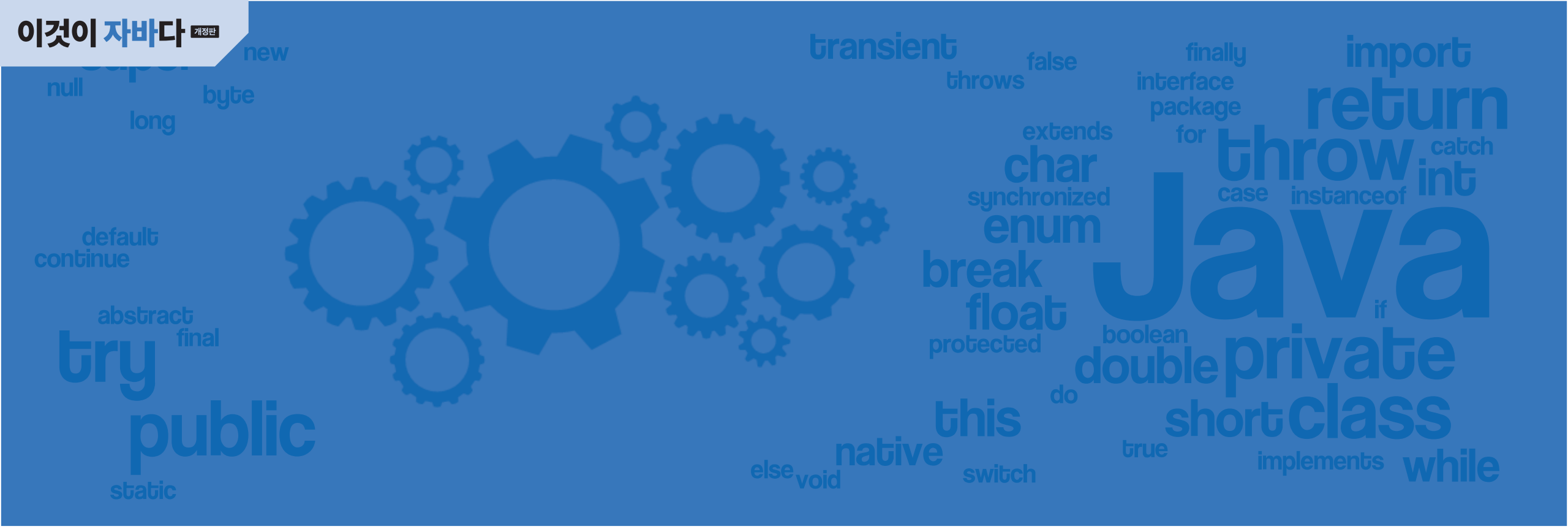
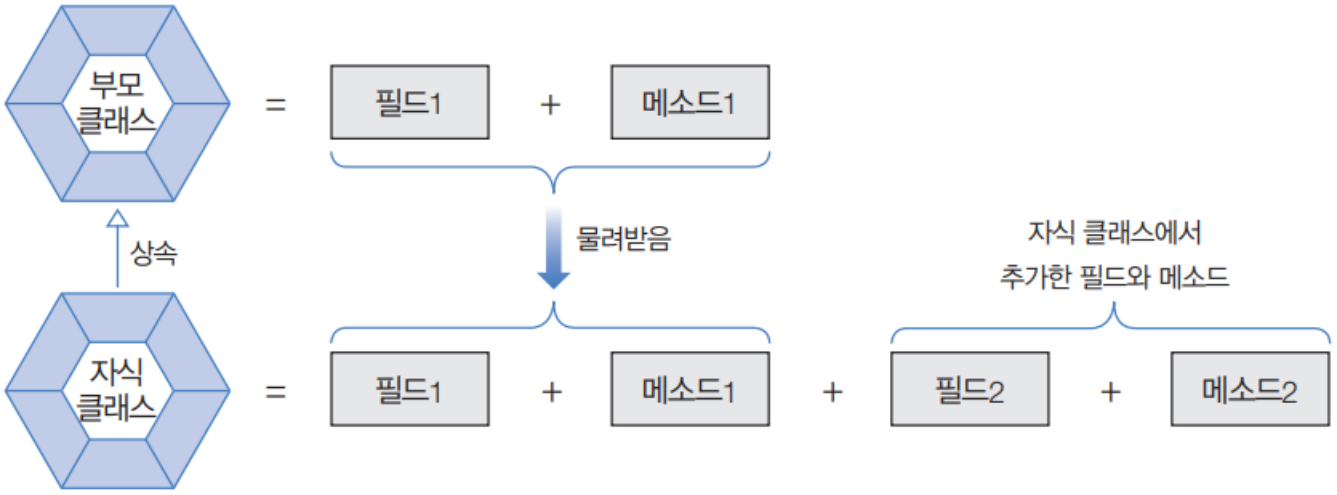


