JAVA 프로그램 실습

상속

경복대학교
소프트웨어융합과
배희호 교수

클래스들의 Relationship

- 종류
  - Has-a 관계 (포함 관계)
    - 하나의 클래스가 다른 클래스를 포함
    - 예) 자동차 has a 엔진, 은행고객 has a 계좌
    - Member 변수로 구현
  - Is-a 관계 (상속 관계)
    - 하나의 클래스가 다른 클래스를 상속
    - 예) 대학생 is a 학생, 관리자 is a 직원
    - 상속으로 구현
  - Use-a 관계 (고용 관계)
    - 하나의 클래스가 다른 클래스를 사용
    - 예) 프로그래머 use a 컴퓨터
    - 메소드나 생성자의 매개 변수로 전달

클래스들의 Relationship

- Has –a 관계
  - Association 관계
    - 연관 관계는 두 클래스가 서로를 알고 있고 서로 상호 작용하는 것을 의미
    - 예) 학생(Student)과 학교(School) 클래스
    - 학생은 학교에 다닐 수 있고, 학교는 여러 학생을 가질 수 있음

```
class Student {
    private School school;
    // ...
 }

class School {
    private List<Student> students;
    // ...
 }
```

클래스들의 Relationship

- Has –a 관계
  - Aggregation 관계
    - 집합 관계는 전체와 부분 간의 관계로, 부분 객체가 전체 객체에 속하지만, 전체 객체의 생명주기에 의존하지 않는 경우를 의미
    - 예) 자동차(Car)와 타이어(Tire) 클래스
    - 자동차는 여러 타이어를 가질 수 있지만, 타이어는 자동차가 없어도 존재할 수 있음

```
class Car {
   private List<Tire> tires;
   // …
 }

class Tire {
   // …
}
```

클래스들의 Relationship

- Has –a 관계
  - Composition 관계
  - 구성 관계는 더 강한 형태의 집합 관계로, 부분 객체가 전체 객체의 생명주기에 의존하는 경우를 의미
  - 예) 컴퓨터(Computer)와 CPU(CPU) 클래스
  - 컴퓨터가 없어지면 CPU도 존재할 수 없음

```
class Computer {
   private CPU cpu;
   // ...
}

class CPU {
   // ...
}
```

클래스들의 Relationship

- Is –a 관계
  - "Is-a" 관계는 객체 지향 프로그래밍에서 상속을 통해 나타내는 관계
  - 이 관계는 "하위 클래스는 상위 클래스의 한 종류다(is a kind of)"로 표현할 수 있음
  - 예) 모든 새는 동물이다.

```
class Animal {
   public void breathe() {
       System.out.println("Breathing...");
   }
}

class Bird extends Animal {
    public void fly() {
       System.out.println("Flying...");
    }
}
```

클래스들의 Relationship

- Use – a 관계
  - "Use-a" 관계는 클래스 간의 상호작용을 나타내며, 대체로 한 클래스가 다른 클래스의 서비스를 사용하거나 특정 기능을 위임 받을 때 사용
  - 예) "자동차는 엔진을 사용한다(Car uses an Engine)"라는 문장에서 자동차 클래스는 엔진 클래스의 기능을 사용할 수 있으며, 엔진은 자동차의 한 부분으로 동작

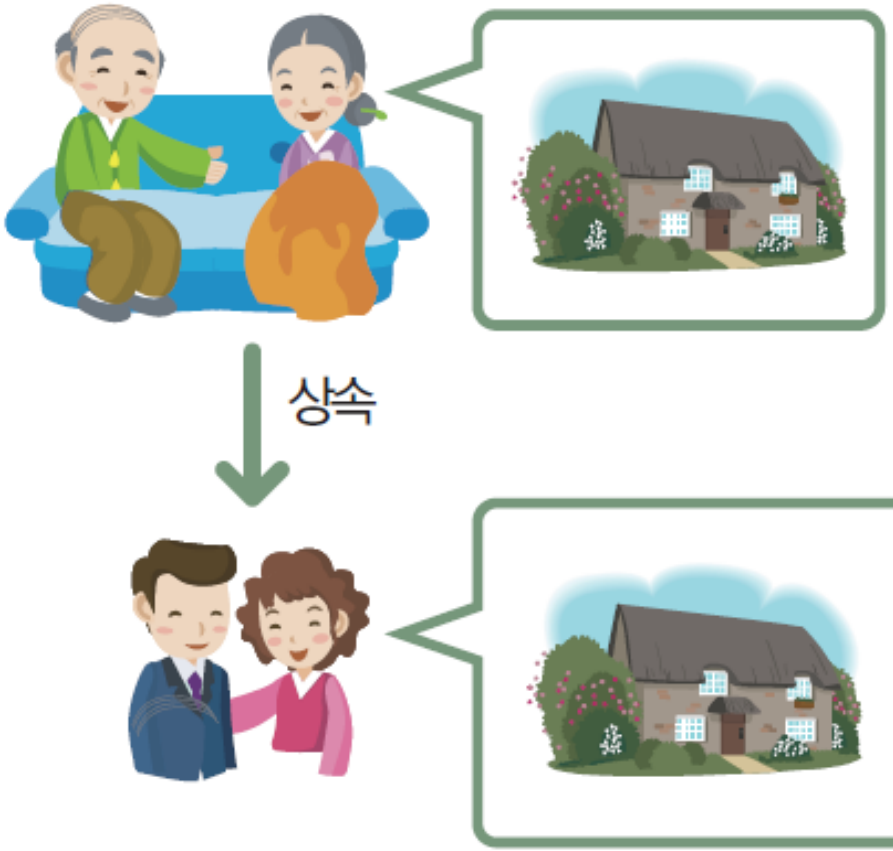
클래스들의 Relationship

- Use – a 관계

```
class Engine {
    public void start() {
       System.out.println("Engine starting...");
    }
 }

class Car {
    private Engine engine;

    public Car() {
        engine = new Engine(); // Car has-a Engine
    }

    public void startCar() {
       engine.start(); // Car uses Engine to start.
    }
}
```

Inheritance

- 상속(相續)의 개념은 현실 세계에도 존재



상속이란 자식이 부모가
가지고 있는 재산이나
권리를 물려받는다는 의미

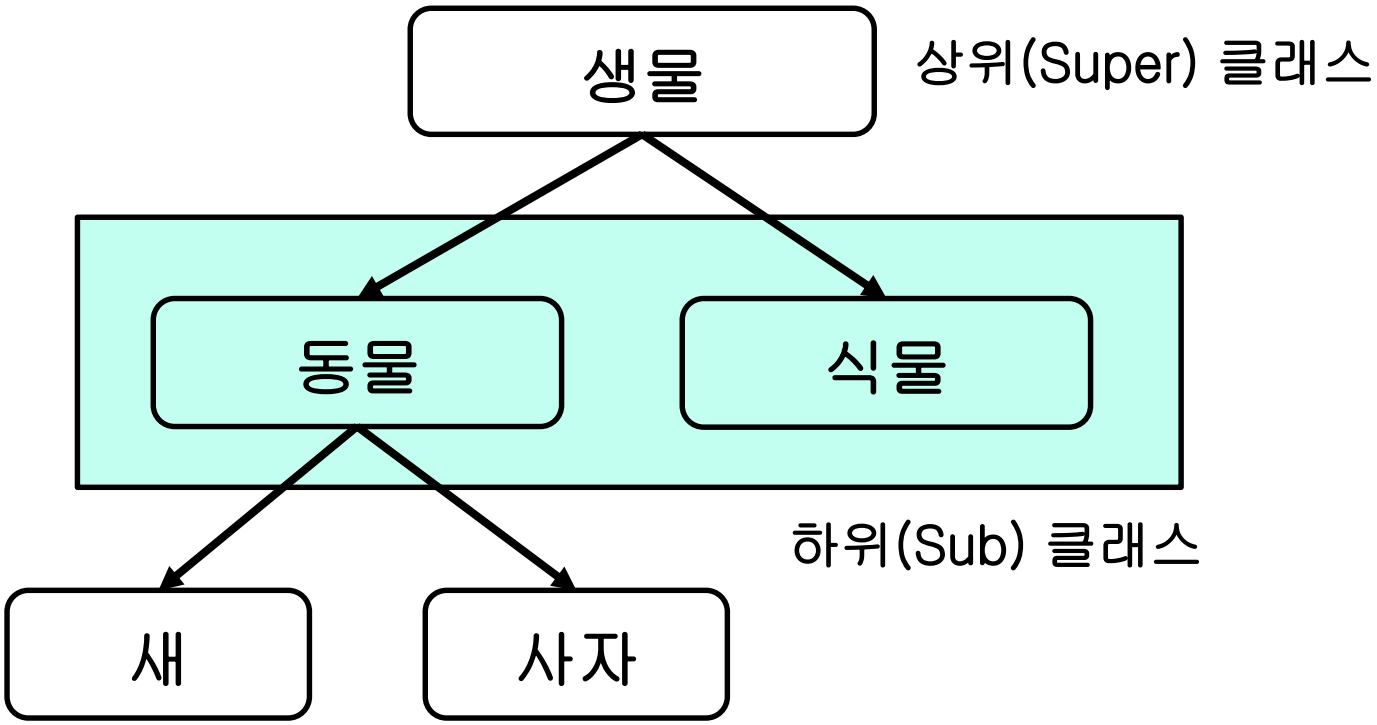
상속을 이용하면 쉽게
재산을 모을 수 있는
것처럼 소프트웨어도
쉽게 개발할 수 있다.

Inheritance

- 상위 클래스의 특성(Field, Method)을 하위 클래스에 물려주는 것
- Super class
  - 특성을 물려주는 상위 클래스
- Sub class
  - 특성을 물려 받는 하위 클래스
  - Super class에 자신만의 특성(필드, 메소드) 추가
  - Super class의 특성(메소드)을 수정
    - 구체적으로 Overriding이라고 부름
- Super class에서 Sub class로 갈수록 구체적
  - 예) 폰 -> 모바일 폰 -> 뮤직 폰
- 상속을 통해 간결한 Sub class 작성
  - 동일한 특성을 재정의할 필요가 없어 Sub class가 간결해 짐

Inheritance

- Super 클래스, Sub 클래스



일반화
-공통 속성 가짐
-속성이 간단

특수화
-부모 속성 상속
-개별 속성 추가
-속성이 많음

동물(자식) 클래스는 생물(부모) 클래스로부터 파생된 객체로서 생물 클래스가 가지고 있는 모든 속성들을 상속받아 사용 가능

Inheritance

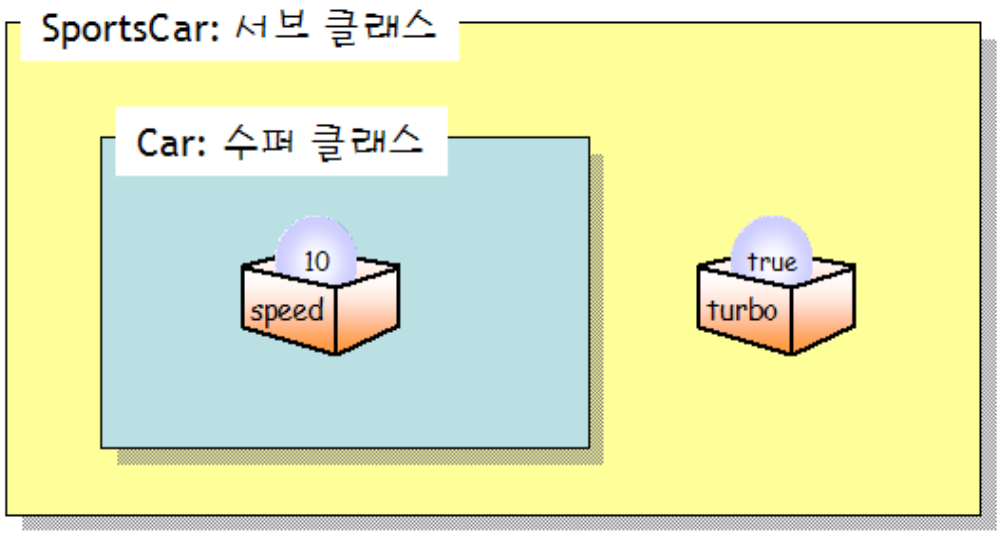
- 상속 관계로 만드는 방법
  - Generalization(일반화)
    - 다수의 클래스들간의 공통점을 발견하는 방법
    - 공통점을 가진 클래스는 부모 클래스로 선언하고, 각각의 차이점을 가진 클래스를 자식 클래스로 선언함
    - 예) 대학생, 고등학생 -> 학생
  - Specialization(전문화)
    - 일반화하고는 반대로 특정 클래스에서 하위 클래스를 생성
    - 예) 직원 -> 관리자, 엔지니어, 비서

