탈잉 개발자와 프로그래밍 무작정 배워보기

4. 접근성과 상속

강대규

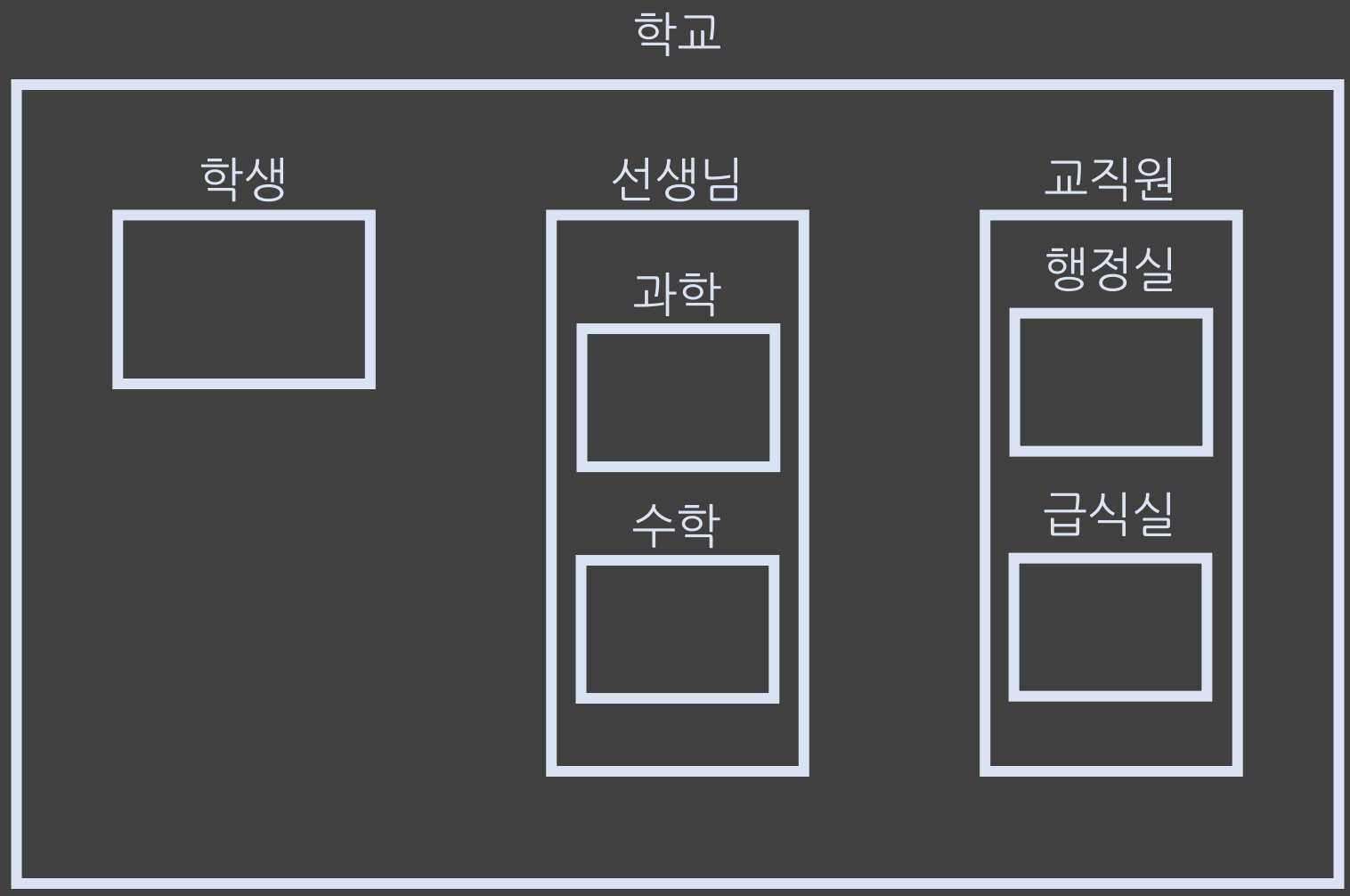
1. 패키지와 접근제어자

목차

패키지

비슷한 클래스, 인터페이스를 묶어놓은 그룹

패키지



패키지

여러명이서 기능을 분담하여 작업할 경우

클래스 이름이 같다면 어떻게 할 것인가?

패키지

여러명이서 기능을 분담하여 작업할 경우

클래스 이름이 같다면 어떻게 할 것인가?

