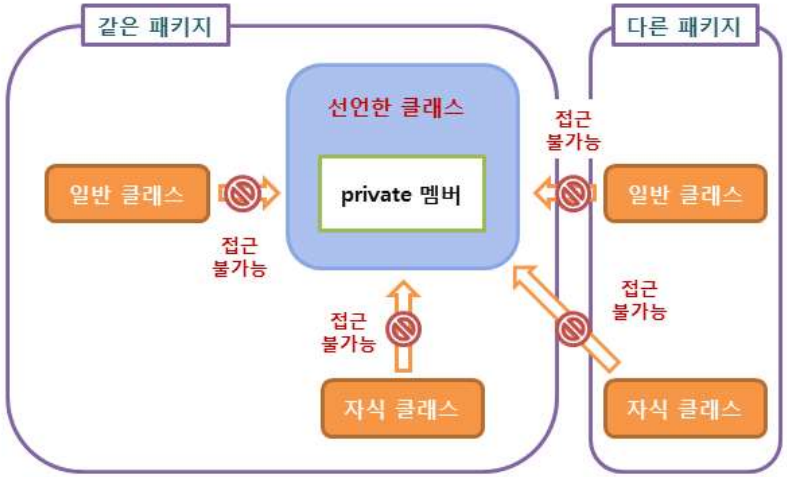
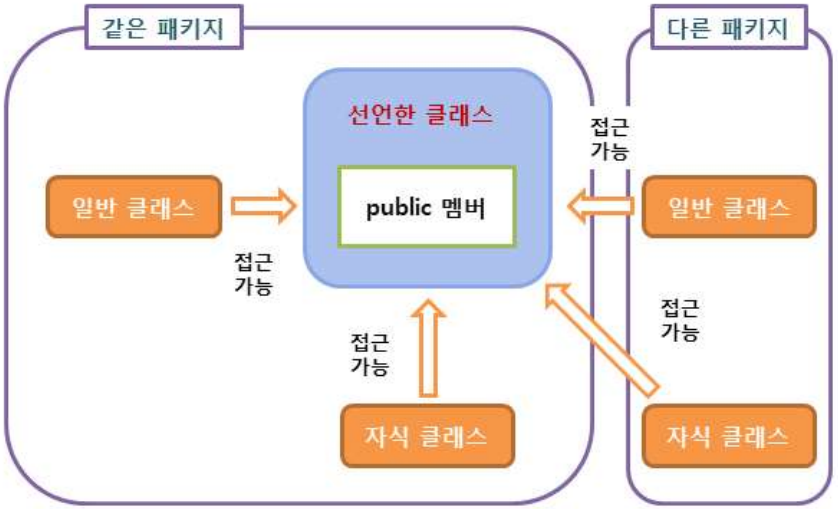
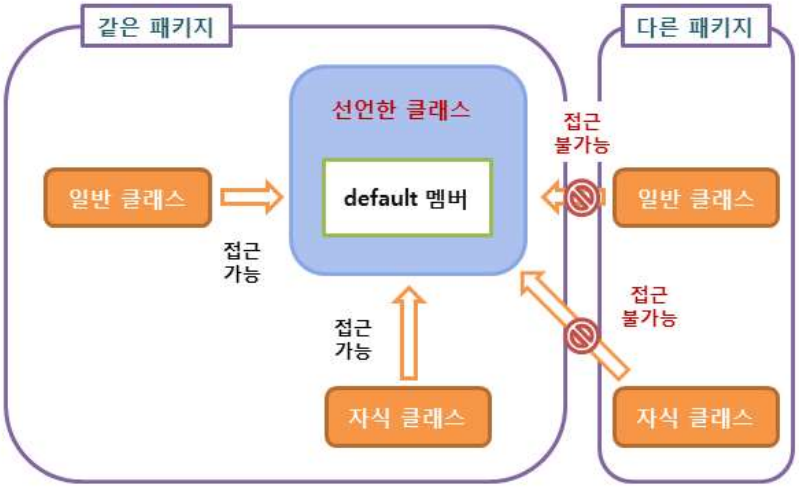
Java 다형성