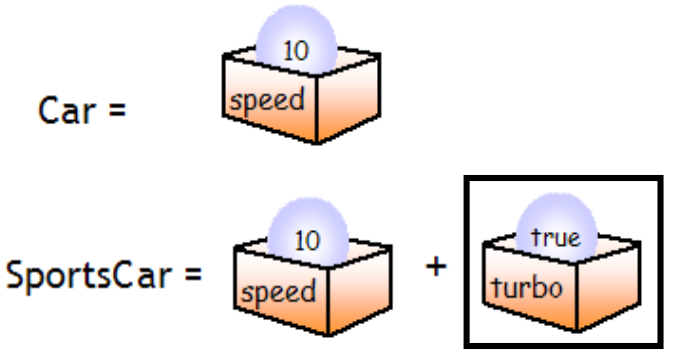
Inheritance

- 대부분의 실 세계는 많은 공통점을 가지고 있음
  - 다양한 공통점들은 많은 클래스들을 생성할 수 있게 하고, 각각의 클래스들은 연관 관계에 따라 계층구조를 이룰 수 있음
- Object Oriented Programming에서도 이와 비슷한 개념으로 쓰이는데, 여기에서는 부모 클래스에 정의된 Member를 자식 클래스가 물려받는 것을 말함
- OOP의 주요 특성인 Module의 재사용(reusing)과 Code의 간결성 제공
- 상속을 통해 기존에 있던 클래스(부모 클래스)를 이용하여 새로운 클래스(자식 클래스)를 만들 수 있음
  -> 기존의 것을 이용하여 만들어내기 때문에 적은 양의 Code로 새로운 클래스를 만들어 낼 수 있음

Inheritance

- 서브 클래스가 수퍼 클래스를 포함(집합 개념)





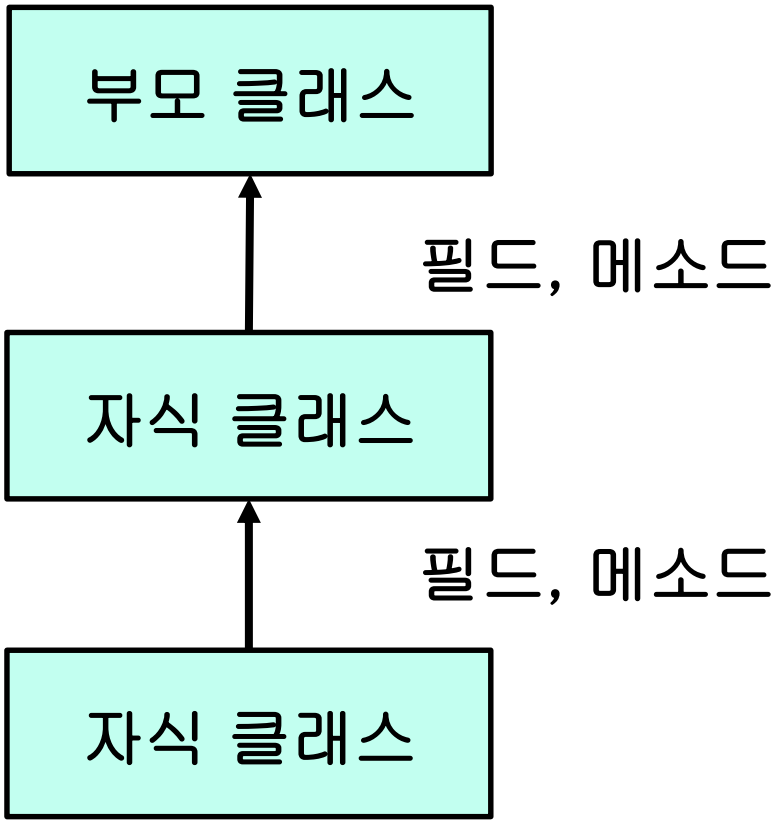
Inheritance

- 슈퍼(Super) 클래스, 서브(Sub) 클래스

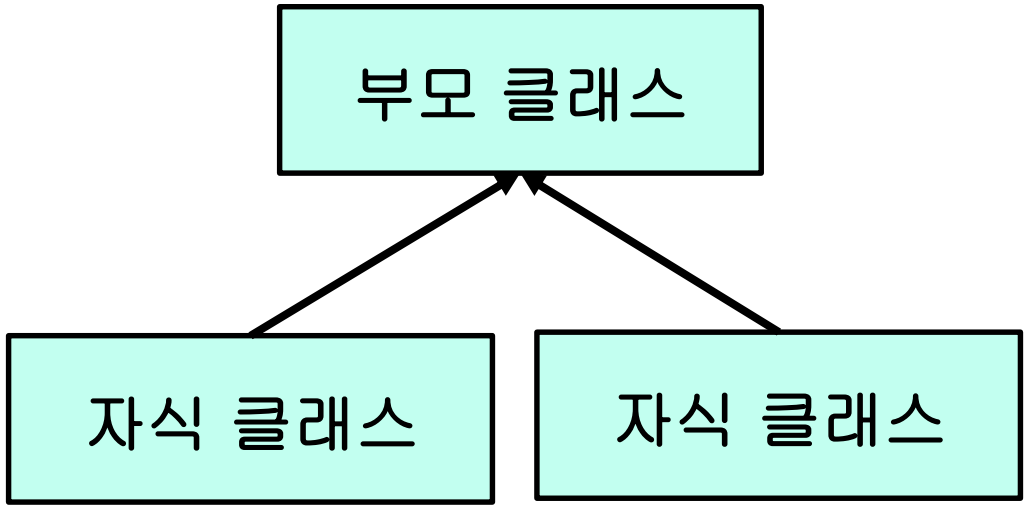
Super Class	Sub Class
Animal(동물)	Lion(사자), Dog(개), Cat(고양이)
Bike(자전거)	MountainBike(산악 자전거)
Vehicle(탈것)	Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motorcycle(오토바이), Bicycle(자전거)
Student(학생)	HighschoolStudent(고등학생), UnderGraduate(학부생)
Employee(사원)	Manager(관리자), Salesman(영업사원)
Shape(도형)	Rectangle(사각형), Triangle(삼각형), Circle(원형)

Inheritance

- Inheritance의 예



다단계 상속



계층적 상속

Inheritance

- JAVA 상속 종류
  - Class Inheritance(클래스 상속)
    - 부모 클래스의 Member 변수와 Method를 자식 클래스에서 상속 받는 것
  - Multiple Inheritance(다중 상속)
    - 한 클래스가 둘 이상의 부모 클래스로부터 상속 받는 것
    - JAVA는 단일 상속 개념으로서 하나의 자식이 여러 개의 부모를 동시에 상속받을 수 없음
      - 부모 클래스는 여러 개의 자식 클래스를 가질 수 있지만, 자식 클래스는 둘 이상의 부모 클래스를 가질 수 없음
    - JAVA에서는 Class 다중 상속은 허용하지 않지만, Interface 다중 상속은 가능

Inheritance

- JAVA 상속 종류
  - Interface Inheritance
    - Interface를 다른 Interface가 상속 받는 것
  - Abstract Class Inheritance
    - 추상 클래스에서 자식 클래스에서 반드시 구현해야 하는 추상 메소드를 상속 받는 것
  - Final Class Inheritance
    - final Keyword로 선언된 클래스는 상속할 수 없음

Inheritance

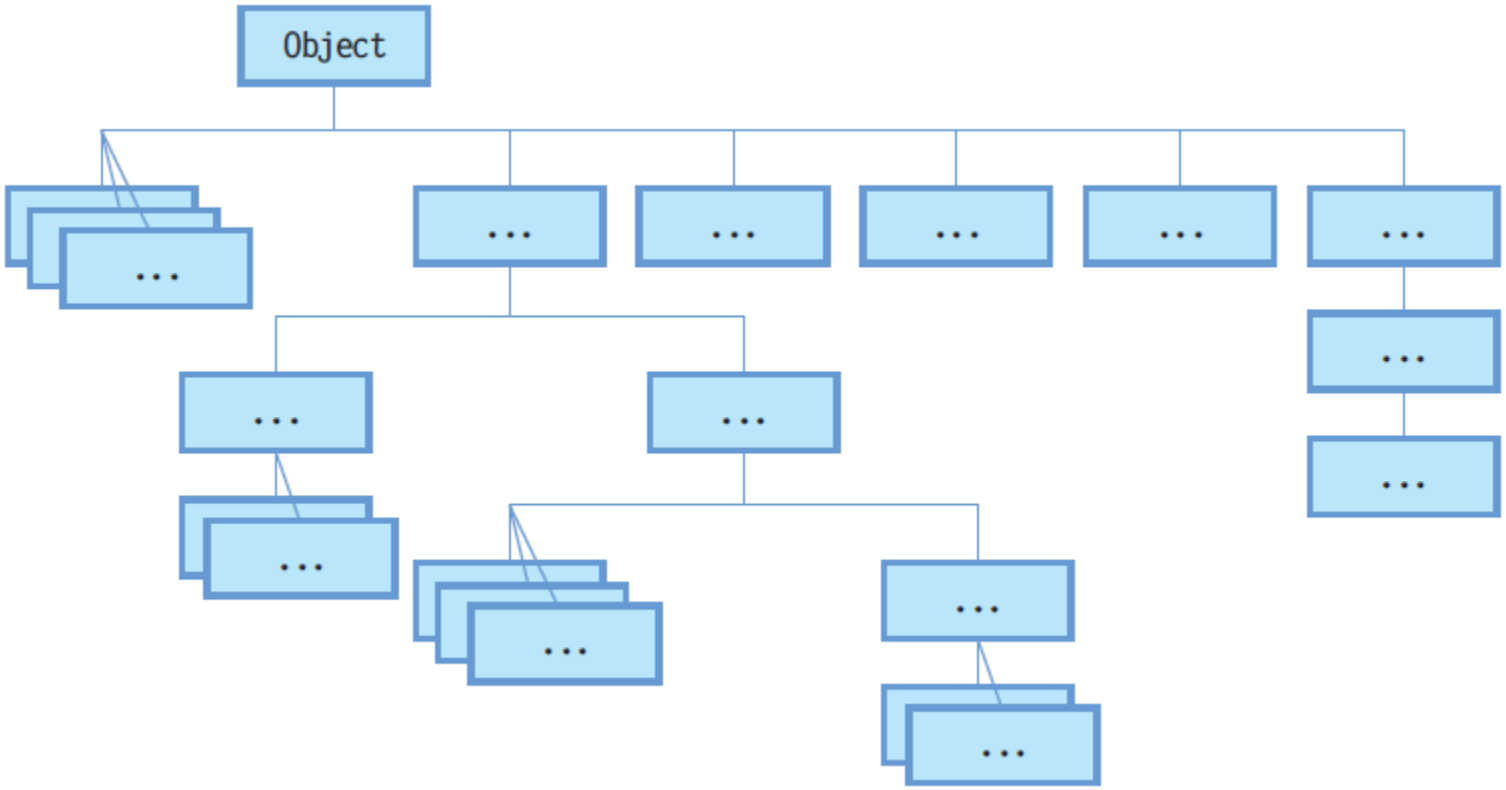
- JAVA에서도 상속은 비슷한 의미로 사용되고 있음
- 상속의 개념을 이용하여 클래스 계층 구조 구성
  - 상속을 해주는 부모(Parent) 클래스는 상위 클래스(슈퍼 클래스, Super Class) 또는 기반 클래스(베이스 클래스, Base Class)라 하며, 상속을 받는 자식(Child) 클래스를 하위 클래스(서브 클래스, Sub Class), 또는 파생 클래스(유도 클래스, Derived Class)라고 함
- Sub 클래스는 다른 클래스로부터 파생된 클래스를 나타내며, Super 클래스의 모든 상태(변수)와 행동(메소드)을 상속하게 됨
- Super 클래스는 클래스 계층 구조에서 바로 한 단계 위 클래스를 나타냄

Inheritance

- JDK에서 제공되는 클래스로부터 상속 받아 JAVA Program 작성
- JAVA에서의 모든 클래스는 Super 클래스를 가짐
  - JAVA Program의 최상위 클래스는 java.lang.Object 클래스
- JAVA에서 상속을 받게 해주려면, 새로운 클래스 이름 뒤에 “extends”와 상속받고자 하는 클래스를 입력해주면 됨

Inheritance

- JAVA에서는 모든 클래스는 반드시 java.lang.Object 클래스를 자동으로 상속받음



Inheritance

- 클래스들 사이의 상속은 소프트웨어 설계를 간단하게 할 수 있는 이점을 제공, 즉 기존의 클래스로부터 모든 요소를 상속 받고 새로운 클래스에는 추가되는 자료구조와 메소드만 지정하면 됨