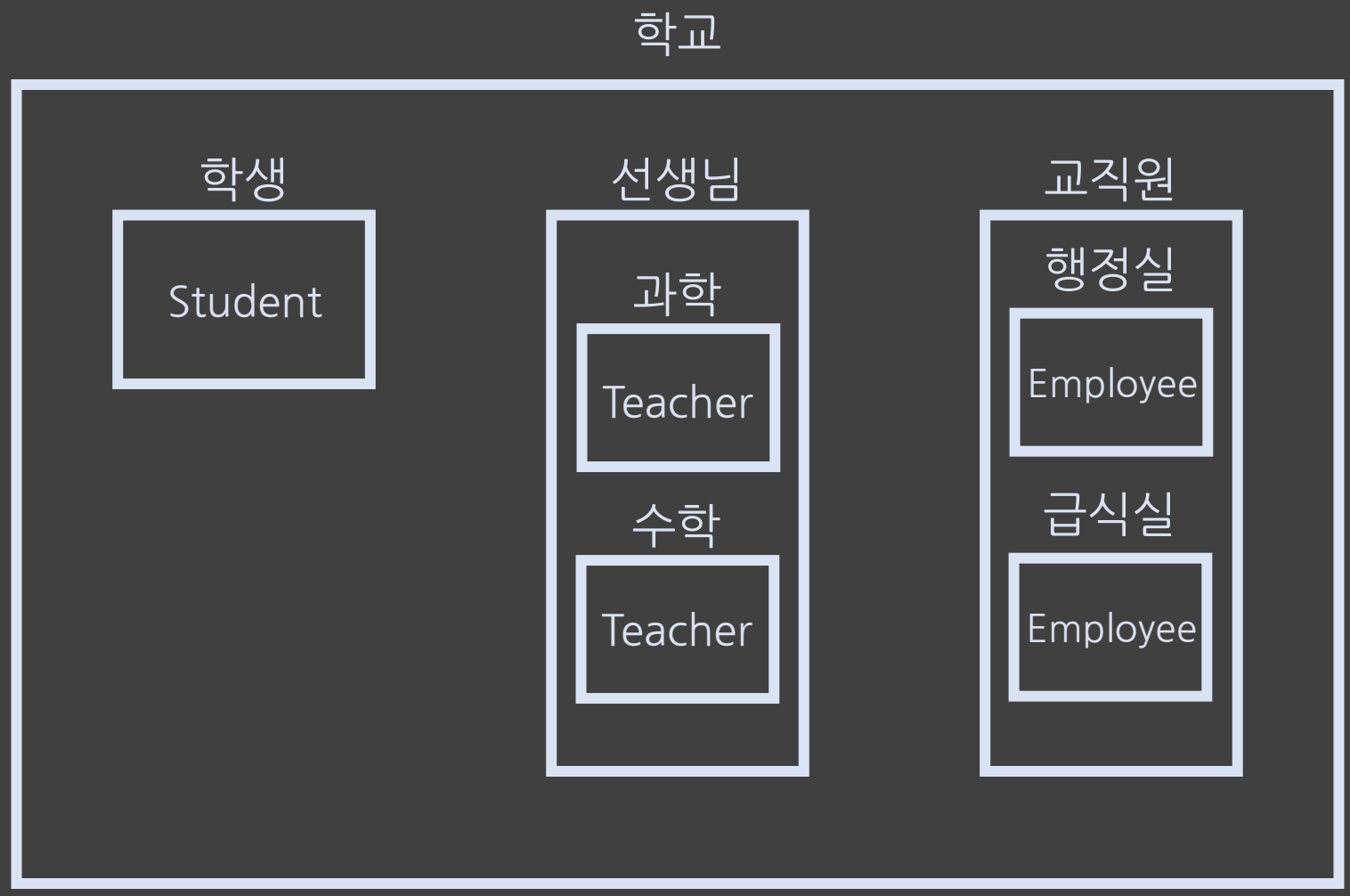
→ 각 기능을 패키지로 나누어서 진행해보자

패키지

비슷한 클래스, 인터페이스를 묶어놓은 그룹

→ 같은 클래스 이름이더라도 패키지가 다르면 다른 파일로 처리

패키지


학교
학생
Student
선생님
과학
Teacher
수학
Teacher
교직원
행정실
Employee
급식실
Employee

패키지

학교
- 학생
  - 학생
- 선생님
  - 과학
    - 선생님
  - 수학
    - 선생님
- 교직원
  - 행정실
    - 직원
  - 급식실
    - 직원