Chapter 07 상속

Contents

상속

- 부모 클래스의 필드와 메소드를 자식 클래스에게 물려줄 수 있음



상속의 이점

- 이미 개발된 클래스를 재사용하므로 중복 코드를 줄임
- 클래스 수정을 최소화

클래스 상속

- 자식 클래스를 선언할 때 어떤 부모로부터 상속받을 것인지를 결정하고, 부모 클래스를 다음과 같이 extends 뒤에 기술

```
public class 자식클래스 extends 부모클래스 {
}
```

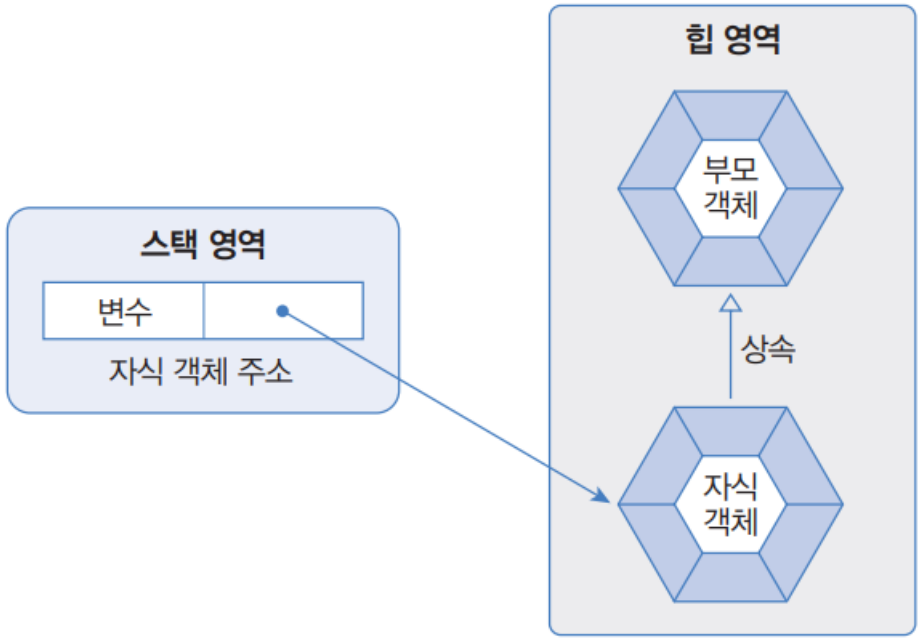
- 다중 상속 허용하지 않음. extends 뒤에 하나의 부모 클래스만 상속

```
public class 자식클래스 extends 부모클래스1, 부모클래스2 {
}
```

