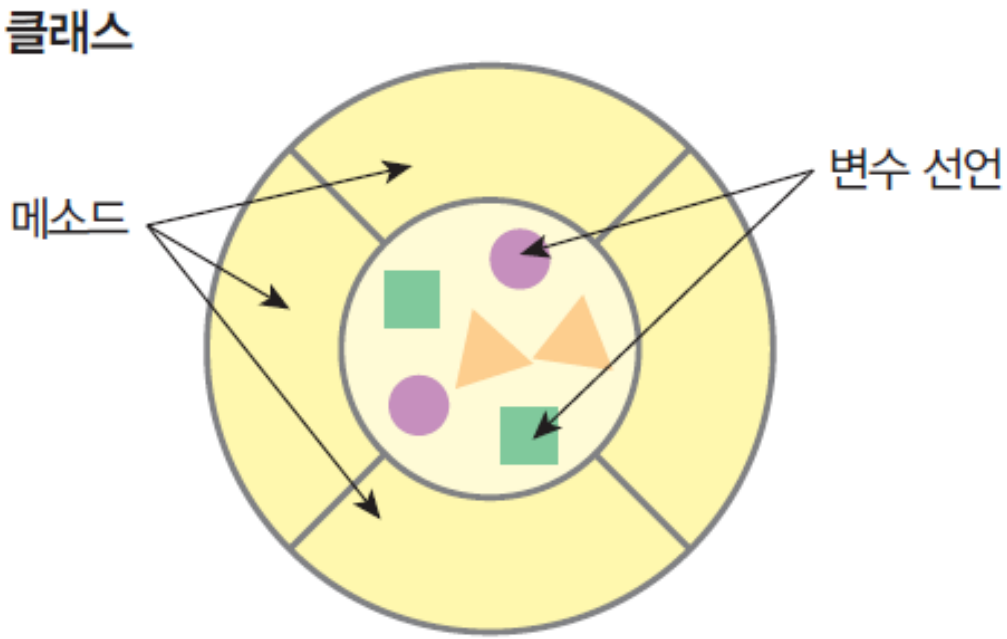
중간 점검 문제

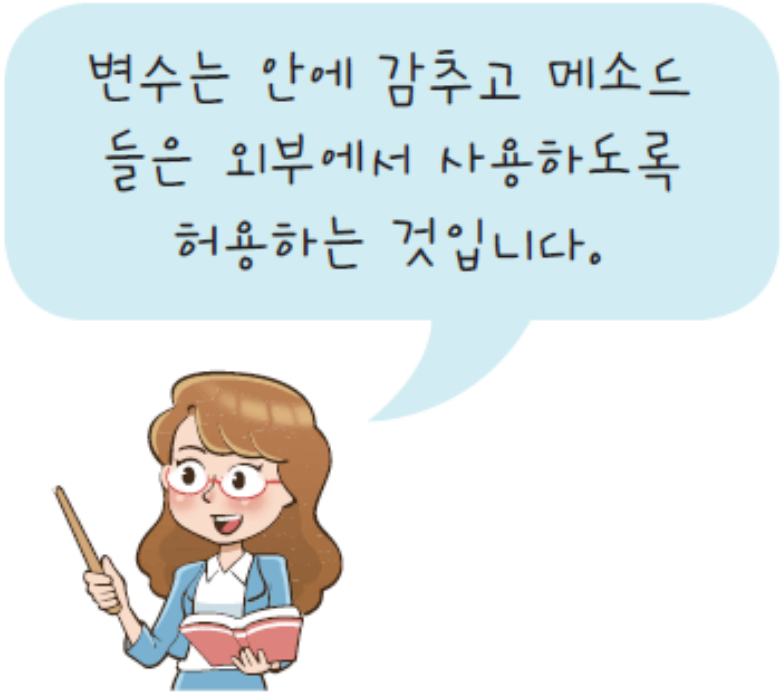
1. 필드의 경우, private로 만드는 것이 바람직한 이유는 무엇인가?

2. 필드를 정의할 때 아무런 접근 제어 수식자를 붙이지 않으면 어떻게 되는가?