- 다중 상속(multiple inheritance)
  - 클래스의 상속은 일반적으로 하나의 클래스로부터 상속되는 것이 일반적이지만, 다수개의 클래스로부터 상속받아 새로운 클래스를 생성하는 경우도 있음
  - JAVA에서는 다중 상속을 지원하지 않음

Inheritance

- 상속 선언

```
public class Person {
    …
}

public class Student extends Person {
                        // Person을 상속받는 클래스 Student 선언
    …
}

public class StudentWorker extends Student {
                        // Student를 상속받는 StudentWorker 선언
    …
}
```

Inheritance

- JAVA 상속의 특징
  - 클래스 다중 상속 지원하지 않음
    - 다수 개의 클래스를 상속받지 못함
  - 상속 횟수 무제한
  - 상속의 최상위 조상 클래스는 java.lang.Object 클래스
    - 모든 클래스는 자동으로 java.lang.Object를 상속받음

Inheritance

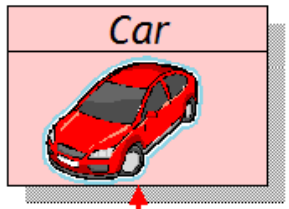
- 상속의 2가지 방식
  - extends를 이용한 단일 상속으로 추상 클래스로 상속 받아 재정의 하는 방식은 @Override 해줘도 되고 안해줘도 상관 없음
  - 각 메소드를 반드시 재정의 할 필요 없음
  - implements를 이용한 다중 상속인 경우에는 반드시 상속 받아 재정의 하지 않을 경우 Error가 나기 때문에 꼭 재정의 해서 메소드를 사용해야 함

Inheritance

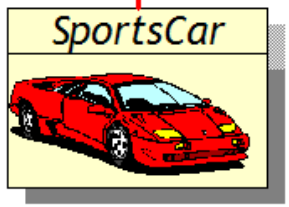
```
class SubClass extends SuperClass{
        ..............                  // 추가된 메소드와 필드
}
```

```
class Car {
        int speed;
}
class SportsCar extends Car {
        int turbo;
}
```

상속한다는 의미



수퍼클래스(superclass)



서브클래스(subclass)

Inheritance

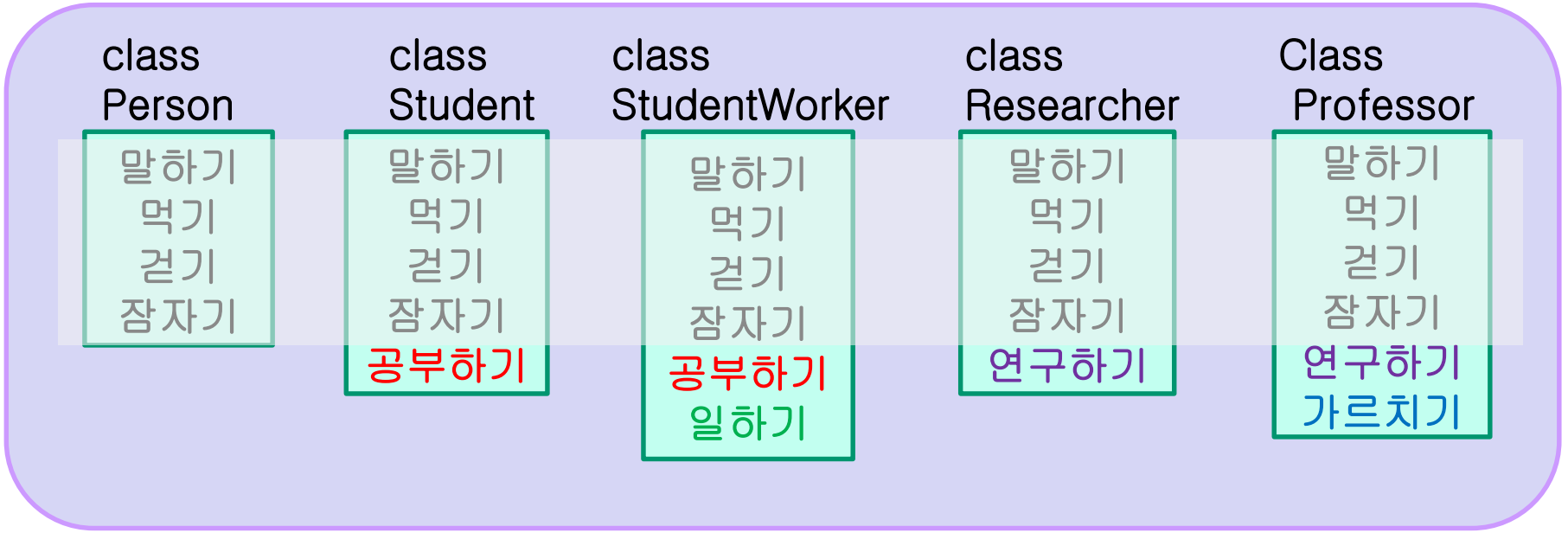
- 부모 클래스로부터 상속받은 자식 클래스는 부모 클래스의 자원(source) 모두를 사용 할 수 있음
  - 반대로 부모 클래스는 자식 클래스의 자원을 가져다 쓸 수는 없음
- 자식 클래스는 또 다른 클래스의 부모 클래스가 될 수 있음
- 자식 클래스는 부모 클래스로부터 물려받은 자원을 override하여 수정해서 사용 할 수 있음
- 부모 클래스가 상속받은 자원도 자식 클래스가 사용 가능함

Inheritance

- JAVA에서 Program 작성자가 명시적으로 상속되는 상위 클래스를 지정하지 않으면 묵시적으로 Object 클래스로부터 상속된 것으로 간주함
- 사용자는 어떤 일을 위해 가장 일반적인 클래스를 정의하고, 그 클래스로부터 상속된 새로운 클래스를 쉽게 생성할 수 있음
- 상속되어 생성되는 클래스는 이미 작성된 일반적인 클래스의 모든 요소를 자동 상속받고, 특정한 일을 위해 필요로 되는 새로운 요소가 추가되어 생성

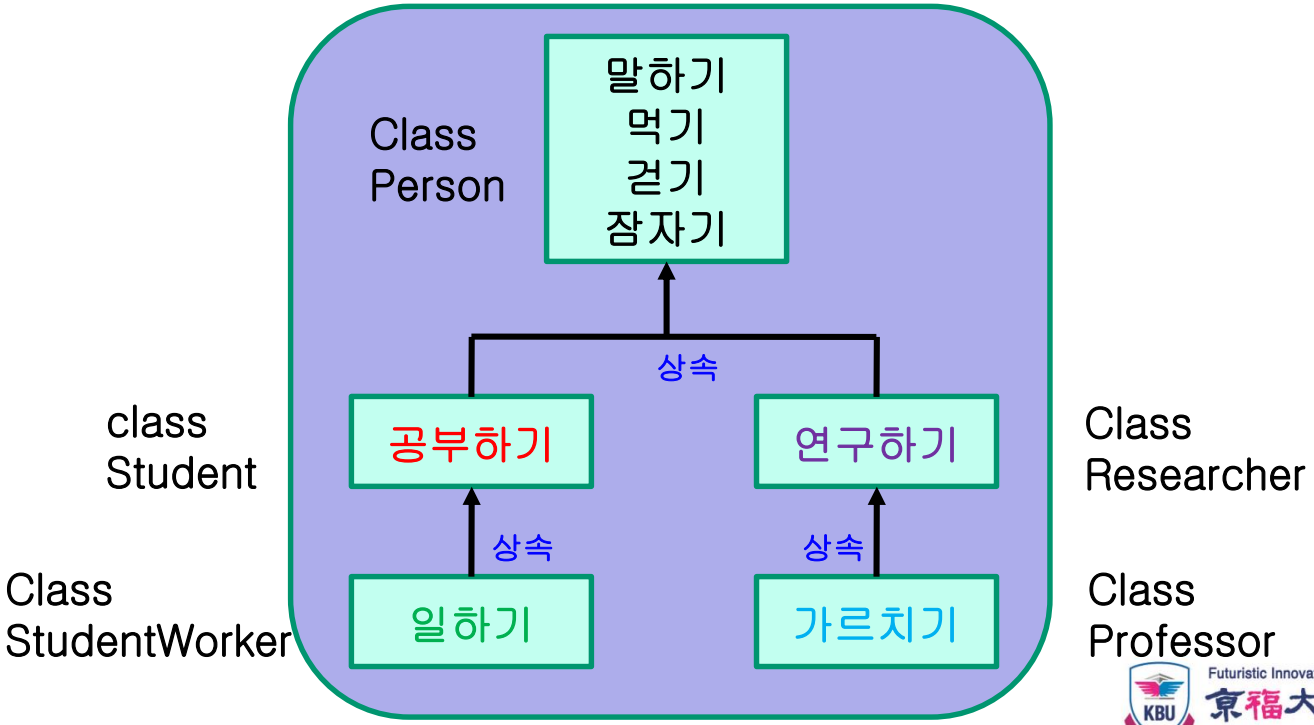
Inheritance

- 상속의 필요성
  - 상속이 없는 경우
    - 중복된 멤버를 가진 5개의 클래스

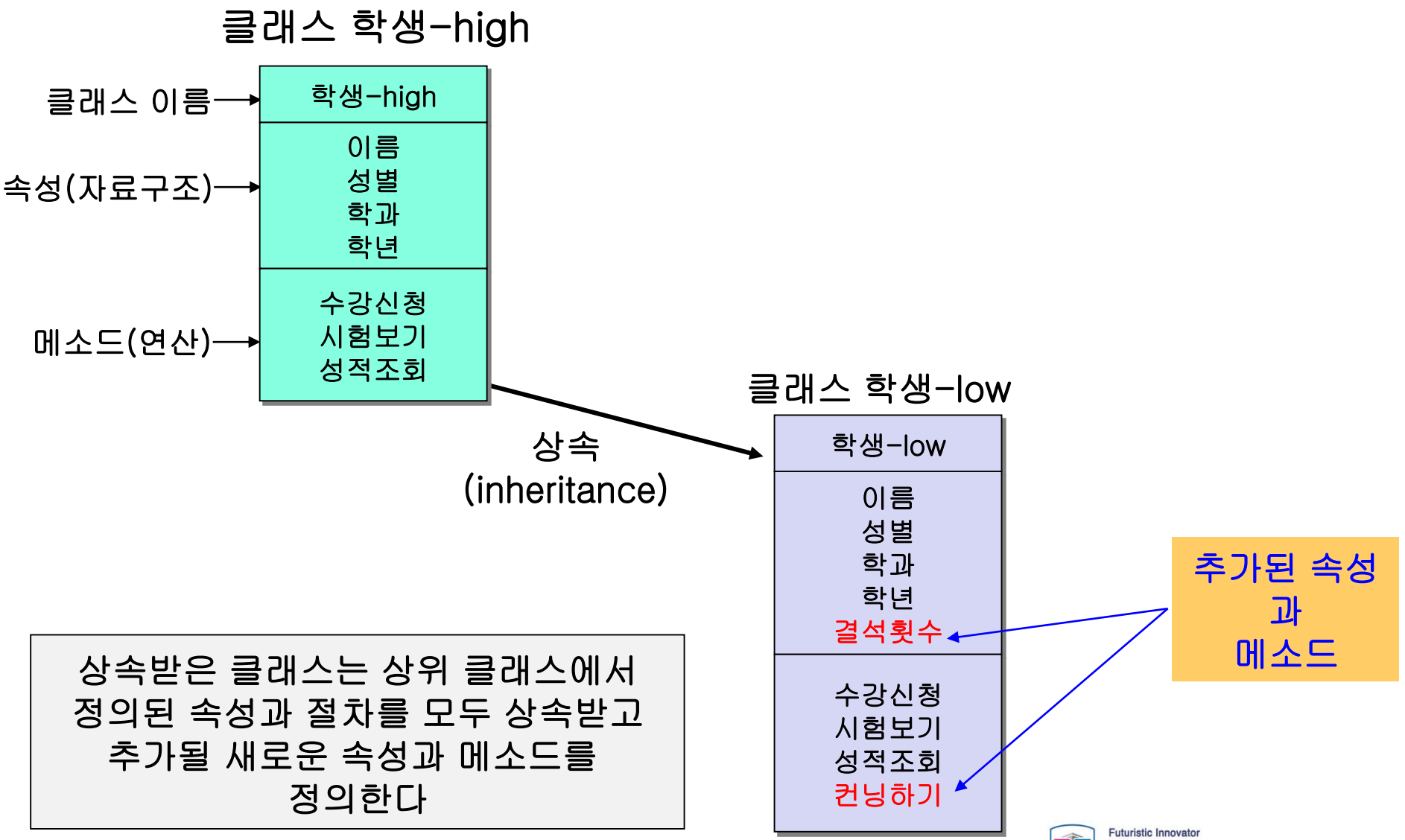


Inheritance

- 상속의 필요성
  - 상속을 이용한 경우
    - 클래스 사이의 멤버 중복 선언 방지
    - 필드와 메소드 재사용으로 클래스 간결화
    - 클래스 간 계층적 분류 및 관리