부모 생성자 호출

- 자식 객체를 생성하면 부모 객체가 먼저 생성된 다음에 자식 객체가 생성

```
자식클래스 변수 = new 자식클래스( );
```

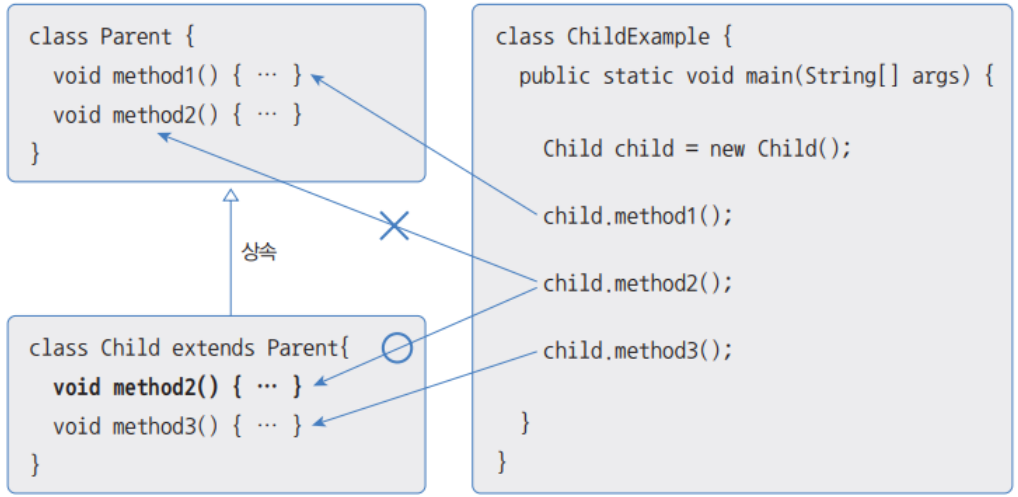


- 부모 생성자는 자식 생성자의 맨 첫 줄에 숨겨져 있는 super()에 의해 호출

```
//자식 생성자 선언
public 자식클래스(…) {
  super();
  …
}
```

메소드 오버라이딩

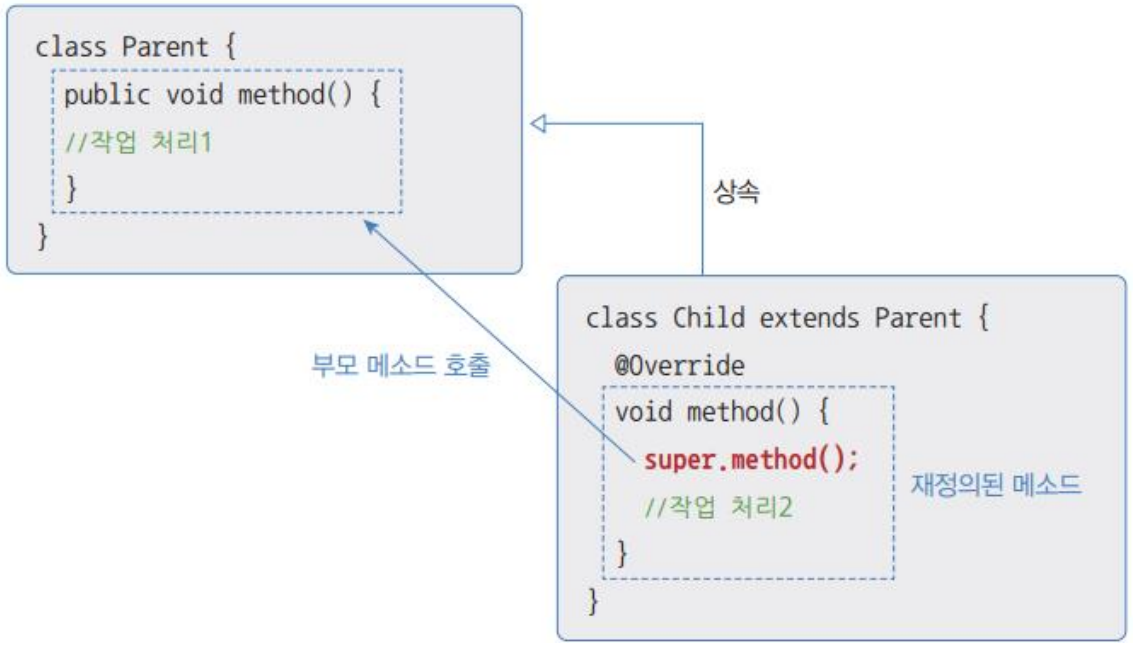
- 상속된 메소드를 자식 클래스에서 재정의하는 것. 해당 부모 메소드는 숨겨지고, 자식 메소드가 우선적으로 사용