패키지

school
- student
  - Student
- teacher
  - science
    - Teacher
  - math
    - Teacher
- employee
  - admin
    - Employee
  - feeding
    - Employee

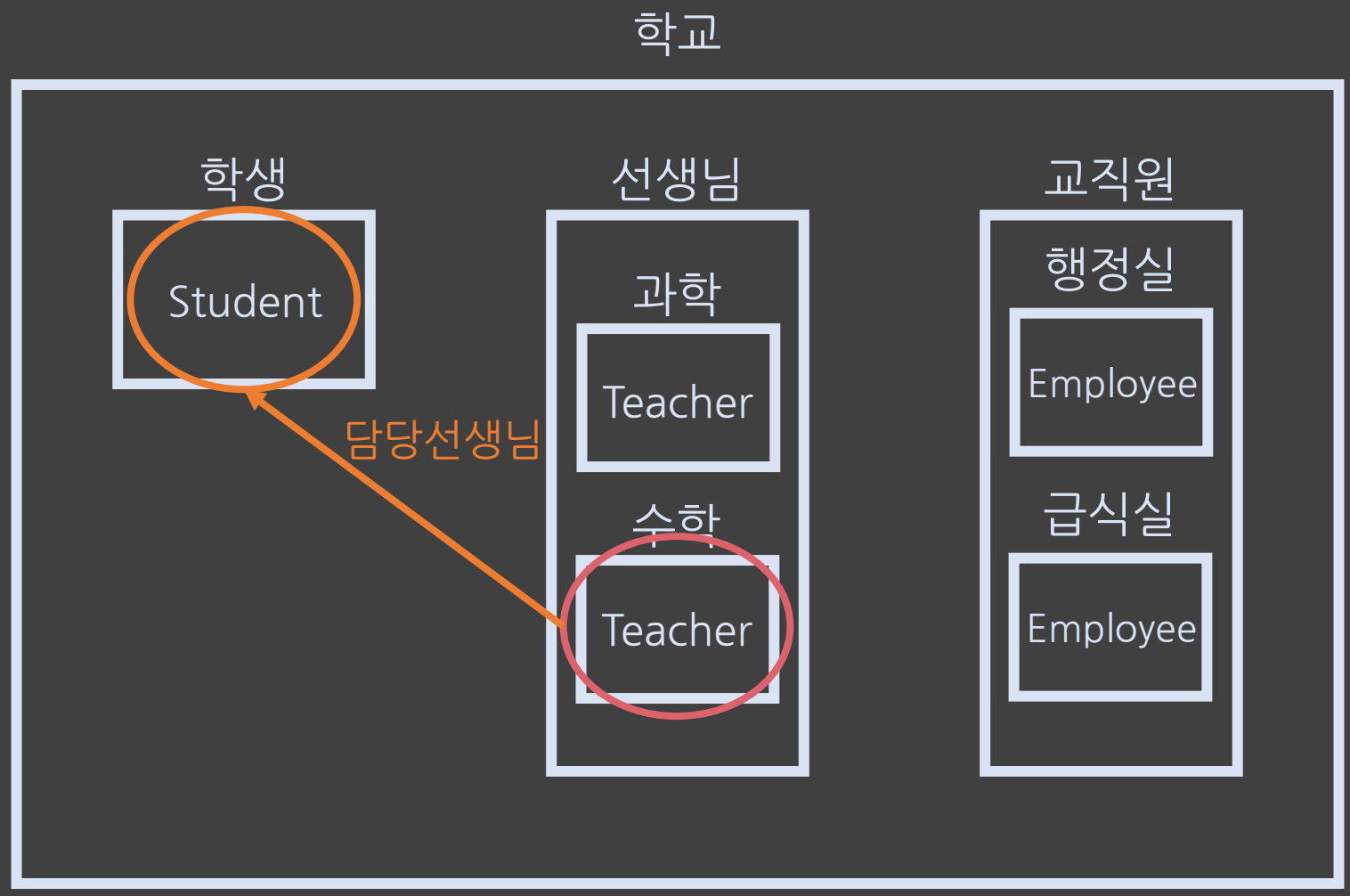
패키지

school
- student
  - Student → school.student.Student
- teacher
  - science
    - Teacher → school.teacher.science.Teacher
  - math
    - Teacher → school.teacher.math.Teacher
- employee
  - admin
    - Employee → school.employee.admin.Employee
  - feeding
    - Employee → school.employee.feeding.Employee