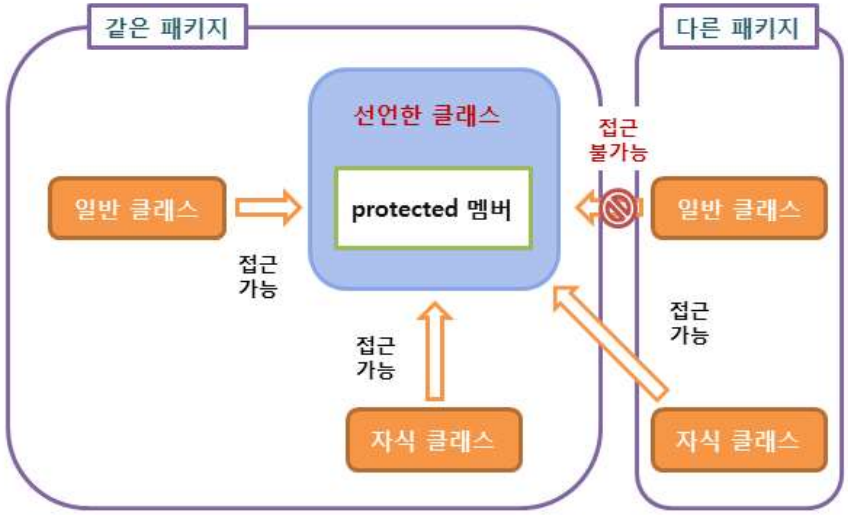
2025 년

접근 제어자


같은 패키지
다른 패키지
선언한 클래스
private 멤버
일반 클래스
자식 클래스
접근 불가능

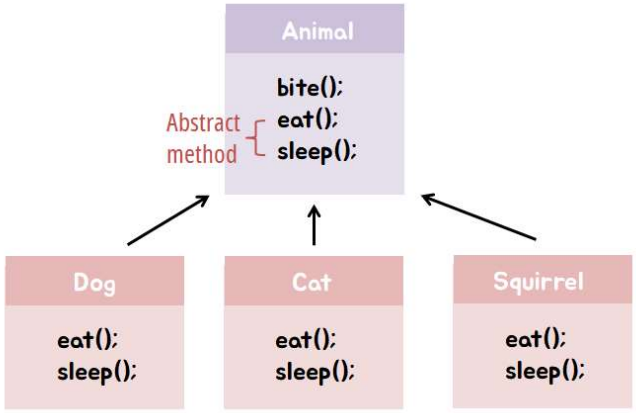

같은 패키지
다른 패키지
선언한 클래스
public 멤버
일반 클래스
자식 클래스
접근 가능

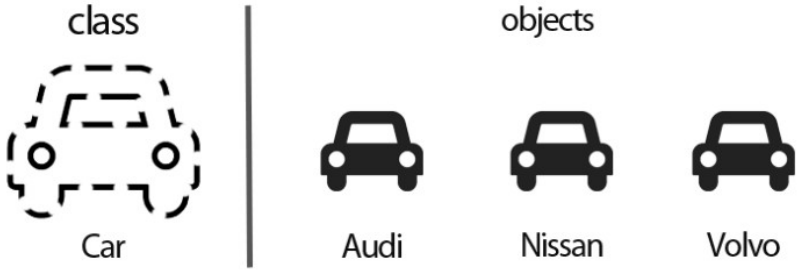

같은 패키지
다른 패키지
선언한 클래스
default 멤버
일반 클래스
자식 클래스
접근 가능
접근 불가능


같은 패키지
다른 패키지
선언한 클래스
protected 멤버
일반 클래스
자식 클래스
접근 가능
접근 불가능

타입변환

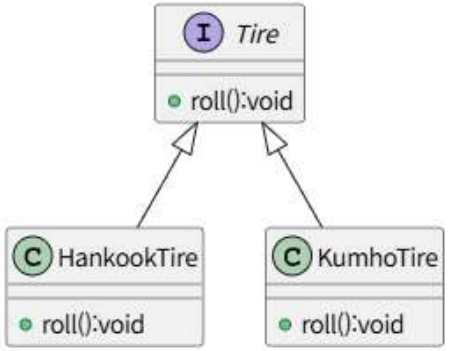
- **참조 변수의 타입 변환**
  - 서로 상속 관계에 있는 클래스 사이에만 타입 변환을 할 수 있습니다.
  - 자식 클래스 타입에서 부모 클래스 타입으로의 타입 변환은 생략할 수 있습니다.
  - 하지만 부모 클래스 타입에서 자식 클래스 타입으로의 타입 변환은 반드시 명시해야 합니다.


Animal
bite();
eat();
sleep();
Abstract method
Dog
eat();
sleep();
Cat
eat();
sleep();
Squirrel
eat();
sleep();