- 부모 메소드의 선언부(리턴 타입, 메소드 이름, 매개변수)와 동일해야 함
- 접근 제한을 더 강하게 오버라이딩할 수 없음(public → private으로 변경 불가)
- 새로운 예외를 throws할 수 없음

부모 메소드 호출

- 자식 메소드 내에서 super 키워드와 도트(.) 연산자를 사용하면 숨겨진 부모 메소드를 호출
- 부모 메소드를 재사용함으로써 자식 메소드의 중복 작업 내용을 없애는 효과



final 클래스

- final 클래스는 부모 클래스가 될 수 없어 자식 클래스를 만들 수 없음

```
public final class 클래스 { … }
```

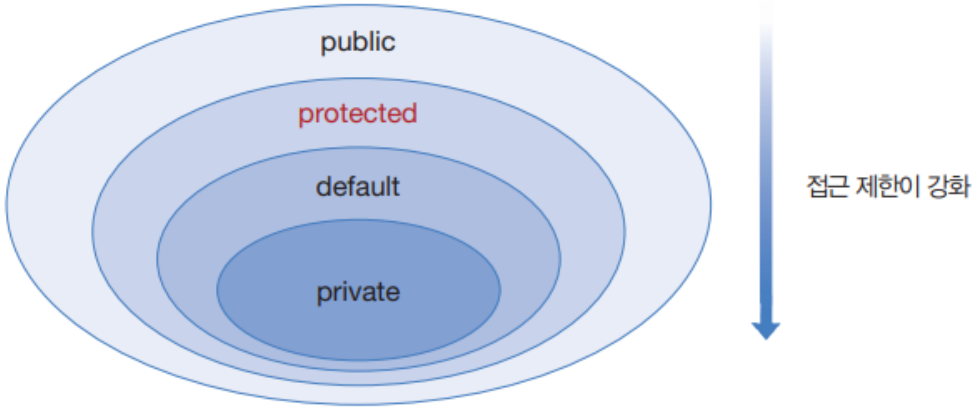
final 메소드

- 메소드를 선언할 때 final 키워드를 붙이면 오버라이딩할 수 없음
- 부모 클래스를 상속해서 자식 클래스를 선언할 때, 부모 클래스에 선언된 final 메소드는 자식 클래스에서 재정의할 수 없음

```
public final 리턴타입 메소드( 매개변수, … ) { … }
```

protected 접근 제한자

- protected는 상속과 관련이 있고, public과 default의 중간쯤에 해당하는 접근 제한
- protected는 같은 패키지에서는 default처럼 접근이 가능하나, 다른 패키지에서는 자식 클래스만 접근을 허용

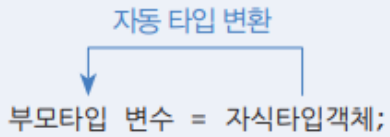


NOTE ▸ default는 접근 제한자가 아니라 접근 제한자가 붙지 않은 상태를 말한다.

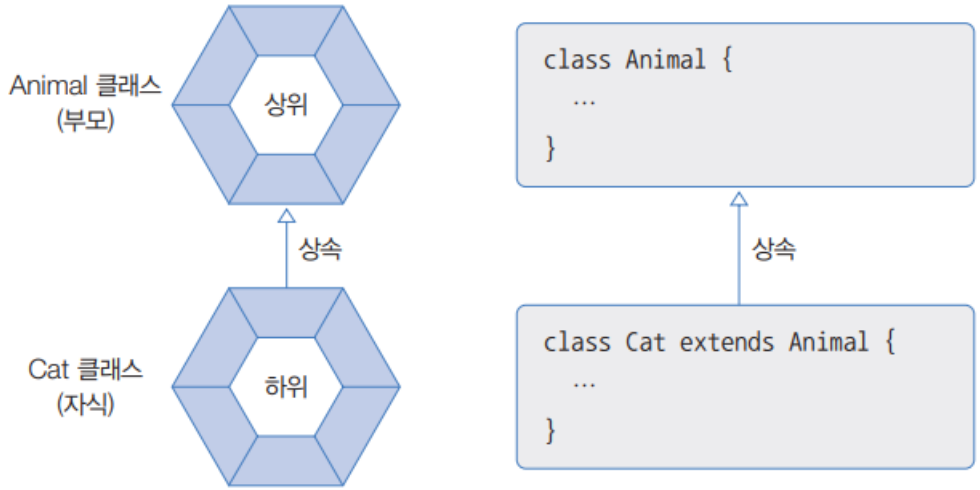
접근 제한자	제한 대상	제한 범위
protected	필드, 생성자, 메소드	같은 패키지이거나, 자식 객체만 사용 가능