패키지 - import

다른 패키지의 클래스를 불러오고자 할 때 쓰임

패키지 - import


학교
학생
Student
선생님
과학
Teacher
수학
Teacher
교직원
행정실
Employee
급식실
Employee
담당선생님

패키지 - import

```
import school.teacher.math.Teacher

class Student {

    Teacher myTeacher;

}
```

접근제어자

클래스, 전역변수, 메소드의 접근 권한을 제한하는 것

접근제어자

클래스, 전역변수, 메소드의 접근 권한을 제한하는 것

→ public, protected, default, private 이 존재

접근제어자

클래스, 전역변수, 메소드의 접근 권한을 제한하는 것

→ public, default 이 존재

접근제어자

```
private int c = 10;

...

public void sum(int a, int b) {
    return a + b;
}
```

접근제어자

1. public : 어디서든 접근 가능

접근제어자

1. public : 어디서든 접근 가능
2. protected : 같은 패키지 내에서만 접근 가능, 상속받은 클래스도 접근 가능

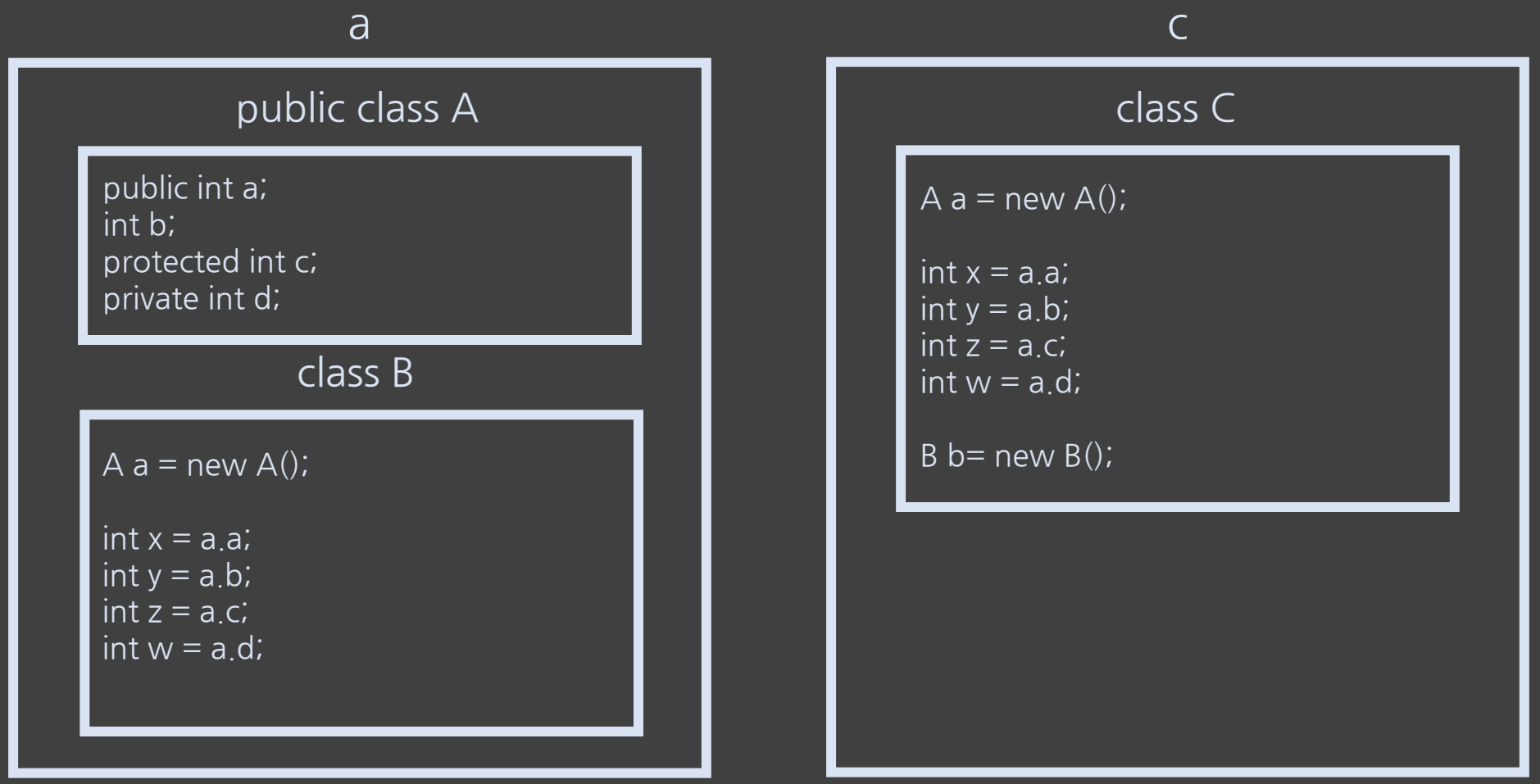
접근제어자

1. public : 어디서든 접근 가능
2. protected : 같은 패키지 내에서만 접근 가능, 상속받은 클래스도 접근 가능
3. default : 같은 패키지 내에서만 접근 가능