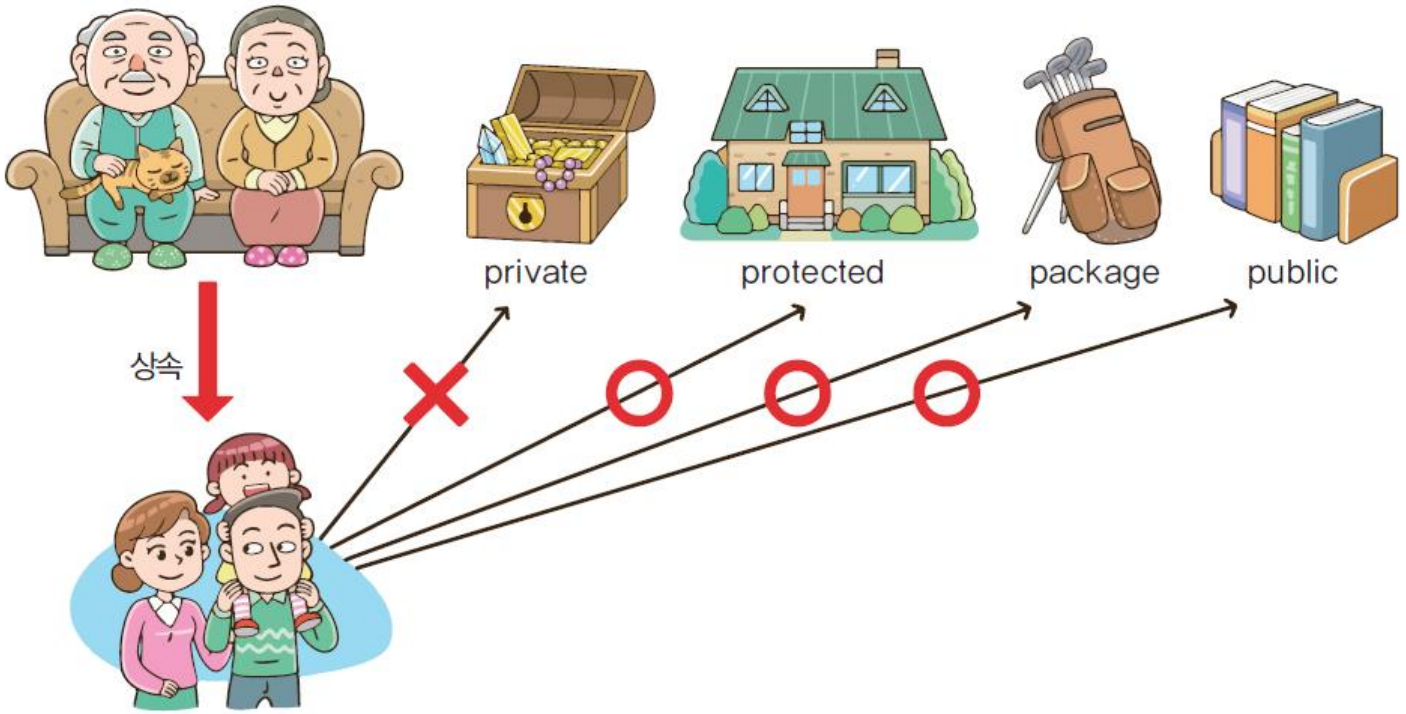
Inheritance



상속받은 클래스는 상위 클래스에서 정의된 속성과 절차를 모두 상속받고 추가될 새로운 속성과 메소드를 정의한다

Inheritance

- 자식 클래스는 부모 클래스의 public 멤버, protected 멤버, package 멤버(부모 클래스와 자식 클래스가 같은 패키지에 있다면)를 상속 받음
- 부모 클래스의 private 멤버는 상속되지 않음



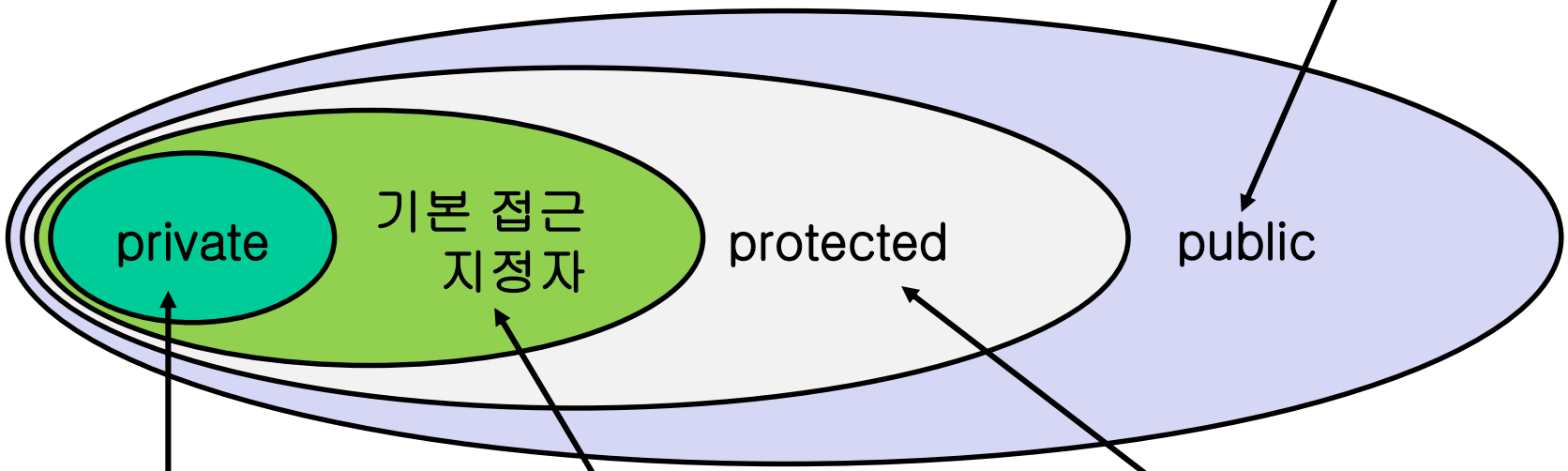
Inheritance

- 접근 지정자

접근 지정자	자신 클래스	같은 패키지	하위 클래스	다른 패키지
private	O	X	X	X
생략 (기본 접근 지정자)	O	O	X	X
protected	O	O	O	X
public	O	O	O	O

Inheritance

- 접근 지정자



Inheritance

- 상속의 장점
  - 상속을 통하여 기존 클래스의 필드와 메소드를 재사용
  - 코드의 간결함에 따른 개발 시간 단축
    - 기존 클래스의 일부 변경도 가능
    - 상속을 이용하게 되면 복잡한 GUI 프로그램을 순식간에 작성 가능
  - 상속은 이미 작성된 검증된 소프트웨어를 재사용
    - 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수 간편
    - 코드의 중복을 줄일 수 있음

Inheritance

- DRY 원칙
  - DRY는 '반복하지 마라'라는 뜻의 Don't Repeat Yourself의 첫 글자를 모아 만든 용어로 간단히 말해 동일한 지식을 중복하지 말라는 것
  - 중복 코드는 변경을 방해함
    - 이것이 중복 코드를 제거해야 하는 가장 큰 이유임
  - 중복 코드가 가지는 가장 큰 문제는 코드를 수정하는 데 필요한 노력을 몇 배로 증가 시킨다는 것
  - 중복 여부를 판단하는 기준은 변경 임
    - 요구사항이 변경 됐을 때 두 코드를 함께 수정해야 한다면 이 코드는 중복 임
  - DRY 원칙은 한번, 단 한번(Once and Only One) 원칙 또는 단일 지점 제어(Single-Point Control) 원칙이라고도 부름

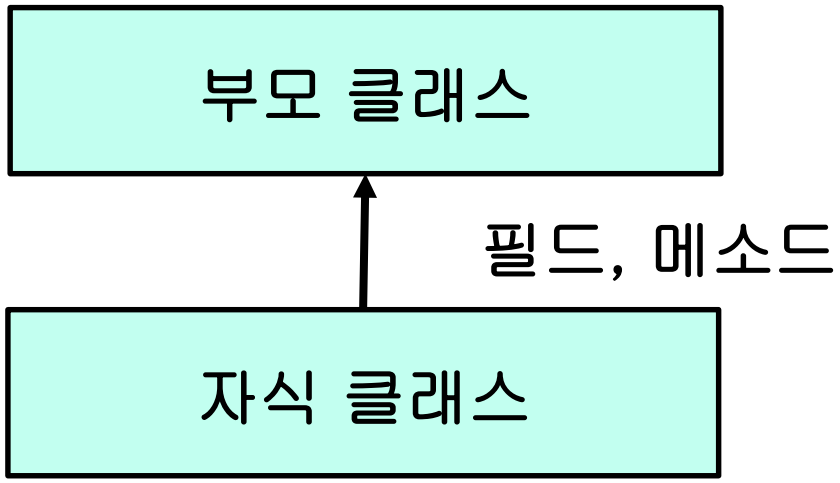
Inheritance

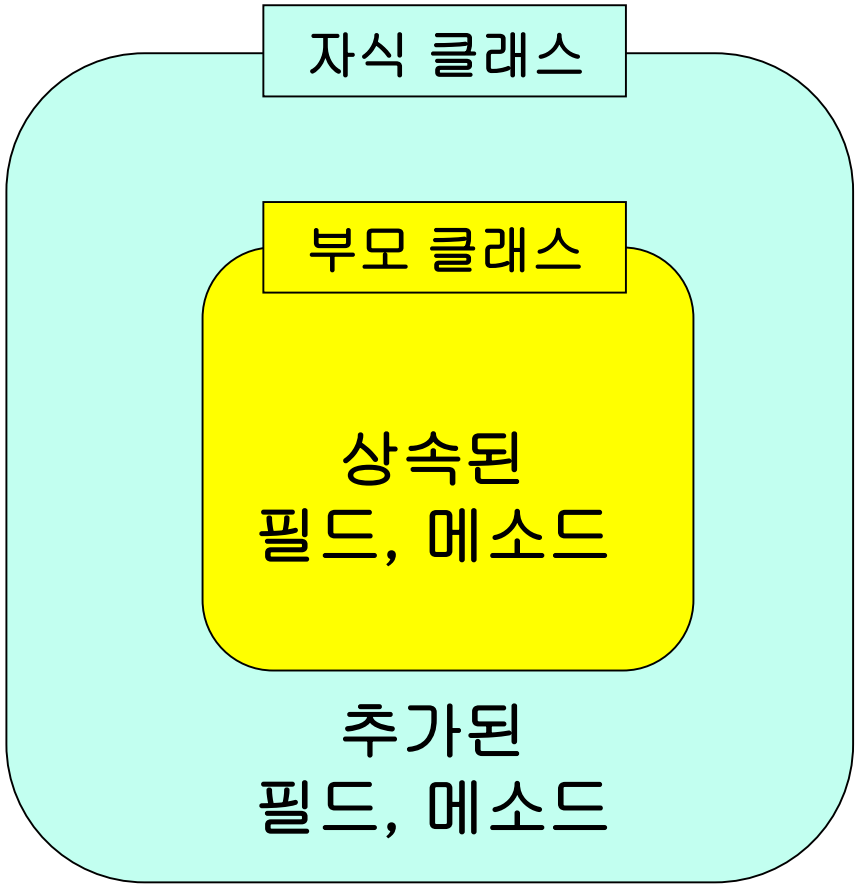
- 기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것
  - 기존의 클래스 = Parent(Super) Class
  - 새로운 클래스 = Child(Sub) Class
- 두 클래스는 Parent Class와 Child Class의 관계로 맺어지며, Sub Class는 Super Class로부터 Super Class의 멤버(변수, 메소드)를 상속 받음

```
public class Sub Class 이름 extends Supper Class 이름 {
    Sub Class에 추가하는 멤버 ..

    Sub Class 생성자(인수 목록) {
        ..
    }
}
```

Inheritance

- 자식 클래스가 부모 클래스의 내용(필드, 메소드)을 고스란히 획득하는 것





Inheritance

- this
  - 자기자신의 인스턴스를 가리키는 변수
  - 용도
    - 다른 생성자 호출
      - 생성자 첫 라인에서 다른 생성자를 호출
      - this();, this(args);
    - 자신의 멤버 호출
      - 로컬 변수와 인스턴스 변수 이름이 동일한 경우
      - this.멤버

```
String name;                    //인스턴스 변수
public void setName(String name){
        this.name = name;
}
```