자동 타입 변환

- 자동적으로 타입 변환이 일어나는 것

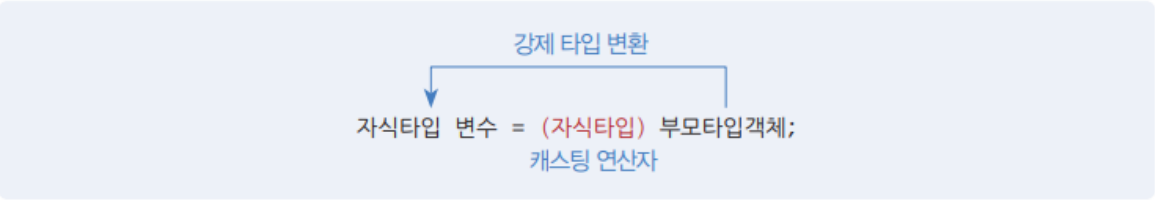


- 자식은 부모의 특징과 기능을 상속받기 때문에 부모와 동일하게 취급

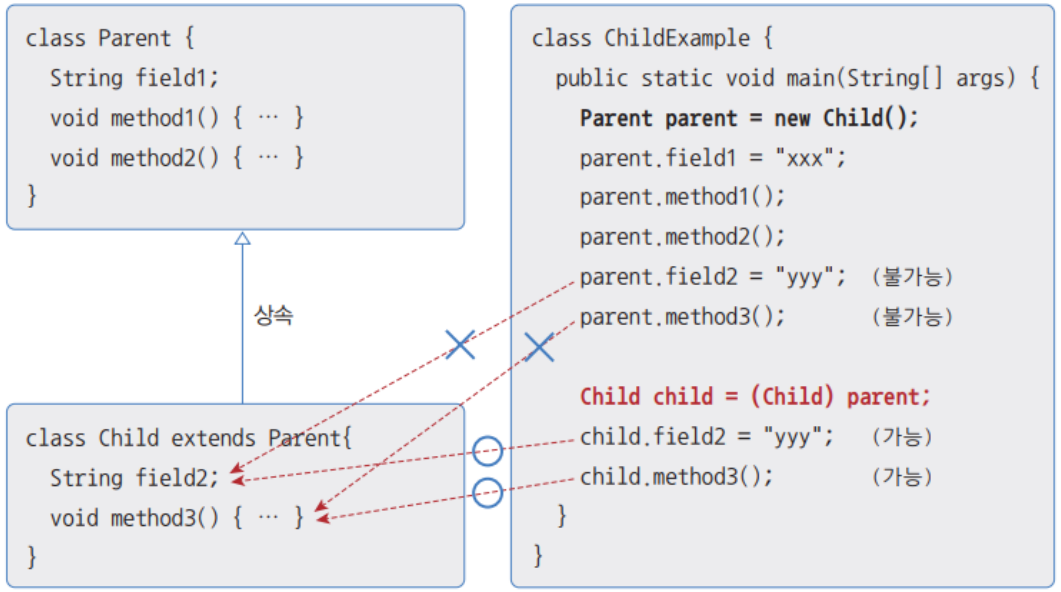


강제 타입 변환

- 부모 타입은 자식 타입으로 자동 변환되지 않음. 대신 캐스팅 연산자로 강제 타입 변환 가능

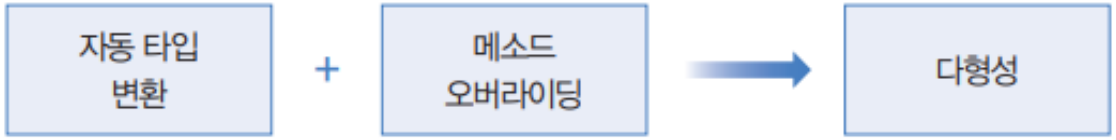


- 자식 객체가 부모 타입으로 자동 변환하면 부모 타입에 선언된 필드와 메소드만 사용 가능



다형성

- 사용 방법은 동일하지만 실행 결과가 다양하게 나오는 성질