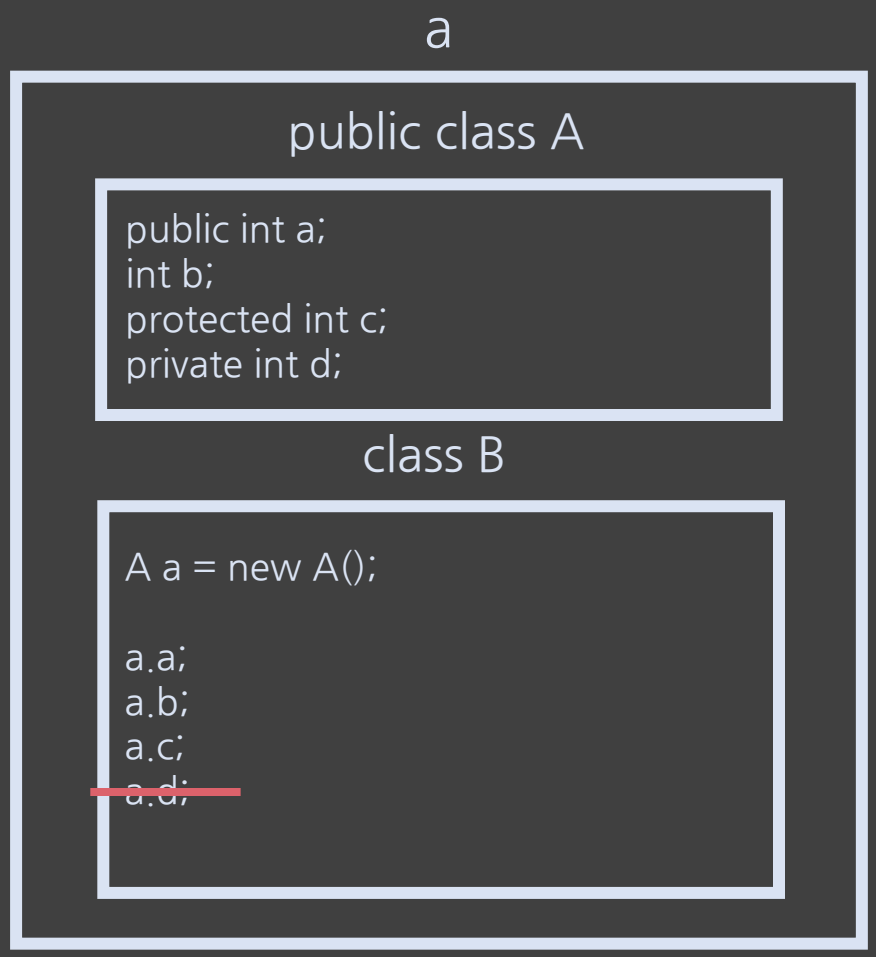
접근제어자

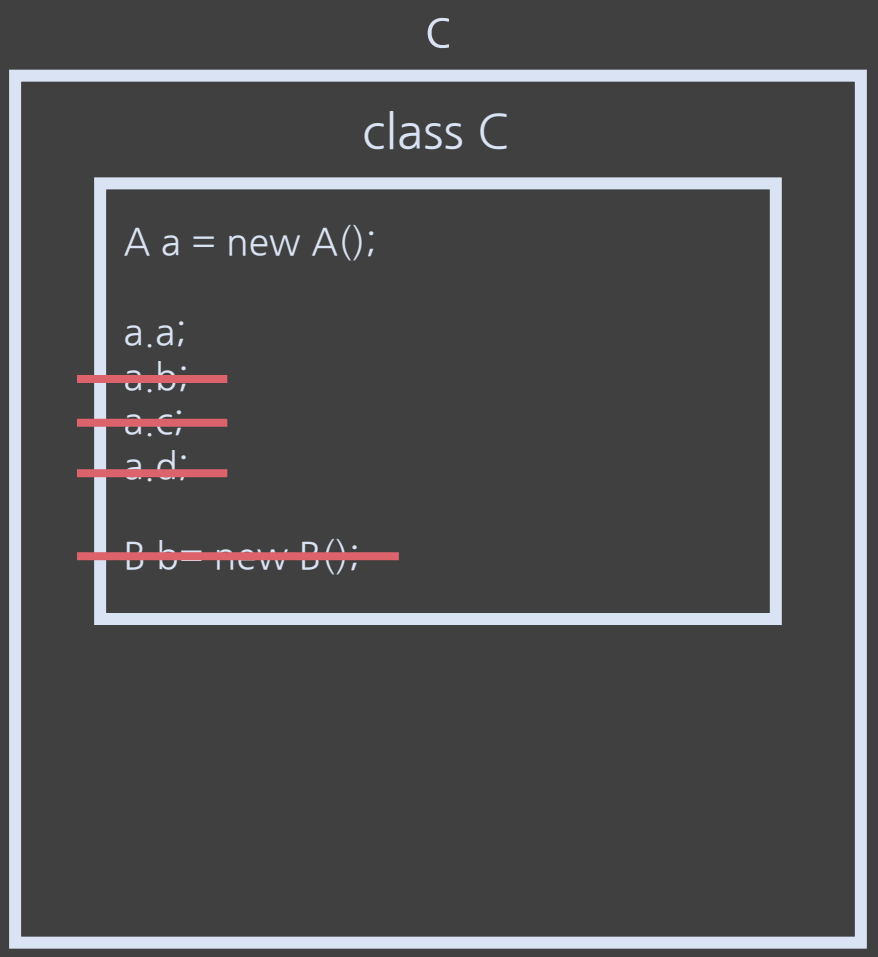
1. public : 어디서든 접근 가능
2. protected : 같은 패키지 내에서만 접근 가능, 상속받은 클래스도 접근 가능
3. default : 같은 패키지 내에서만 접근 가능
4. private : 같은 클래스 내에서만 접근 가능

접근제어자


a
public class A
public int a;
int b;
protected int c;
private int d;
class B
A a = new A();
int x = a.a;
int y = a.b;
int z = a.c;
int w = a.d;
c
class C
A a = new A();
int x = a.a;
int y = a.b;
int z = a.c;
int w = a.d;
B b= new B();

접근제어자


a
public class A
public int a;
int b;
protected int c;
private int d;
class B
A a = new A();
a.a;
a.b;
a.c;
a.d;


c
class C
A a = new A();
a.a;
a.b;
a.c;
a.d;
B b= new B();

2. static 과 final

static

전역 변수나 메소드 앞에 붙어서 쓰이며

응용 프로그램이 종료되지 않는 한 메모리 할당을 한번만 함

static 변수

어떤한 인스턴스에서도

공통된 변수를 사용하고 싶다면

static 변수

```
class A {
    public static int count = 0;

    public void increaseCount() {
        count++;
    }
}
```

```
class B {

    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new A();

        a1.increaseCount();
        System.out.println(a1.count);

        a2.increaseCount();
        System.out.println(a2.count);

        System.out.println(A.count);
    }
}
```

static 메소드

인스턴스 생성 없이도 메소드에 접근할 수 있도록 할 때

static 메소드

```
class A {
    private static int count = 0;

    public void increaseCount() {
        count++;
    }

    public static int getCount() {
        return count;
    }
}
```

```
class B {

    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new A();

        a1.increaseCount();
        System.out.println(A.getCount());

        a2.increaseCount();
        System.out.println(A.getCount());

        System.out.println(A.getCount());
    }
}
```

static 변수와 메소드의 예

System.out.println(“Hello World”);

long time = System.currentTimeMillis();

static 변수와 메소드의 예

```
System.out.println(“Hello World”);

long time = System.currentTimeMillis();
```

static 변수와 메소드의 예

System.out.println(“Hello World”);

long time = System.currentTimeMillis();

→ 또한 어디서든 우리는 접근할 수 있기 때문에 public 이다!

final

변수를 선언할 때 쓰이며, 값을 할당한 이후에는 값을 바꿀 수 없다.

final

```
final int a = 10;

a = 20;
```

final

```
final int a = 10;

a = 20;
```

final

변수를 선언할 때 쓰이며, 값을 할당한 이후에는 값을 바꿀 수 없다.

