객체지향 프로그래밍

Object-Oriented Programming

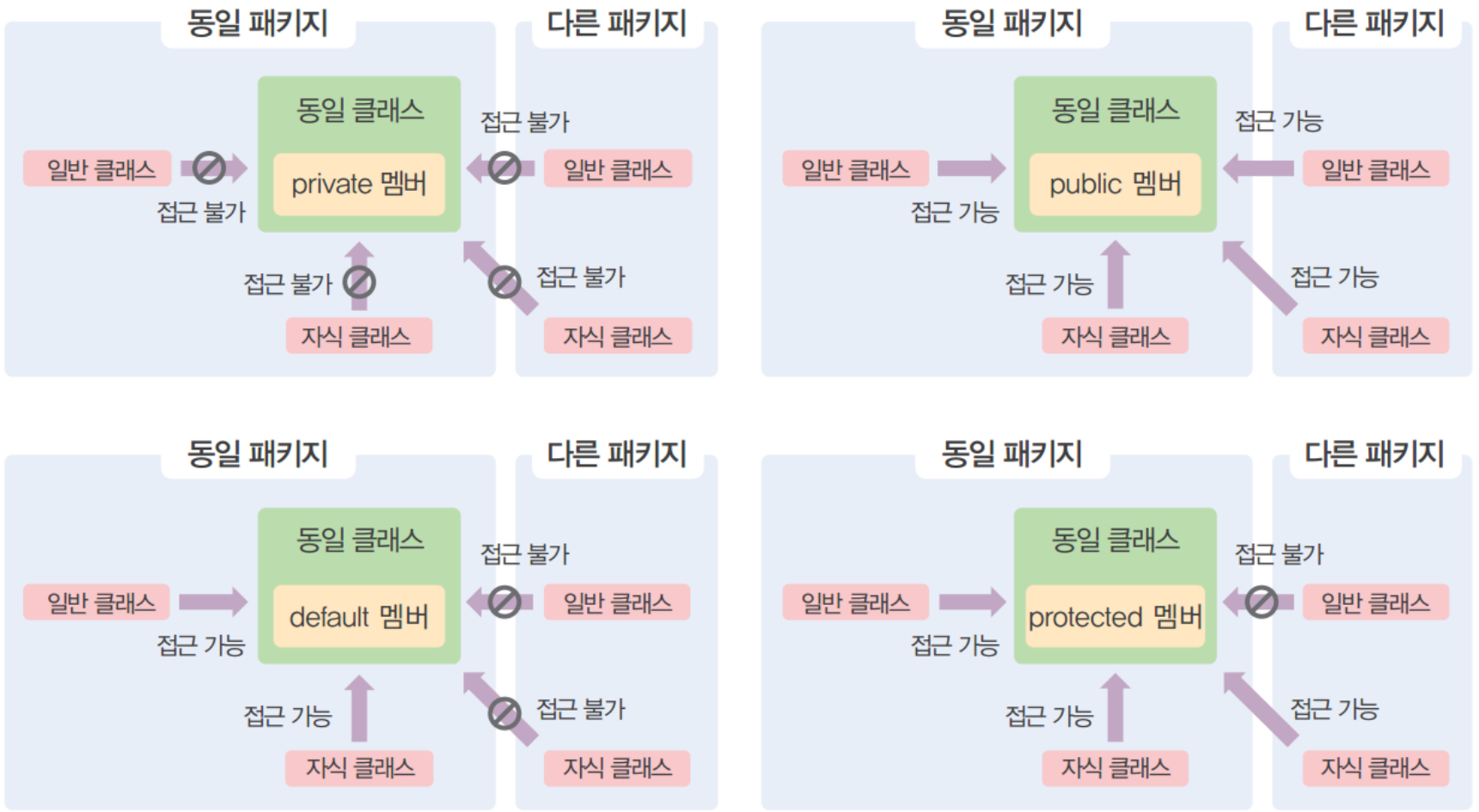
본 자료는 한빛아카데미에서 제공한 강의자료를 기반으로 재구성하였습니다.