Inheritance

- super
  - 부모 클래스의 Instance를 가리키는 변수
- 용도
  - 부모 생성자 호출
    - 자식 생성자 첫 Line에서 부모 생성자를 호출
    - 명시적으로 호출하지 않으면 super()가 자동으로 삽입
    - super();, super(args);
  - 부모 클래스 Member 호출
    - 자식 클래스에서 부모 클래스의 Member를 호출
    - super.name;, super.getName();

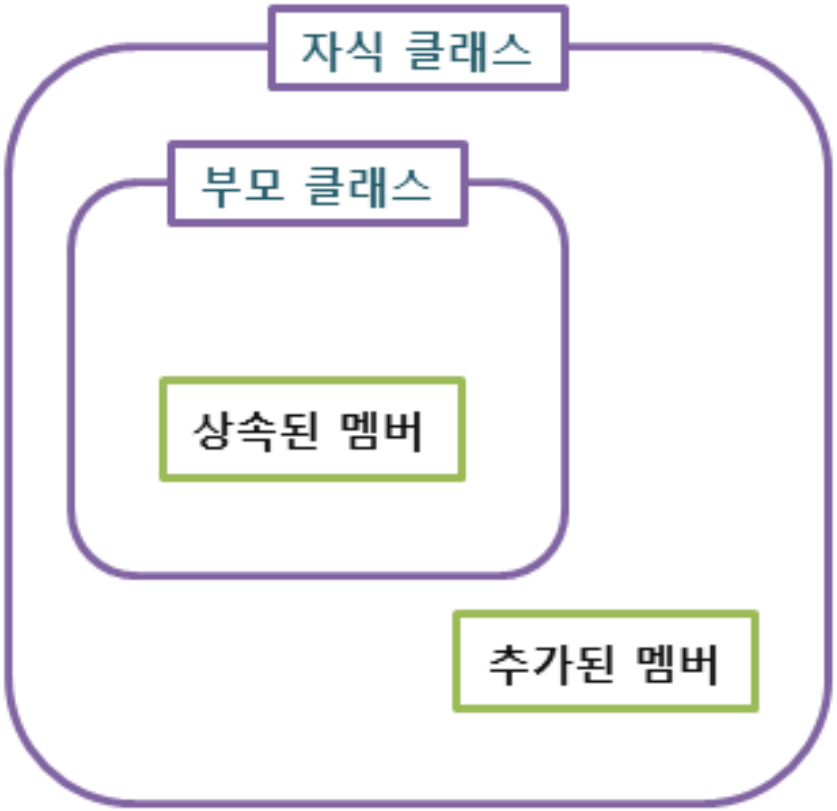
Inheritance

- super 예약어를 사용하면서 주의 할 점
  - 반드시 자식 클래스의 생성자 첫 라인에서 부모의 생성자를 호출해야 함
  - 자식 클래스의 생성자 내에서 반드시 부모의 생성자를 호출해야 함
  - 명시적으로 자식의 클래스에서 부모의 생성자를 호출하지 않아도 super가 자동 삽입되어 부모 클래스의 생성자를 호출

Inheritance

- 메소드 Overriding
  - Sub 클래스에서 메소드를 Super 클래스의 기존 메소드와 signature 즉, 이름, 매개 변수의 자료 형과 매개 변수의 개수를 동일하게 정의
  - Sub 클래스에서 메소드는 Super 클래스의 접근 제어보다 더 좁아질 수 없음

Inheritance



Inheritance

- Parent Class와 Children Class는 JAVA 지정 예약어 extends에 의하여 정해짐
- 하나의 Parent Class는 여러 개의 Children Class를 가질 수 있음
- 반대로 하나의 Class는 여러 개의 Class로부터 상속을 받을 수는 없음
- Parent Class로부터 상속받은 Children Class는 Parent Class의 자원(Resource) 모두를 사용 할 수 있음 (허용한 것)
- Children Class는 Parent Class로부터 물려받은 자원을 override하여 수정해서 사용 할 수도 있음
- 반대로 Parent Class는 Children Class의 자원을 가져다 쓸 수는 없음
- Children Class는 또 다른 Class의 Parent Class가 될 수 있음

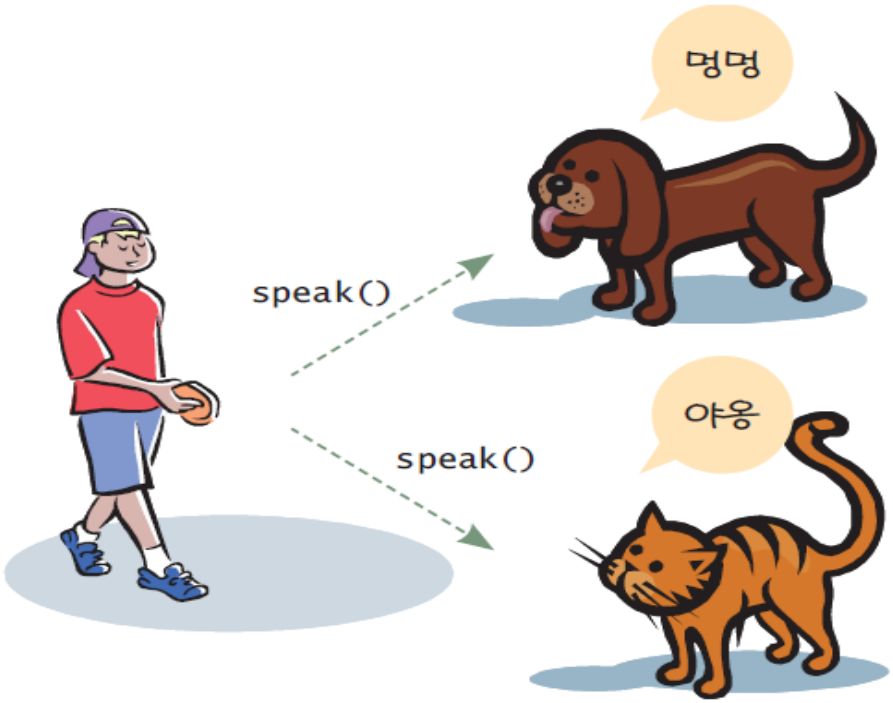
this, super

- this
  - 현재 클래스의 Instance를 의미
  - 즉, 현재 클래스의 Member 변수를 지정할 때 사용
- this()
  - 현재 클래스에 정의된 생성자를 호출할 때 사용

- super
  - 자식 클래스에서 상속받은 부모 클래스의 Member 변수를 참조할 때 사용
- super()
  - 자식 클래스가 자신을 생성할 때 부모 클래스의 생성자를 불러 초기화 할 때 사용
  - 기본적으로 자식 클래스의 생성자에 추가

Polymorphism

- Object들의 Type이 다르면 똑같은 Message가 전달되더라도 서로 다른 동작을 하는 것



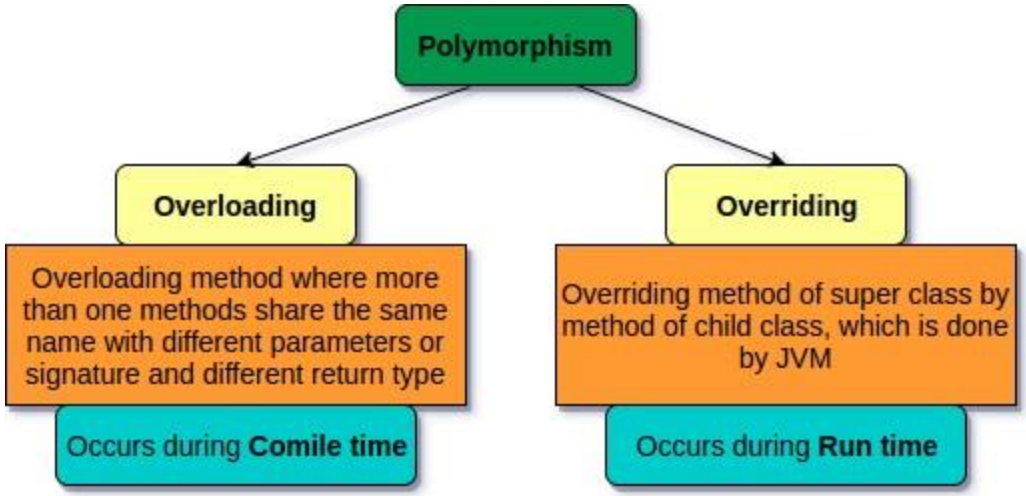
다형성의 개념

Polymorphism

- Polymorphism의 어원
  - Greek : "*having many forms*"
  - OOP : "*many methods with the same signature*"
- 특성
  - 연관성 있는 2개 이상의 용도를 하나의 이름으로 사용
  - 예) Window 클래스와 각 하위 클래스에서 display() 메소드를 정의하고 있으면, 이 display() 메소드를 다형성을 가진 메소드라 함
- Overriding
  - 상속 계층에서, 자식 클래스가 같은 시그너처의 부모 메소드를 재정의(override)
    - 예) MenuWindow가 자신의 display() 메소드를 정의한다면, 부모 클래스 Window에서 사용하는 display() 메소드를 MenuWindow가 재정의

Polymorphism

- 하나의 객체가 여러 가지 Type을 가질 수 있는 것을 의미
- JAVA에서는 이러한 다형성을 Parent Class 타입의 참조 변수로 Child Class 타입의 Instance를 참조할 수 있도록 하여 구현
- 다형성은 상속, 추상화와 더불어 Object Oriented Programming을 구성하는 중요한 특징 중 하나임



Method Overriding

- JAVA에서는 Object-Oriented 개념에서 정의하고 있는 특성 중 하나인 Polymorphism을 제공해 주기 위해, Method Overriding 및 Method Overloading를 할 수 있도록 함
- 하위 클래스는 상위 클래스로부터 상속되는 State(변수)와 Behavior(메소드)들을 가질 수 있으며 하위 클래스는 자신에게 필요한 변수들과 메소드를 추가적으로 정의할 수 있음
- 하위 클래스는 상위 클래스에서 정의된 메소드와 같은 이름, 같은 인자들을 갖는 새로운 메소드를 정의하여 상위 클래스에서 상속되는 메소드를 재정의(overriding)할 수 있음

Method Overriding

- 클래스들의 상속 관계에서 상위 클래스에 선언된 메소드는 하위 클래스에서 사용할 수 있음
- 서브 클래스가 필요에 따라 상속된 메소드를 다시 정의하는 것을 Method Overriding이라고 함
- Overriding은 상위 클래스의 메소드와 하위 클래스의 메소드가 메소드 이름은 물론 매개변수의 타입과 개수까지도 같아야 함
- Overriding과는 또다른 개념, 즉 Overloading은 같은 클래스 내에서 같은 이름의 생성자나 메소드를 사용하여 그 매개 변수를 달리 하여 사용하는 경우이고, Overriding은 상속 관계에 있는 클래스들간에 같은 이름의 메소드를 재정의하여 사용하는 경우임

Method Overriding

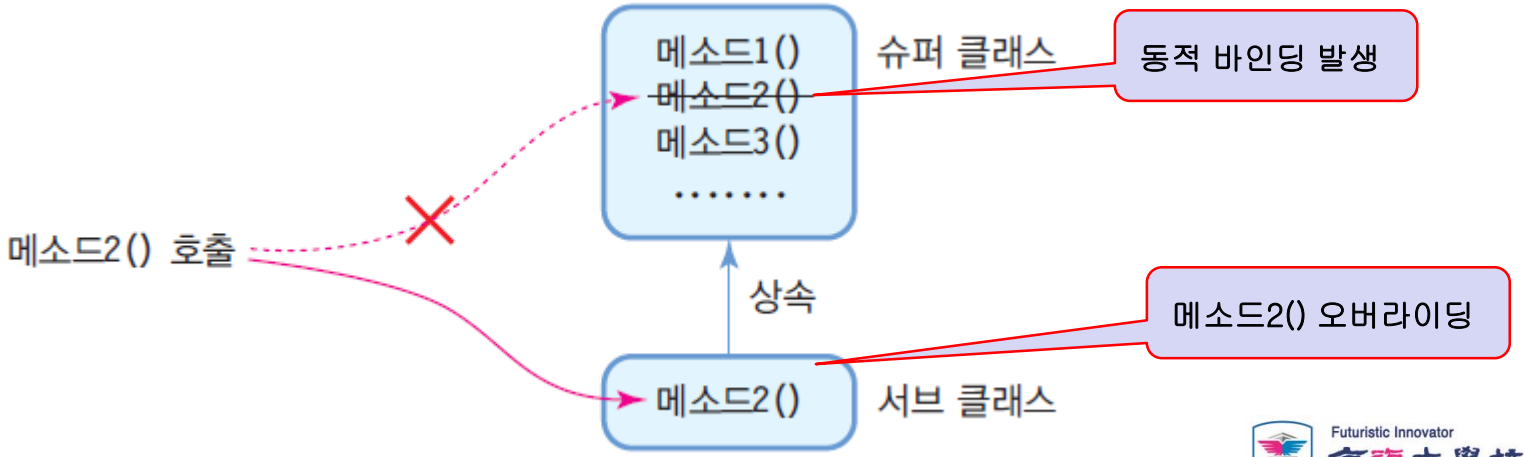
- 상속 관계의 클래스에서 Method가 Overriding 되었다면 상위 클래스의 메소드가 하위 클래스에 의해 가려지게 됨
- 이때 하위 클래스의 객체에서 상위 클래스에서 Overriding 된 메소드를 사용하려면 예약어 super를 이용
- 이렇게 하위 클래스가 상위 클래스의 Instance 메소드를 새로 구현함으로써 상위 클래스에서 제공해주고 있는 메소드를 하위 클래스에 맞게 새롭게 구현할 수 있는 것임