또한 변수를 선언할 때 동시에 값을 할당해야만 한다.

final

```
final int a;

a = 20;
```

final

```
final int a;

a = 20;
```

static final 변수

어떠한 클래스 내에서 바뀔 수 없는 공통의 상수

static final 변수

```
class A {
    public static final int COUNT = 0;
}
```

```
class B {

    public static void main(String[] args) {
        System.out.println(A.COUNT);
    }
}
```

3. 상속과 Override

상속

B 라는 클래스가 A 클래스를 상속받으면

B 클래스는 A 의 기능을 쓸 수 있다.

상속

```
class Pet {
    public void eatFeed() {
        …
    }
}
```

```
class Dog extends Pet {
    public void bark() {
        …
    }
}
```

상속

```
public static void main(String[] args) {
    Pet pet = new Pet();
    pet.eatFeed();

    Dog dog = new Dog();
    dog.eatFeed();
    dog.bark();
}
```

상속

Is - a 라는 관계가 성립한다면 상속을 시키는 것이 좋다.

상속

Is - a 라는 관계가 성립한다면 상속을 시키는 것이 좋다.

→ Dog is a Pet

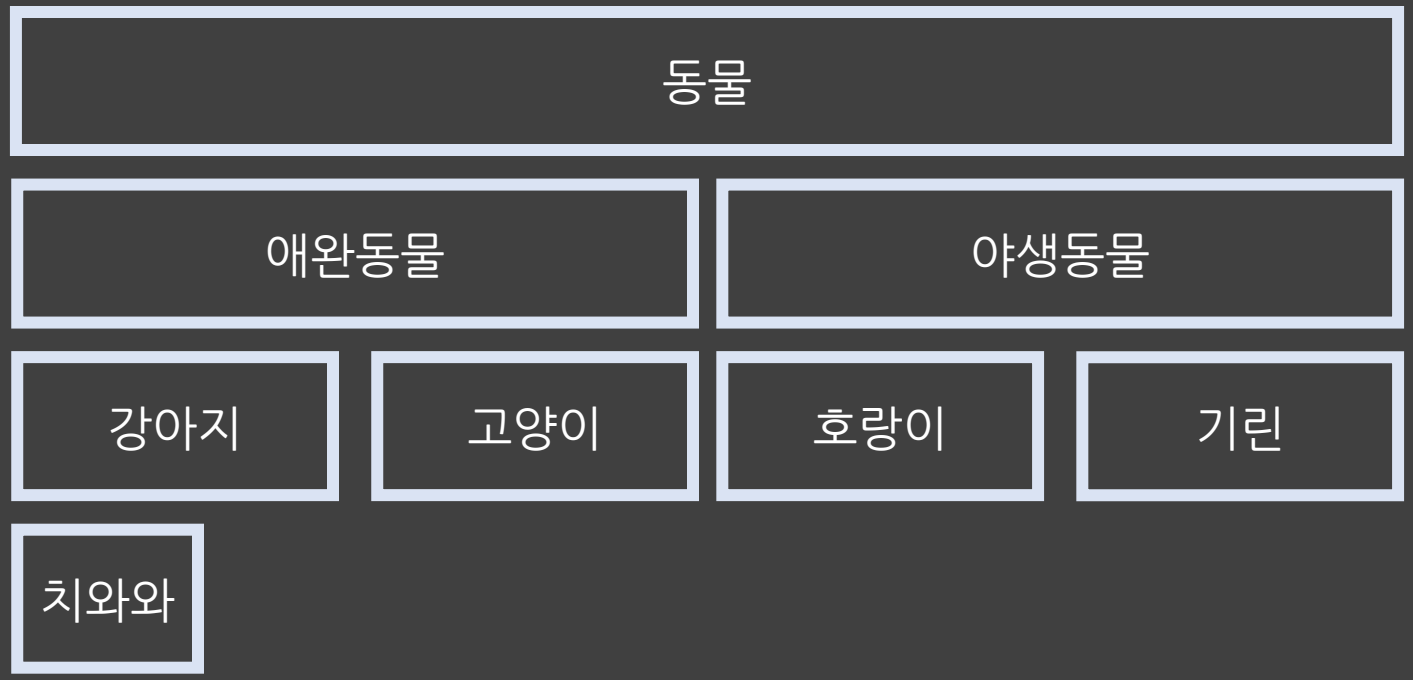
상속

Is - a 라는 관계가 성립한다면 상속을 시키는 것이 좋다.