- 다형성을 구현하기 위해서는 자동 타입 변환과 메소드 재정의가 필요



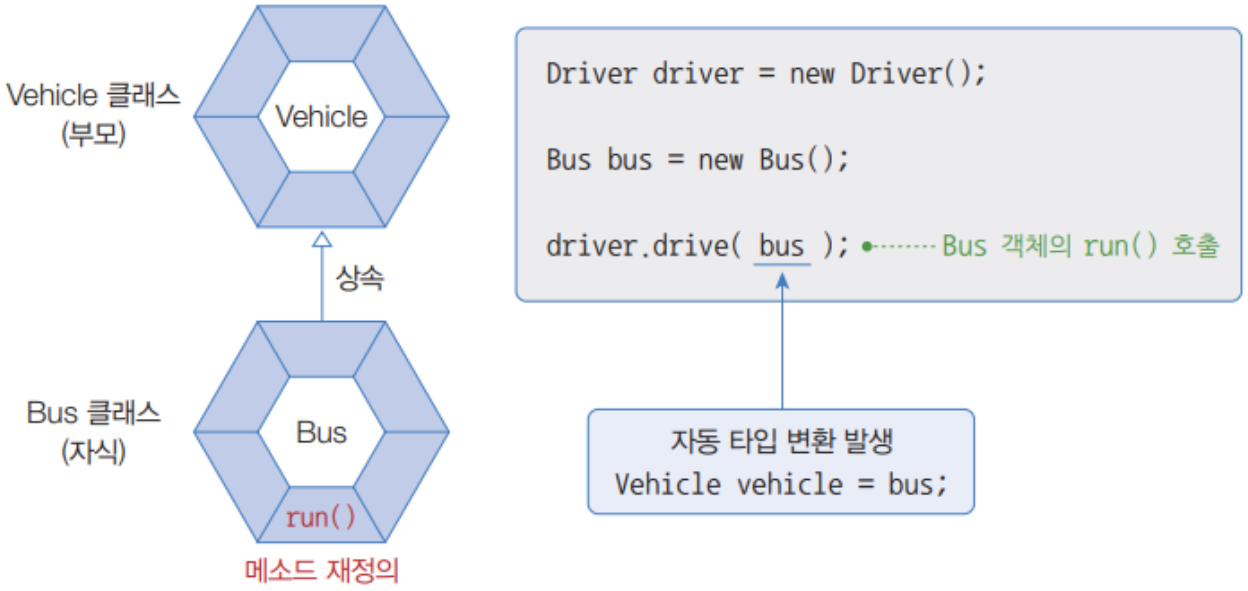
필드 다형성

- 필드 타입은 동일하지만, 대입되는 객체가 달라져서 실행 결과가 다양하게 나올 수 있는 것

```
//Car 객체 생성
Car myCar = new Car();
//HankookTire 장착
myCar.tire = new HankookTire();
//KumhoTire 장착
myCar.tire = new KumhoTire();
```

매개변수 다형성

- 메소드가 클래스 타입의 매개변수를 가지고 있을 경우, 호출할 때 동일한 타입의 자식 객체를 제공할 수 있음
- 어떤 자식 객체가 제공되느냐에 따라서 메소드의 실행 결과가 달라짐