Method Overriding

- 정의
  - 상속 관계 시 부모의 메소드를 자식이 수정하여 재정의 한 메소드
- 규칙
  - 메소드 이름은 동일
  - 반환 Data Type은 동일
  - 매개 변수가 동일 (순서, Data Type, 개수)
  - 접근 지정자는 확대만 가능(축소 불가)
  - Exception 처리는 축소만 가능(확대 불가)
  - static, final, private 메소드는 overriding이 불가능
- 용도
  - 자식 클래스에서 추가 작업이 필요한 경우에 수정하여 사용

Method Overriding

- 슈퍼 클래스의 메소드를 서브 클래스에서 재정의
  - 슈퍼 클래스의 메소드 이름, 메소드 인자 타입 및 개수, 리턴 타입 등 모든 것 동일하게 작성
    - 이 중 하나라도 다르면 메소드 Overriding 실패
- "메소드 무시하기"로 번역되기도 함
- 동적 Binding 발생
  - 서브 클래스에 Overriding된 메소드가 무조건 실행되도록 동적 Binding 됨



Overriding vs. Overloading

재 정의(overriding)	다중 정의(overloading)
메소드가 하위 클래스에서 정의	메소드가 같은 클래스에서 정의
메소드의 이름과 인자값이 동일	메소드의 이름은 동일하나 인자값 (개수 또는 데이터형)은 다름
메소드의 리턴형이 같음	메소드의 리턴형은 다를 수 있음
같은 메소드 시그니쳐	다른 메소드 시그니쳐
하위 메소드의 접근 범위가 상위 메소드의 접근 범위보다 커야함	접근 범위는 관계없음
예외 객체의 발생 시 같은 예외 형식이거나 같은 종류의 예외 형식이어야 함	예외 처리 관계없음

Upcasting, Downcasting

- Reference 변수들 사이의 형 변환은 상속 관계에서만 가능
- 상속 관계에 있는 클래스 사이의 형 변환은 UpCasting(업 캐스팅)과 DownCasting(다운 캐스팅) 2가지로 구분됨

```
class Child extends Parent{
 . .
}
```

Upcasting

- Program에서 이루어지는 자동 타입 변환
- 서브 클래스의 레퍼런스 값을 슈퍼 클래스 레퍼런스에 대입
  - 슈퍼 클래스 레퍼런스가 서브 클래스 객체를 가리키게 되는 현상
  - 객체 내에 있는 모든 Member를 접근할 수 없고, 슈퍼 클래스의 Member만 접근 가능

```
class Person {
}

class Student extends Person {
}
…

Student student = new Student();
Person person = strudent; // 업캐스팅, 자동 타입 변환
```

Downcasting

- Super 클래스 Reference를 Sub 클래스 Reference에 대입
- Upcasting된 것을 다시 원래대로 되돌리는 것
- 명시적으로 타입 지정

```
class Person {
}

class Student extends Person {
}
…

Person person = new Person();
Student student = (Student) person;  // 다운캐스팅, 강제 타입 변환
```

Upcasting/Downcasting 필요성

- 여러 Type(클래스)의 인자를 전달 받고자하는 메소드는 인자를 최상위 클래스인 Object Type으로 선언
  - 예) math.h에 선언된 수학 함수는 모두 double Type 사용
  - Upcasting : always safe
- method_name(Object obj)
  - ex) equals(new String(“abc”));
  - ex) equals(new Integer(123));
- 모든 종류의 객체를 전달 받을 수 있음
- 메소드 구현 시 각 타입 별로 구분해서 처리함
  - Downcasting
  - instanceof 연산자 : 객체의 타입 조사
    - ex) str instanceof Object ： true

Upcasting 예제

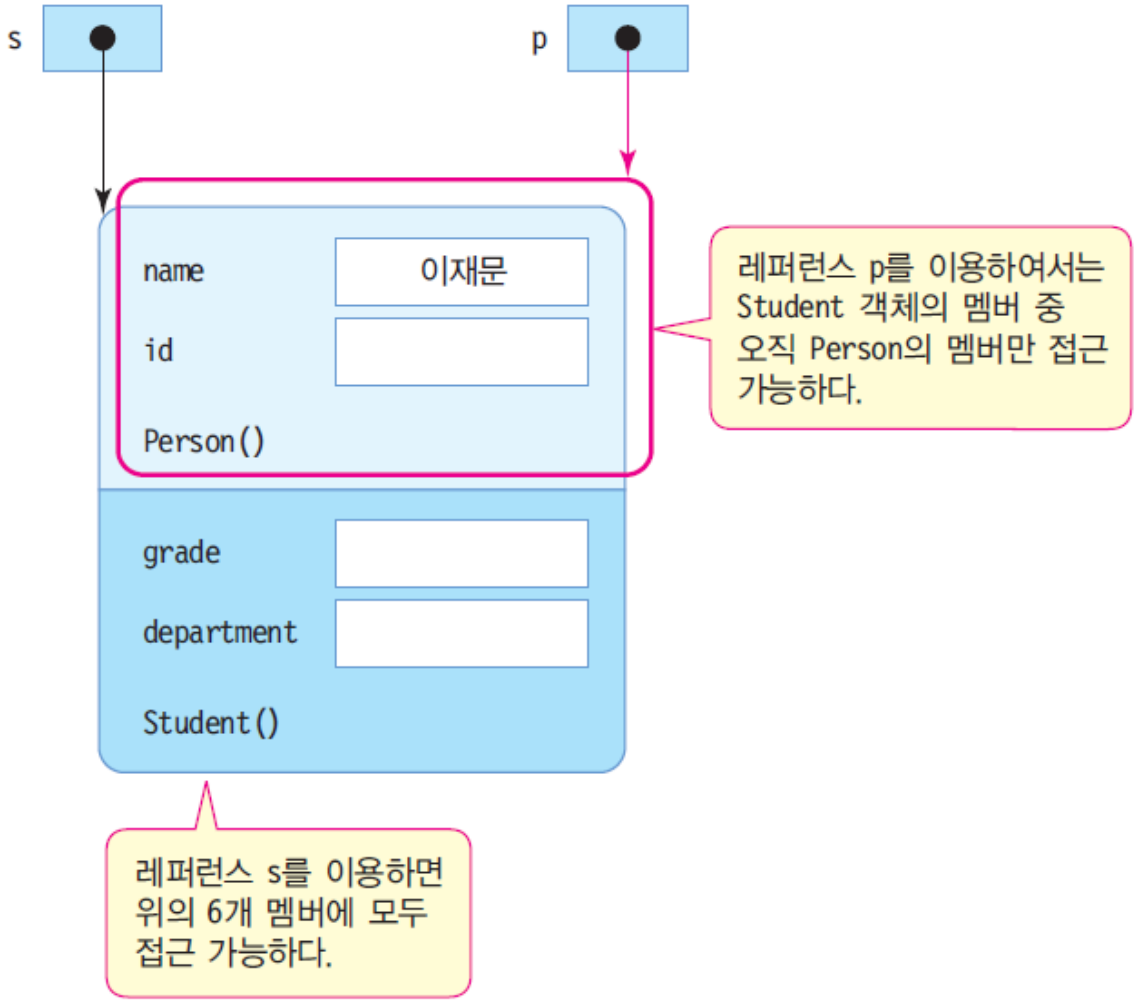
```
class  Person {
  String name;
  String id;

  public Person(String name) {
    this.name = name;
  }
}

class  Student extends Person {
  String grade;
  String department;

  public Student(String name) {
    super(name);
  }
}
```

Upcasting 예제

```
public class Main {
  public static void main(String[] args) {
        Person  person;
        Student student = new Student("이재문");
        person = student;                    // 업캐스팅 발생

        System.out.println(p.name);    // 오류 없음

        person.grade = "A";                  // 컴파일 오류
        person.department = "Com";  // 컴파일 오류
  }
}
```

이재문

Upcasting 예제



Downcasting 예제

```
public class Main {
  public static void main(String[] args) {
        Person person = new Student("이재문"); // 업캐스팅 발생
        Student student;
        student = (Student) person;          // 다운케스팅 발생

        System.out.println(p.name);    // 오류 없음

        student.grade = "A";               // 오류 없음
        student.department = "Com";  // 오류 없음
  }
}
```

이재문

상속 예제 1

- 자동차(Car)의 기능을 정의하는 클래스가 있다. 이 때 특수한 자동차인 경주용 차(Racing Car)의 클래스를 생성해보자

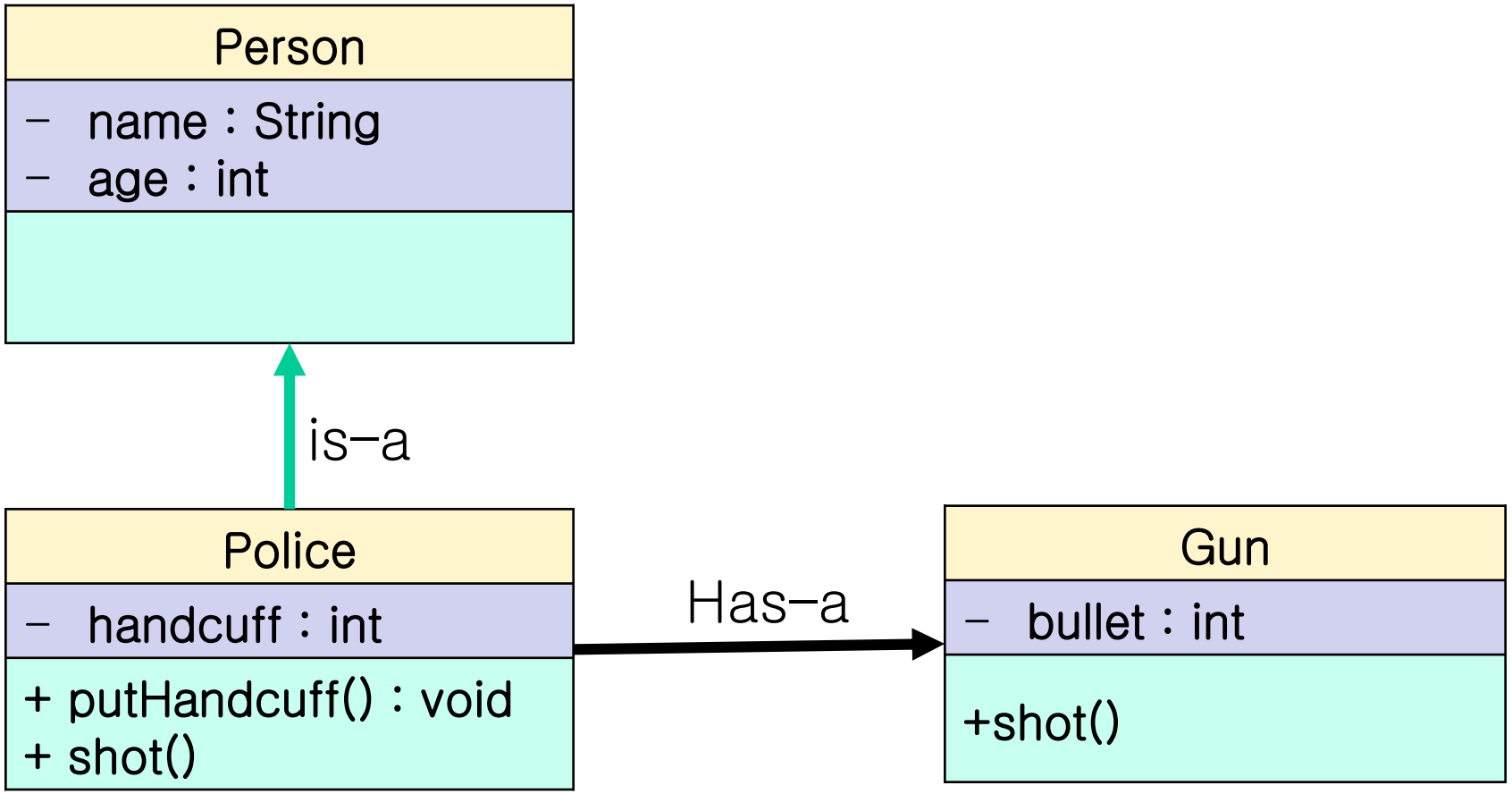
- 문제 분석
  - 경주용 차는 자동차의 일종이므로 많은 공통점을 가지고 있다. 따라서 자동차 클래스를 바탕으로 경주용 차 클래스를 만들 수 있음
    - Car Class -> Super Class
    - RacingCar Class -> Sub Class

```
class Car { }
class RacingCar extends Car { }
```