class
objects
Car
Audi
Nissan
Volvo

비교	추상클래스	인터페이스
공통점	1. 추상클래스(인터페이스도 일종의 추상클래스) 2. 하위 클래스에서 모든 추상메소드를 구현 해야 한다. 3. 그 자체가 객체로 인스턴스화 될 수 없다. (new 해서 객체 생성 불가) → extends 또는 implements 한 자식클래스만 객체로 생성할 수 있다.	
차이점	1. 추상메서드 외 일반 멤버 변수와 메소드를 가질 수 있다 2. extends 키워드 사용 3. 단일 상속만 가능 4. 작업의 레벨 분할을 위해서 사용 * 공통 메소드가 있는 경우 추상클래스가 적합	1. 추상메소드와 상수만 사용 2. 인터페이스는 모든 메소드가 추상메소드 3. implements 키워드 사용 4. 다중 상속 가능 5. 공동 작업에 용이하다

```java
public interface Tire {
    public void roll();
}
```




```java
public class Car {
    Tire frontLeft = new HankookTire();
    Tire frontRight= new HankookTire();
    Tire backLeft= new HankookTire();
    Tire backRight= new HankookTire();

    void run(){
        frontLeft.roll();
        frontRight.roll();
        backLeft.roll();
        backRight.roll();
    }
}
```

```java
public class HankookTire implements Tire{

    @Override
    public void roll(){
        System.out.println(x:"한국타이어가 굴러 갑니다.");
    }
}
```

```java
public class KumhoTire implements Tire{

    @Override
    public void roll(){
        System.out.println(x:"금호타이어가 굴러 갑니다.");
    }
}
```

```java
public class CarExam {
    Run | Debug
    public static void main(String[] args) {
        Car myCar = new Car();

        myCar.run();

        myCar.frontLeft = new KumhoTire();
        myCar.frontRight = new KumhoTire();
        myCar.run();
    }
}
```

```java
public interface Vehicle {
    public void run();
}
```

```java
public class Bus implements Vehicle{
    public void run(){
        System.out.println(x:"버스가 달립니다.");
    }
}
```

```java
public class Taxi implements Vehicle{
    public void run(){
        System.out.println(x:"버스가 달립니다.");
    }
}
```

```java
public class Driver {
    public void drive(Vehicle vehicle) {
        vehicle.run();
    }
}
```

```java
public class DriveExam {
    Run | Debug
    public static void main(String[] args) {
        Driver driver = new Driver();

        Bus bus = new Bus();
        Taxi taxi = new Taxi();

        driver.drive(bus);
        driver.drive(taxi);
    }
}
```

```java
public interface Vehicle {
    public void run();
}
```

```java
public class Bus implements Vehicle{
    public void run(){
        System.out.println(x:"버스가 달립니다.");
    }
    public void checkFare() {
        System.out.println(x:"승차요금을 체크합니다.");
    }
}
```

```java
public class Taxi implements Vehicle{
    public void run(){
        System.out.println(x:"버스가 달립니다.");
    }
}
```

```java
public class Driver {
    public void drive(Vehicle vehicle) {
            vehicle.run();
    }
}
```

```java
public class CastExam {
    Run | Debug
    public static void main(String[] args) {
        Vehicle vehicle = new Bus();
        vehicle.run();
        // vehicle.checkFare();

        Bus bus1 = (Bus)vehicle;
        // 가능....함.
        bus1.checkFare();
        if ( vehicle instanceof Bus)
            System.out.println(x:"버스 객체다...");
        if ( bus1 instanceof Bus)
            System.out.println(x:"버스 객체다...");
    }
}

```

```java
public interface Vehicle {
    public void run();
}
```

```java
public class Bus implements Vehicle{
    public void run(){
        System.out.println(x:"버스가 달립니다.");
    }
    public void checkFare() {
        System.out.println(x:"승차요금을 체크합니다.");
    }
}
```

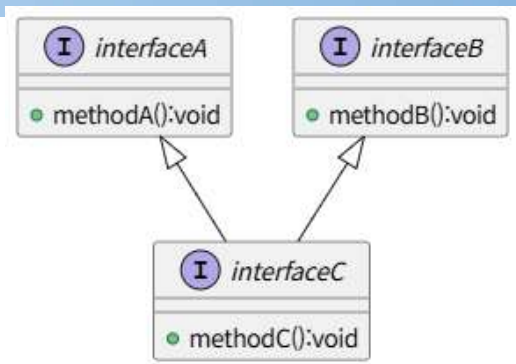
```java
public class Taxi implements Vehicle{
    public void run(){
        System.out.println(x:"버스가 달립니다.");
    }
}
```

```java
public class Driver {
    public void drive(Vehicle vehicle) {
        if(vehicle instanceof Bus) {
            Bus bus = (Bus) vehicle;
            bus.checkFare();
        }
        vehicle.run();
    }
}
```

```java
public class DriveExam {
    Run | Debug
    public static void main(String[] args) {
        Driver driver = new Driver();

        Bus bus = new Bus();
        Taxi taxi = new Taxi();

        driver.drive(bus);
        driver.drive(taxi);
    }
}
```

```java
public interface InterfaceA {
    public void methodA();
}

```

```java
public interface InterfaceB {
    public void methodB();
}

```




```java
public interface InterfaceC extends InterfaceA, InterfaceB{
    public void methodC();
}
```

```java
public class ImplementationC implements InterfaceC{
    public void methodA(){

    }
    public void methodB(){

    }
    public void methodC(){

    }

}
```