동국대학교 dongguk university

Chapter 06.
상속

상속과 접근 제어

접근 지정자의 접근 범위



상속과 접근 제어

접근 지정자의 접근 범위

접근 지정자	동일 클래스	동일 패키지	자식 클래스	다른 패키지
public	O	O	O	O
protected	O	O	O	X
없음	O	O	X	X
private	O	X	X	X

상속과 접근 제어

접근 지정자 사용 시 주의 사항

- private 멤버는 자식 클래스에 상속되지 않는다.
- 클래스 멤버는 어떤 접근 지정자로도 지정할 수 있지만, 클래스는 protected와 private으로 지정할 수 없다.
- 메서드를 오버라이딩할 때 부모 클래스의 메서드보다 가시성을 더 좁게 할 수는 없다

final 클래스와 메서드

final 클래스

- 더 이상 상속될 수 없는 클래스
- 대표적인 final 클래스로는 String 클래스

final 메서드

- 자식 클래스에서의 오버라이딩을 금지

class ChildString **extends** String {…} ➔ X

final 클래스와 메서드

Practice

- 예제 6-14 ~ 6-15: final 클래스, 메서드

```java
package chap06;

class Good {
}

class Better extends Good {
}

final class Best extends Better {}
// class NumberOne extends Best {}

public class FinalClassDemo {
    public static void main(String[] args) {
        // new NumberOne();
        new Best();
    }
}
```

```java
package chap06;

class Chess {
    enum ChessPlayer {
        WHITE, BLACK
    }

    final ChessPlayer getFirstPlayer() {
        return ChessPlayer.WHITE;
    }
}

class WorldChess extends Chess {
    // ChessPlayer getFirstPlayer() {}
}

public class FinalMethodDemo {
    public static void main(String[] args) {
        WorldChess w = new WorldChess();
        w.getFirstPlayer();
    }
}
```

타입 변환과 다형성

객체의 타입 변환

- 참조 타입 데이터도 기초 타입 데이터처럼 타입 변환 가능.
- 단, 상속 관계일 경우만 타입 변환 가능.
- 기초 타입처럼 자동 타입 변환과 강제 타입 변환이 있다.

타입 변환과 다형성

자동 타입 변환

[예제 6-16] 객체 타입 변환을 하는 Person 클래스

```
01  public class Person {
02      String name = "사람";
03
04      void whoami() {
05          System.out.println("나는 사람이다.");
06      }
07  }
```

[예제 6-17] 객체 타입 변환을 하는 Student 클래스

```
01  public class Student extends Person {
02      int number = 7;
03
04      void work() {
05          System.out.println("나는 공부한다.");
06      }
07  }
```

```
Student s = new Student();
Person p = s;        // 자동으로 타입 변환을 한다.
```

타입 변환과 다형성

자동 타입 변환

```
Person p;
Student s = new Student();
```

```
p = s;
```

자동으로 타입 변환되며 p = (Person)s와 동일하다.

```
// p.number = 1;
// p.work();
```

number와 work( )는 부모 타입에 없는 멤버이므로 부모 타입 변수에서 볼 수 없다.

```
p.whoami();
```

Person 타입 멤버이므로 호출할 수 있다.

타입 변환과 다형성

강제 타입 변환

- 자동 타입 변환은 **자식 → 부모** 방향의 타입 변환
- **부모 → 자식** 방향의 강제 타입 변환이 가능

```
Person  p = new Person();
Student s = (Student) p;      // 강제로 타입 변환을 하면 오류가 발생한다.
```

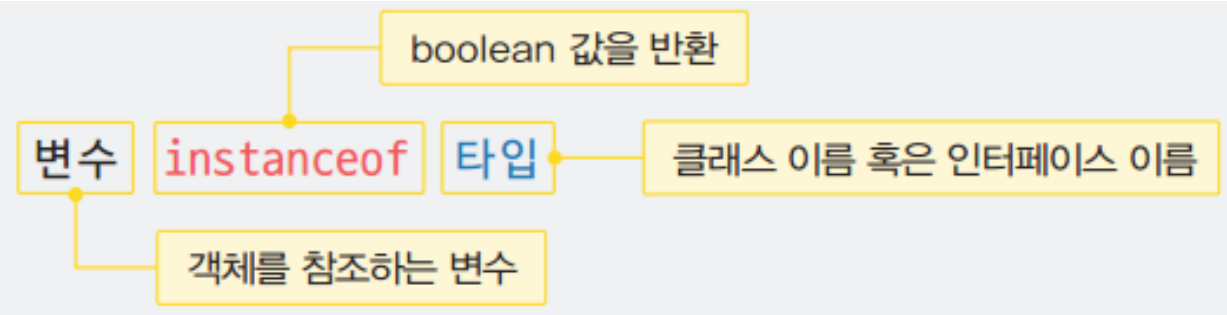
```
Student s1 = new Student();
Person  p  = s1;
Student s2 = (Student) p;     // 강제로 타입 변환을 할 수 있다.
```

부모 타입 변수이지만 자식 객체를 가리킨다.