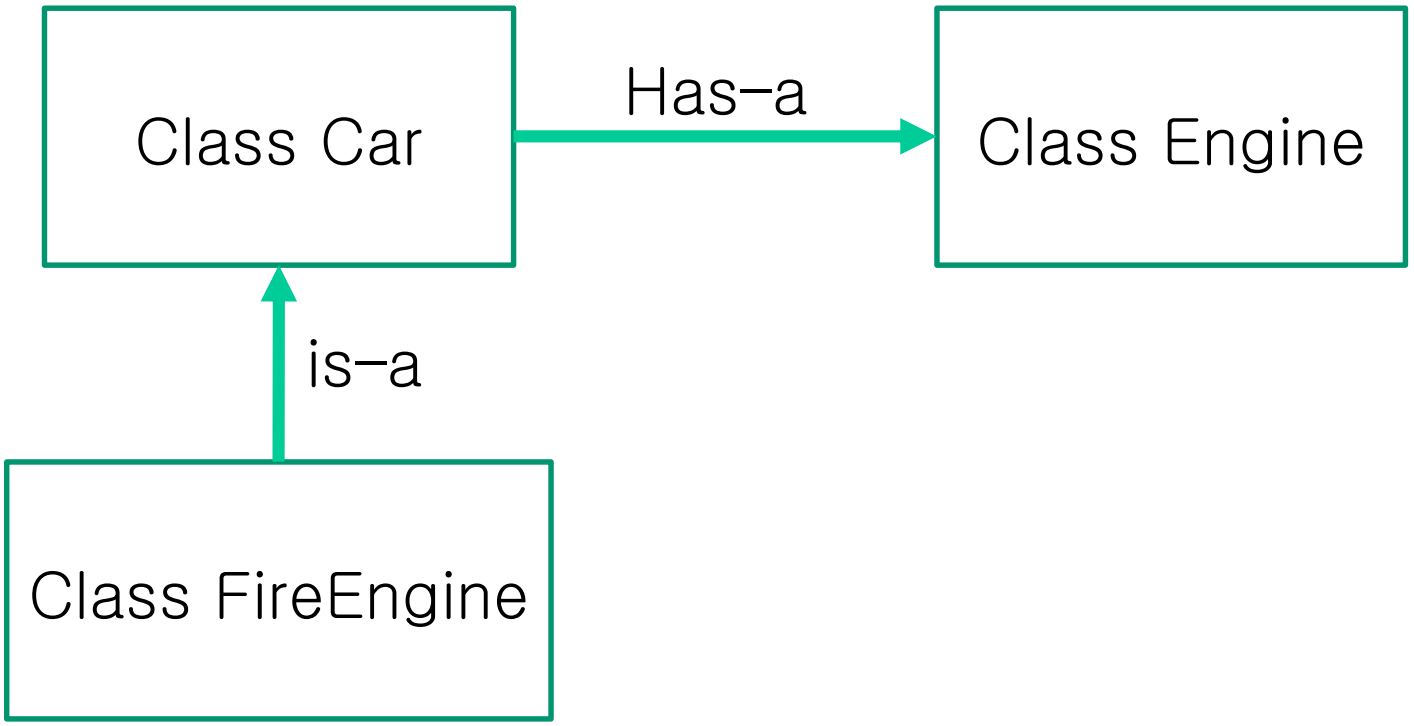
- Computer와 Notebook과 Tablet이 서로 is-a관계
  - 어떤 걸 부모 클래스로 해야할 지 감이 오지 않는다면, 이렇게 생각하면 됨
  - “Notebook은 Computer이다”는 틀리지 않지만, “Computer는 Notebook이다”는 뭔가 이상하지 않은 가?

상속

- 경찰관은 총과 수갑을 소지하기도 한다. (has-a 관계)



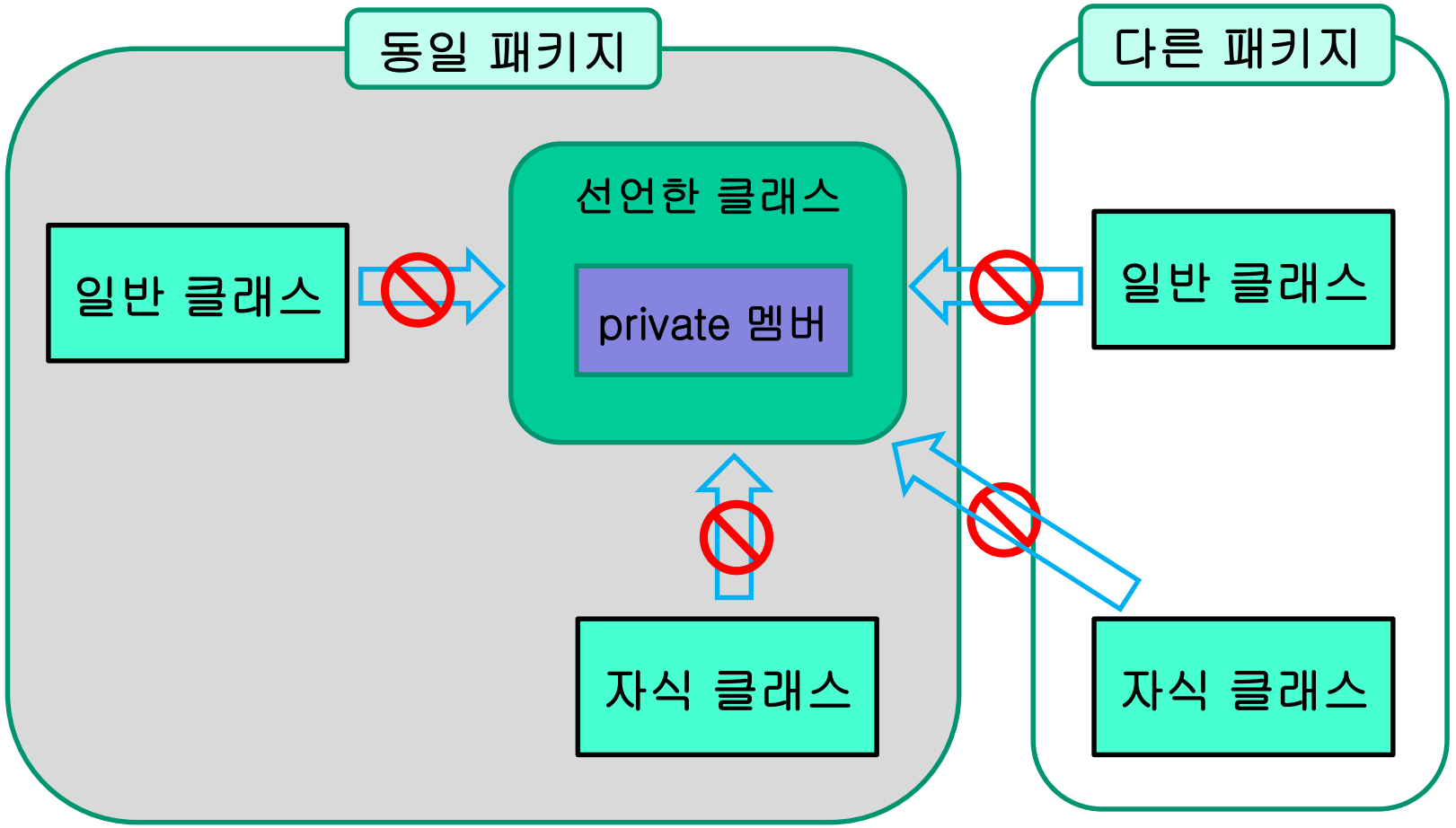
- 차는 엔진을 가지고 있으며, 소방차는 차의 일종이다.



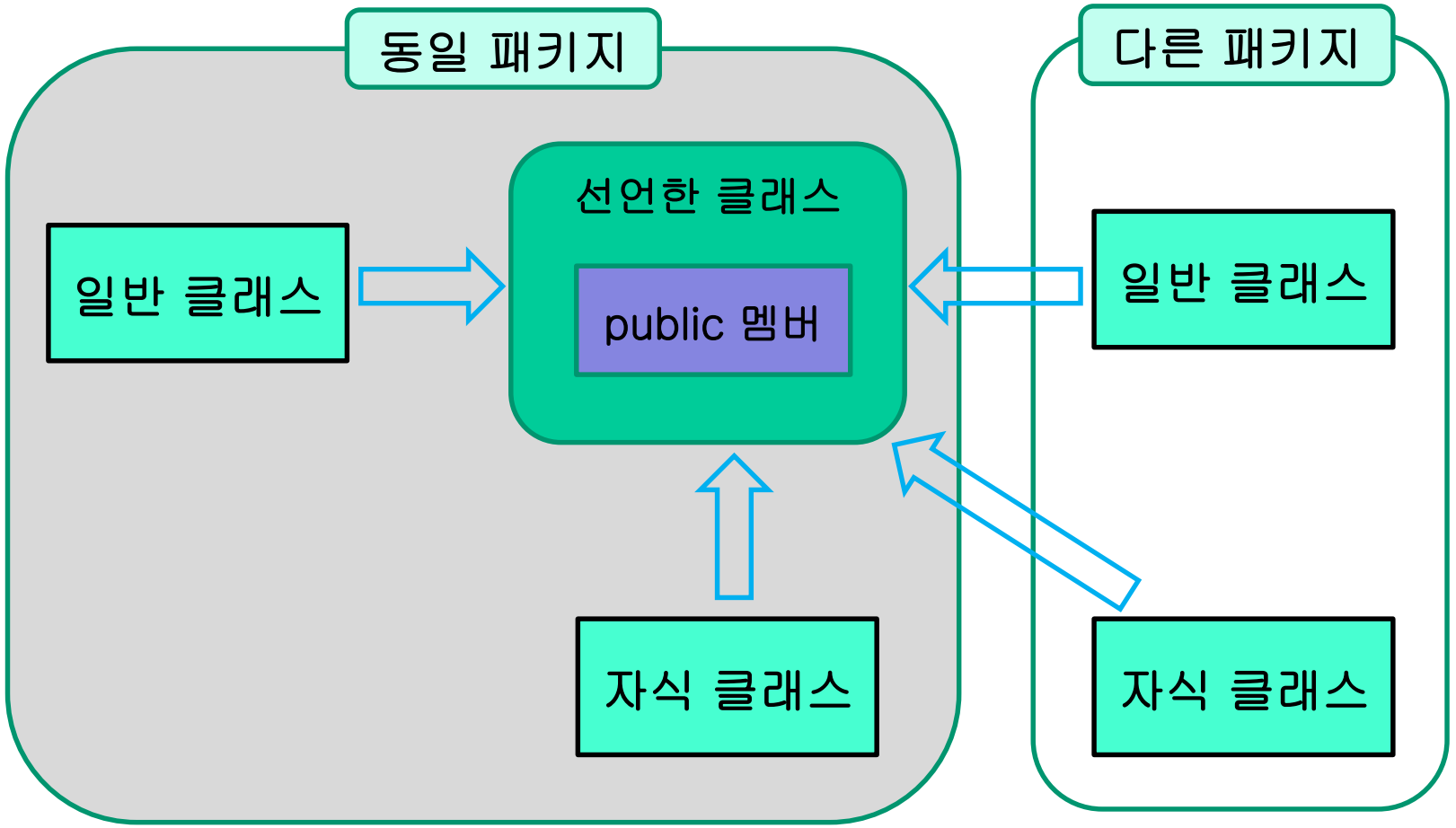
JAVA의 접근자(Modifier)

Modifier	설명
private	같은 클래스 내에서만 접근 가능
default (No modifier)	같은 폴더(패키지)내에서만 가능
protected	같은 폴더(패키지)및 그 클래스를 상속(extends)해서 구현하는 경우 접근이 가능
public	모든 클래스에서 접근이 가능