정보은닉

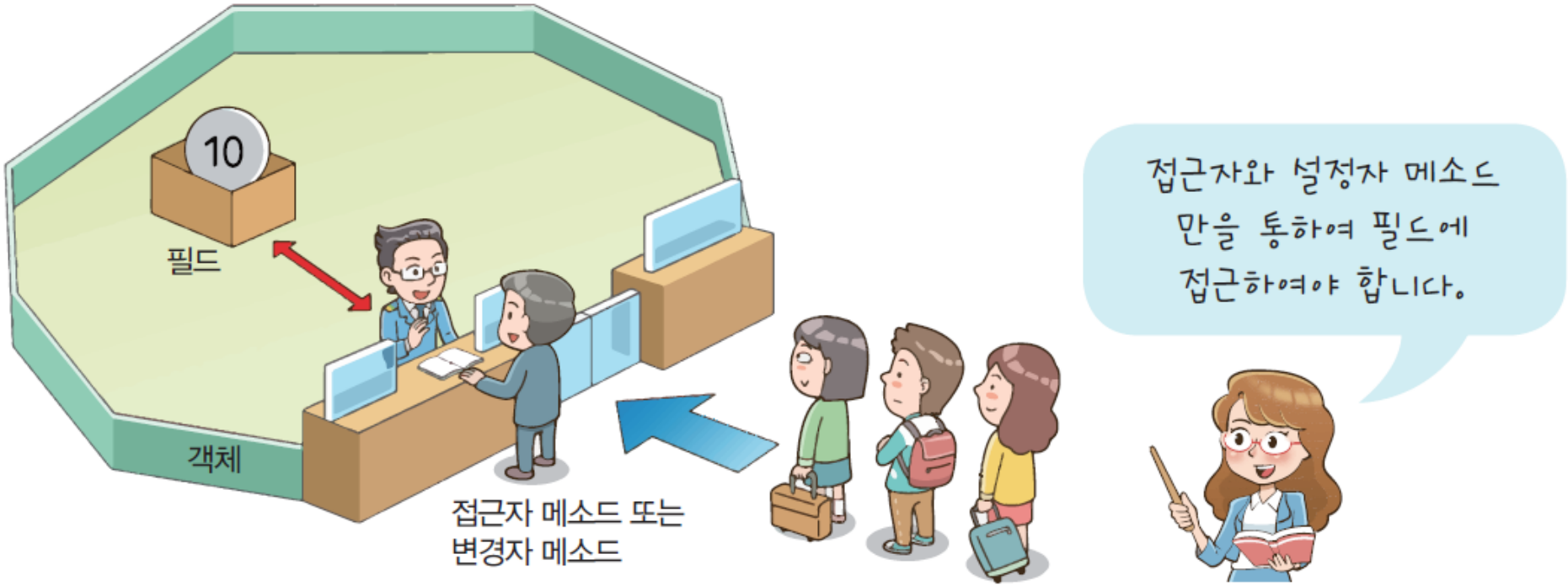
- 정보 은닉이란 구현의 세부 사항을 클래스 안에 감추는 것이다





설정자와 접근자

- 설정자(mutator)
- 접근자(accessor)



설정자와 접근자

- 설정자(mutator)
  - 필드의 값을 설정하는 메소드
  - setXXX() 형식
- 접근자(accessor)
  - 필드의 값을 반환하는 메소드
  - getXXX() 형식

설정자와 접근자

```
public class Account {
	private int regNumber;
	private String name;
	private int balance;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getBalance() {
		return balance;
	}
	public void setBalance(int balance) {
		this.balance = balance;
	}
}
```

예제

```
public class AccountTest {
	public static void main(String[] args) {
		Account obj = new Account();
		obj.setName("Tom");
		obj.setBalance(100000);
		System.out.println("이름은 " + obj.getName()
				+ " 통장 잔고는 "
				+ obj.getBalance() + "입니다.");
	}
}
```

이름은 Tom 통장 잔고는 100000입니다.

설정자와 접근자는 왜 사용하는가?

- 접근자와 설정자를 사용해야만 나중에 클래스를 업그레이드할 때 편하다.
- 접근자에서 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있다.
- 필요할 때마다 필드값을 계산하여 반환할 수 있다.
- 접근자만을 제공하면 자동적으로 읽기만 가능한 필드를 만들 수 있다.

접근자와 설정자의 장점 예제

- 설정자는 변수의 값을 변경하려는 외부의 시도를 주의 깊게 검사할 수 있다.

```
public void setAge(int age)
{
      if( age < 0 )
            this.age = 0;
      else
            this.age = age;
}
```

LAB 4-1: 안전한 배열 만들기

- 만약 인덱스가 배열의 크기를 벗어나게 되면 실행 오류가 발생한다.
- 따라서 실행 오류를 발생하지 않는 안전한 배열을 작성하여 보자.

```
public class SafeArray {
	private int a[];
	public int length;
	public SafeArray(int size) {
		a = new int[size];
		length = size;
	}
	public int get(int index) {
		??
	}
	public void put(int index, int value) {
		??
	}
}
```

Test Code

```
public class SafeArrayTest {
	public static void main(String args[]) {
		SafeArray array = new SafeArray(3);

		for (int i = 0; i < (array.length + 1); i++) {
			array.put(i, i * 10);
		}
	}
}
```

잘못된 인덱스 3

Q & A