타입 변환과 다형성

instanceof 연산자

- 타입 변환된 객체의 구별



타입 변환과 다형성

Practice

- 예제 6-16 ~ 6-19: 객체의 타입 변환

```java
package chap06;

public class Student extends Person {
    int number = 7;

    void work() {
        System.out.println("나는 공부한다.");
    }
}
```

```java
package chap06;

public class Person {
    String name = "사람";

    void whoami() {
        System.out.println("나는 사람이다.");
    }
}
```

```java
package chap06;

public class UpcastDemo {
    public static void main(String[] args) {
        Person p;
        Student s = new Student();

        p = s;

        // p.number = 1;
        // p.work();

        p.whoami();
    }
}
```

타입 변환과 다형성

Practice

- 예제 6-19 instanceof 연산자 활용

```java
package chap06;

public class InstanceofDemo {
    public static void main(String[] args) {
        Student s = new Student();
        Person p = new Person();

        System.out.println(s instanceof Person);   T

        System.out.println(s instanceof Student); T

        System.out.println(p instanceof Student); F

        // System.out.println(s instanceof String);오류

        downcast(s);
    }

    static void downcast(Person p) {
        if (p instanceof Student) {
            Student s = (Student) p;
            System.out.println("ok, 하향 타입 변환");
        }
    }
}
```

타입 변환과 다형성 (참고)

개선된 instanceof 연산자

- 타입 확인과 캐스팅을 동시에 수행
- Java 16에서 도입

<변수> instanceof <타입> <새로 선언할 변수>

기존 instanceof

```
if (p instanceof Student) {
  Student s = (Student) p;
  s.work();
}
```

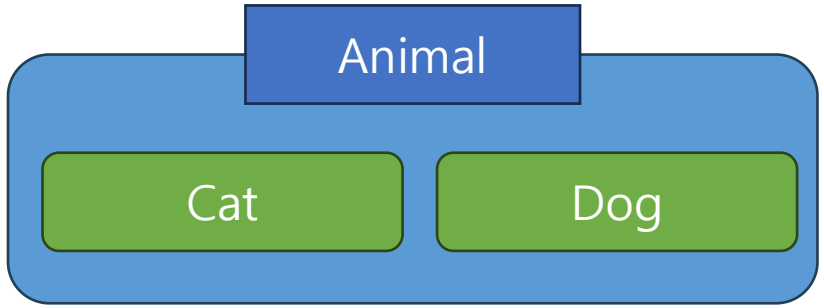
개선된 instanceof

```
if (p instanceof Student s) {
    s.work();
}
```

타입 변환과 다형성

다형성

- 하나의 참조 변수에 여러 객체를 대입하여, 동일한 명령어를 적용하더라도 객체의 종류에 따라 다양한 동작을 수행할 수 있는 성질.




```
Animal a = new Animal();
Animal b = new Dog();
Animal c = new Cat();
```

타입 변환과 다형성

동적 바인딩

- 실행 도중에 객체의 실제 타입을 기준으로 오버라이딩된 메서드를 연결하고 호출하는 것.

```java
class Animal {
  void move() {
    System.out.println("Animal-move");
  }
}

class Dog extends Animal {
  void move() {
    System.out.println("Dog-move");
  }
}
```

```java
Animal a = new Animal();
Animal b = new Dog();
a.move(); // Animal-Move
b.move(); // Dog-Move
```

타입 변환과 다형성

Practice

예제 6-20: 오버라이딩 여부에 따른 타입 변환의 영향

```java
package chap06;

class Vehicle {
    String name = "탈 것";

    void whoami() {
        System.out.println("나는 탈 것이다.");
    }

    static void move() {
        System.out.println("이동하다.");
    }
}

class Car extends Vehicle {
    String name = "자동차";

    void whoami() {
        System.out.println("나는 자동차이다.");
    }

    static void move() {
        System.out.println("달리다.");
    }
}

public class OverTypeDemo {
    public static void main(String[] args) {
        Vehicle v = new Car();
        System.out.println(v.name);
        v.whoami();
        v.move();
    }
}
```