→ Dog is a Pet

→ Dog extends Pet

상속



상속과 Override

부모 클래스가 가지고 있는 메소드를

자식 클래스가 재정의해서 사용

상속과 Override

강아지는 사료를 먹을 때 짖는다고 가정해보자

상속과 Override

```
public static void main(String[] args) {
    Pet pet = new Pet();
    pet.eatFeed();

    Dog dog = new Dog();
    dog.eatFeed();
    dog.bark();

    dog.eatFeed();
    dog.bark();
}
```

상속과 Override

```
public static void main(String[] args) {
    Pet pet = new Pet();
    pet.eatFeed();

    Dog dog = new Dog();
    dog.eatFeed();
    dog.bark();

    dog.eatFeed();
    dog.bark();
}
```

매번 이렇게 해줘야할까?
eatFeed 에 기능을 추가하면 되지 않을까..?!

상속과 Override

```
class Pet {
    public void eatFeed() {
        …
    }
}

class Dog extends Pet {
    public void bark() {
        …
    }

    @Override
    public void eatFeed() {
        super.eatFeed();
        bark();
    }
}
```

상속과 Override

```
class Pet {
    public void eatFeed() {
        …
    }
}

class Dog extends Pet {
    public void bark() {
        …
    }

    @Override
    public void eatFeed() {
        super.eatFeed();
        bark();
    }
}
```

Pet 의 eatFeed 를 재정의하여 사용

상속과 Override

```
class Pet {
    public void eatFeed() {
        …
    }
}

class Dog extends Pet {
    public void bark() {
        …
    }

    @Override
    public void eatFeed() {
        super.eatFeed();
        bark();
    }
}
```

Pet 의 eatFeed 기능을 수행

상속과 instanceof

인스턴스의 타입을 알아보기 위해 쓰는

비교연산자

상속과 instanceof

```
public class AnimalDoctor {

    public void cure(Pet pet) {

        if (pet instanceof Dog) {
            System.out.println("강아지가 왔군");
        } else if (pet instanceof Cat) {
            System.out.println("고양이가 왔군");
        } else {
            System.out.println("독특한 애완동물!");
        }
    }
}
```

상속과 instanceof

```
public class AnimalHospital {

    public static void main(String[] args) {
        Dog dog = new Dog();
        Cat cat = new Cat();

        AnimalDoctor doctor = new AnimalDoctor();

        doctor.cure(dog);
        doctor.cure(cat);
    }

}
```

상속과 형변환

자식 클래스는 부모 클래스로의 형변환이 가능하다.

상속과 형변환

자식 클래스는 부모 클래스로의 형변환이 가능하다.

반대로는 불가능하다.

상속과 형변환

```
public class AnimalHospital {

    public static void main(String[] args) {
        Pet pet1 = new Dog();
        Pet pet2 = new Cat();

        AnimalDoctor doctor = new AnimalDoctor();

        doctor.cure(pet1);
        doctor.cure(pet2);
    }

}
```

상속과 형변환

```
public class AnimalHospital {

    public static void main(String[] args) {
        Pet pet1 = new Dog();
        Pet pet2 = new Cat();

        AnimalDoctor doctor = new AnimalDoctor();

        doctor.cure(pet1);
        doctor.cure(pet2);
    }

}
```

Dog, Cat 을 Pet 으로 형변환

상속과 접근지정자

앞에서 설명했듯이 public, protected 는 접근 가능, 나머지는 불가능

상속과 접근지정자

앞에서 설명했듯이 public, protected 는 접근 가능, 나머지는 불가능

→ Doctor 와 Pet 은 다른 package 에 있다고 가정해보자

상속과 접근지정자

```
class Dog extends Pet {
    public void bark() {
        …
    }

    @Override
    public void eatFeed() {
        super.eatFeed();
        bark();
    }

    protected void shakeTail() {
        …
    }
}
```

```
class Beagle extends Dog {

    @Override
    public void bark() {
        super.bark();
        …
    }

    @Override
    protected void shakeTail() {
        super.shakeTail();
        …
    }
}
```

상속과 접근지정자

```
public class AnimalDoctor {

    public void feed(Pet pet) {
        pet.eatFeed();
    }

    public void playWithDog(Dog dog) {
        dog.bark();
        dog.shakeTail();
    }

    public void playWithBeagle(Beagle beagle) {
        beagle.shakeTail();
    }
}
```

상속과 접근지정자

```
public class AnimalDoctor {

    public void feed(Pet pet) {
        pet.eatFeed();
    }

    public void playWithDog(Dog dog) {
        dog.bark();
        dog.shakeTail();
    }

    public void playWithBeagle(Beagle beagle) {
        beagle.shakeTail();
    }
}
```