instanceof 연산자

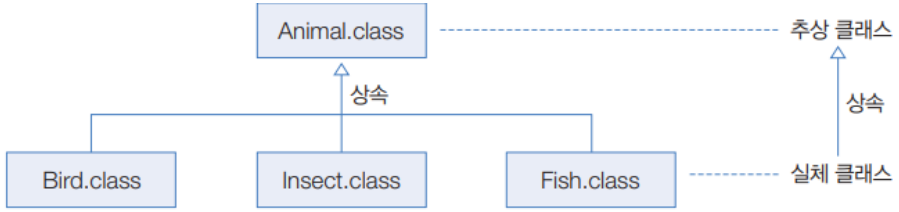
- 매개변수가 아니더라도 변수가 참조하는 객체의 타입을 확인할 때 instanceof 연산자를 사용
- instanceof 연산자에서 좌항의 객체가 우항의 타입이면 true를 산출하고 그렇지 않으면 false를 산출

```
boolean result = 객체 instanceof 타입;
```

- Java 12부터는 instanceof 연산의 결과가 true일 경우 우측 타입 변수를 사용할 수 있기 때문에 강제 타입 변환이 필요 없음

추상 클래스

- 객체를 생성할 수 있는 실체 클래스들의 공통적인 필드나 메소드를 추출해서 선언한 클래스
- 추상 클래스는 실체 클래스의 부모 역할. 공통적인 필드나 메소드를 물려받을 수 있음



추상 클래스 선언

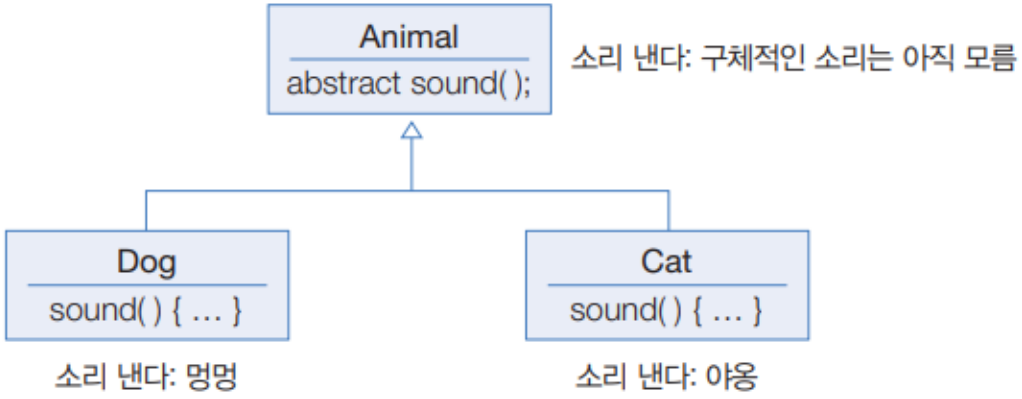
- 클래스 선언에 abstract 키워드를 붙임
- new 연산자를 이용해서 객체를 직접 만들지 못하고 상속을 통해 자식 클래스만 만들 수 있다.

```
public abstract class 클래스명 {
  //필드
  //생성자
  //메소드
}
```

추상 메소드와 재정의

- 자식 클래스들이 가지고 있는 공통 메소드를 뽑아내어 추상 클래스로 작성할 때, 메소드 선언부만 동일하고 실행 내용은 자식 클래스마다 달라야 하는 경우 추상 메소드를 선언할 수 있음
- 일반 메소드 선언과의 차이점은 abstract 키워드가 붙고, 메소드 실행 내용인 중괄호 { }가 없다.

```
abstract 리턴타입 메소드명(매개변수, …);
```



sealed 클래스

- Java 15부터 무분별한 자식 클래스 생성을 방지하기 위해 봉인된 클래스가 도입
- sealed 키워드를 사용하면 permits 키워드 뒤에 상속 가능한 자식 클래스를 지정
- final은 더 이상 상속할 수 없다는 뜻이고, non-sealed는 봉인을 해제한다는 뜻

```
public sealed class Person  permits Employee, Manager { ··· }
```

```
public final class Employee extends Person { ··· }
public non-sealed class Manager extends Person { ··· }
```


transient
false
finally
import
new
byte
null
long
throws
interface
return
package
extends
for
throw
catch
char
int
synchronized
case
instanceof
enum
default
continue
abstract
final
try
boolean
double
private
public
do
this
short
class
native
true
implements
while
else
void
switch
static

Thank you!