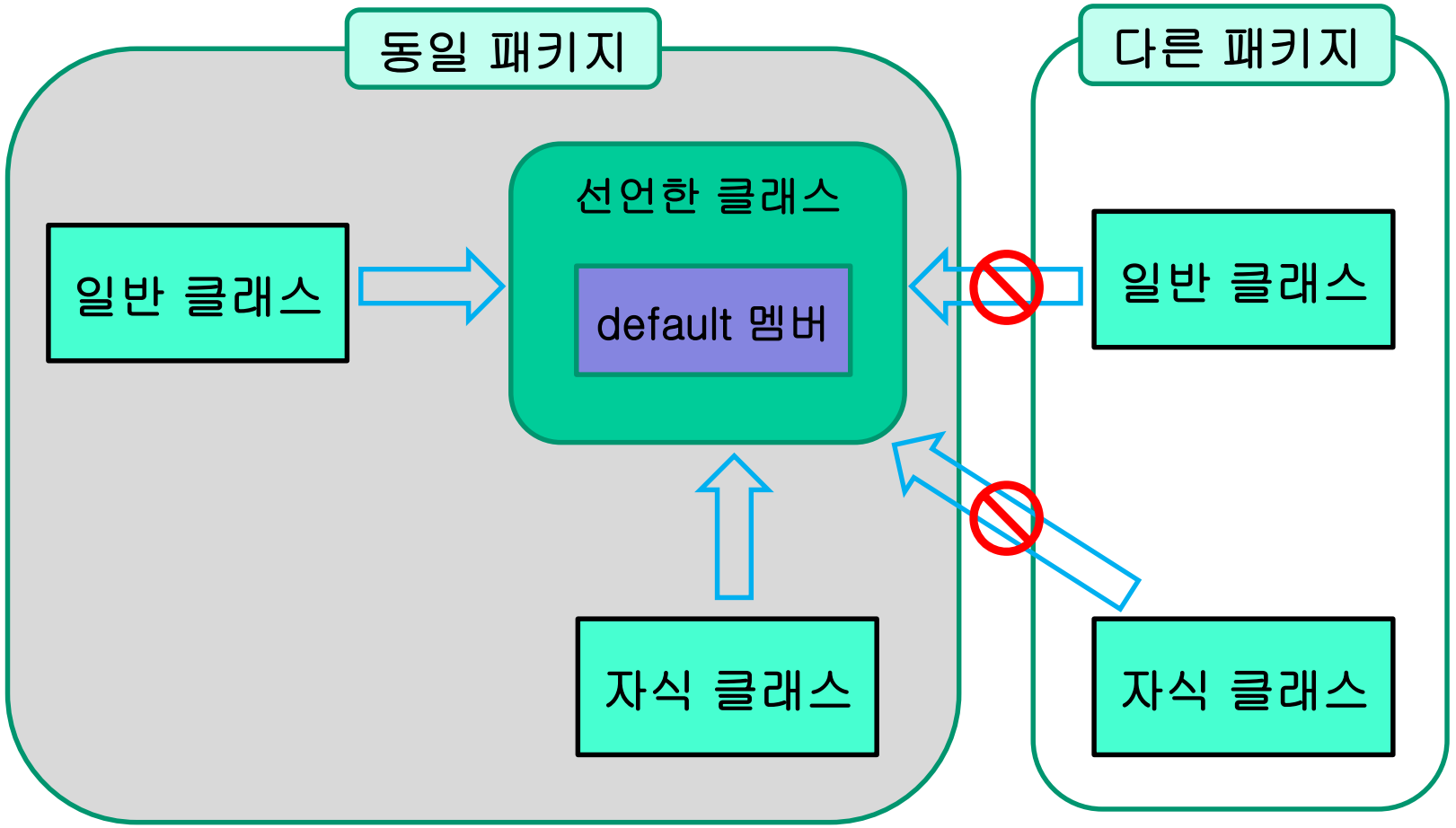
JAVA의 접근자(Modifier)



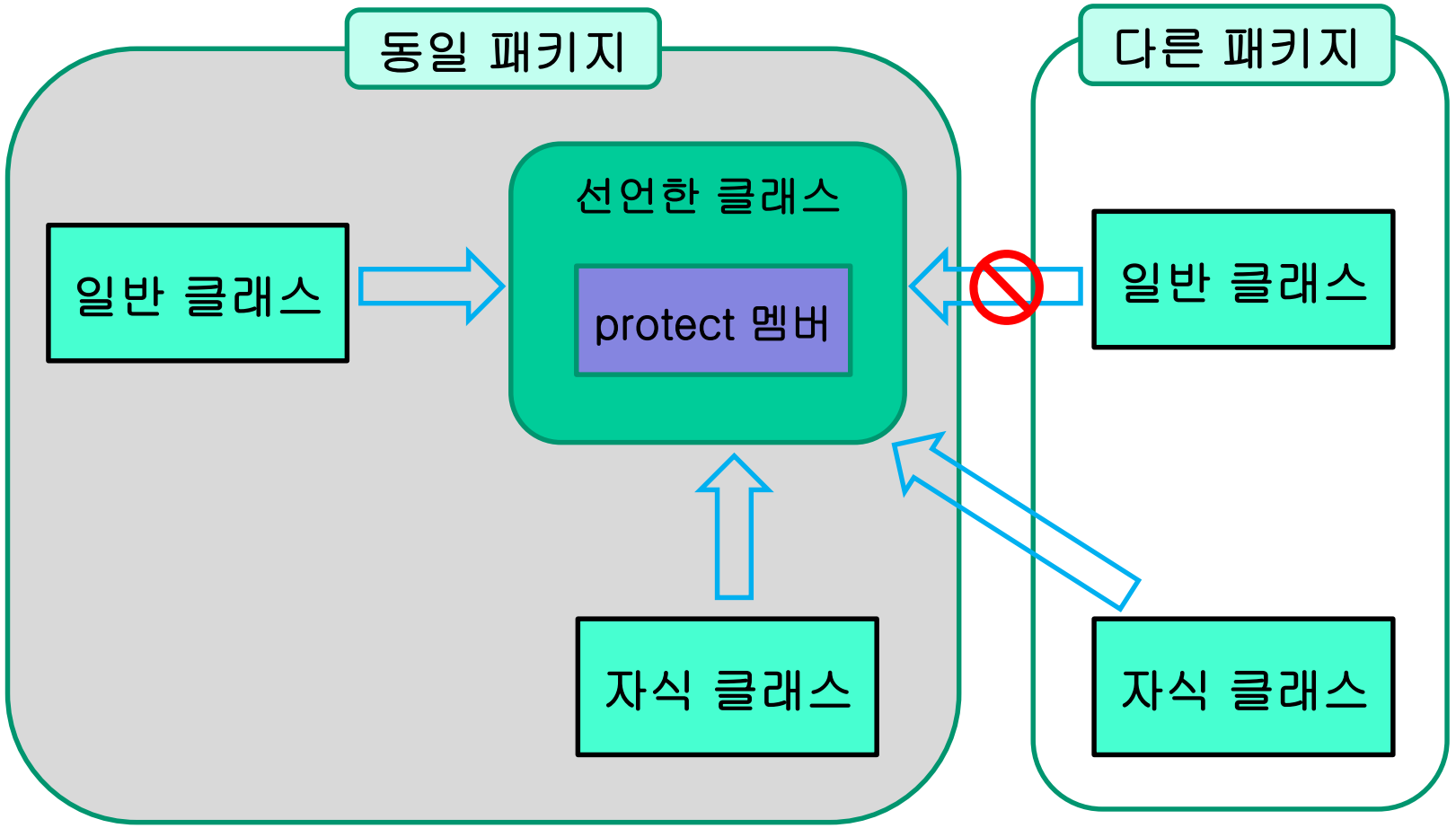
JAVA의 접근자(Modifier)



JAVA의 접근자(Modifier)



JAVA의 접근자(Modifier)



Inheritance

- Inheritance와 생성자
  - Inheritance된 클래스는 자신에게 정의된 모든 요소와 상위 클래스들이 가지고 있는 모든 요소를 가지고 있음
  - Inheritance 관계의 클래스들은 자신의 초기화를 위해 생성자들을 가질 수 있으며, 상속된 하위 클래스에서 객체가 생성될 때에는 자신의 생성자뿐만 아니라 상위 클래스의 생성자들도 수행되어야 함
  - JAVA에서는 상속된 하위 클래스에서 객체가 생성되면 자동적으로 상위 클래스의 생성자가 실행되며 상위 클래스의 생성자가 하위 클래스의 생성자보다 먼저 수행됨
  - 즉 상위 클래스의 생성자가 먼저 수행되어 초기화를 수행한 다음, 하위 클래스의 초기화가 이루어짐

Inheritance

- Inheritance와 생성자
  - 만일 상위 클래스의 생성자가 Overriding되어 여러 개 있을 때 묵시적 생성자가 아닌 다른 생성자의 호출을 원한다면 명시적인 호출 문장을 사용하여야 함
  - 상위 클래스의 생성자를 명시적으로 호출하기 위해 예약어 super를 사용하며 클래스의 특정 생성자를 호출하는 super 문장은 반드시 생성자 첫번째 라인에 위치해야 하며, 이것은 상위 클래스의 생성자가 항상 하위 클래스 생성자보다 먼저 수행되어야 함을 의미 함
  - 상위 클래스의 생성자를 호출하기 위해서는 다음과 같은 형식으로 사용함

```
super(매개변수)
```

Inheritance

- 상속과 생성자

//앞의 예제 프로그램 Super_test2.java도 같이 참조

- [Example] Inheritance_Constructor1.java

```
class A1 {
  double d1;
  A1() {                        // 클래스 A1의 묵시적 생성자
    System.out.println("클래스 A1의 생성자 수행");
    d1 = 10*10;
  }
}

class A2 extends A1 {
  double d2;
  A2() {
    System.out.println("클래스 A2의 생성자 수행");
    d2 = 10*10*10;
  }
}
```

Inheritance

- 상속과 생성자

```
class A3 extends A2 {
  double d3;
  A3() {
    System.out.println("클래스 A3의 생성자 수행");
    d3 = 10*10*10*10;
  }
}

public class Inheritance_Constructor1 {
  public static void main(String args[]) {
    A3 super1 = new A3();
    System.out.println("10의 2제곱 : " + super1.d1);
    System.out.println("10의 3제곱 : " + super1.d2);
    System.out.println("10의 4제곱 : " + super1.d3);
  }
}
```

Inheritance

- 상속과 생성자

실행결과]

```
클래스 A1의 생성자 수행
클래스 A2의 생성자 수행
클래스 A3의 생성자 수행
10의 2제곱 : 100.0
10의 3제곱 : 1000.0
10의 4제곱 : 10000.0
```

Inheritance

- 상속과 생성자

[Example] Inheritance_Constructor2.java
//이 프로그램은 생성자를 명시적으로 호출하는 예제 프로그램 , 만약 명시적으로 생성자를 호출하지 않으면 기본 생성자(아무 일도 하지 않는)가 호출

```
 class A1
{
    int d1;
    int s;
    A1(int s1)
    {
      System.out.println("클래스 A1의 생성자 수행");
      s = s1;
      d1 = s * s ;
    }
}
```

Inheritance

상속과 생성자

```
class A2 extends A1
{
    int d2;
    int t;
    A2(int s1, int t1)
    {
      super(s1);  // 상위 클래스의 생성자를 명시적으로 호출
      System.out.println("클래스 A2의 생성자 수행");
      t = t1;
      d2 = t * t ;
    }
}
```

Inheritance

상속과 생성자

```
public class Inheritance_Constructor2
{
    public static void main(String args[])
    {
      A2 super2 = new A2(10,20);
      System.out.println("10의 제곱은 : " + super2.d1);
      System.out.println("20의 제곱은 : " + super2.d2);
    }
}
```

Inheritance

상속과 생성자

[실행 결과]

클래스 A1의 생성자 수행
클래스 A2의 생성자 수행
10의 제곱은 : 100
20의 제곱은 : 400

Inheritance

상속과 생성자

[주의할 점]

상속 관계에 있어서 상위 클래스(아버지 클래스)의 생성자는 상속되지 않기 때문에 상위클래스를 상속받는 하위 클래스는 반드시 super(매개변수) 예약어를 사용하여 명시적으로 상위 클래스의 생성자를 호출 해 주거나

다른 방법으로는 상위클래스에 명시적으로 묵시적인 생성자, 예를 들어 생성자(){} 형태의 묵시적인 생성자를 명시하여야 함 , 즉 이때에는 상속받는 sub 클래스에서 super 예약어를 사용하지 않았더라도 컴파일러가 자동으로 super()라는 상위클래스의 묵시적인 생성자를 호출하는 코드를 삽입하므로 error 가 발생하지 않음