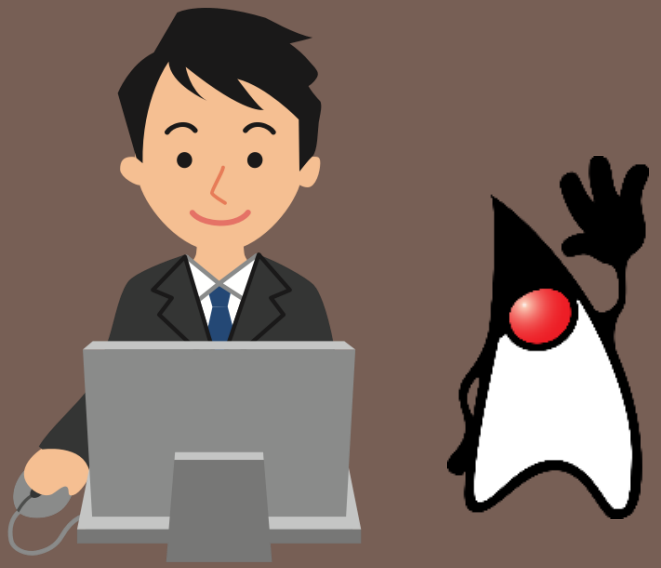
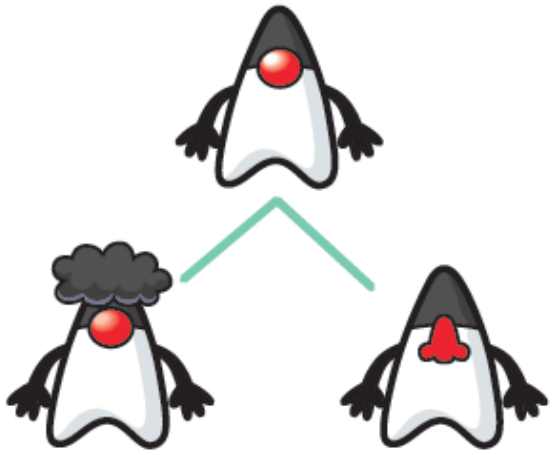
파워자바(개정3 판)



6장 상속

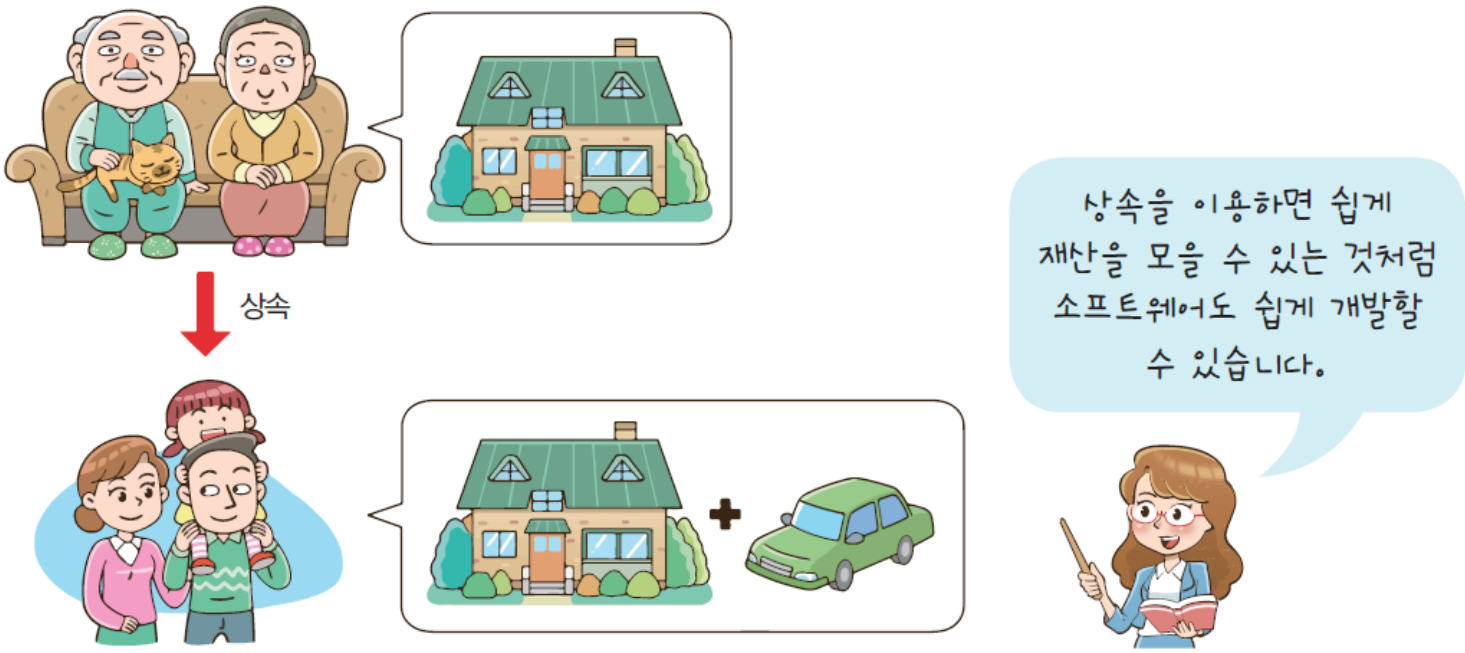
6장의 목표

1. 상속이 왜 필요한지를 설명할 수 있나요?
2. 상속을 이용하여 자식 클래스를 작성할 수 있나요?
3. 부모 클래스의 어떤 부분에 접근할 수 있는지를 설명할 수 있나요?
4. 오버라이딩을 이용하여 부모 클래스의 메소드를 재정의할 수 있나요?
5. 추상 클래스와 인터페이스를 이용하여 코드를 작성할 수 있나요?
6. 상속과 구성의 차이점을 설명하고 적절하게 사용할 수 있나요?



상속이란?

- 상속의 개념은 현실 세계에도 존재한다.



상속의 형식

Syntax: 상속

자식 클래스 또는 서브 클래스

부모 클래스 또는 슈퍼 클래스

```
class ElectricCar extends Car  {
   int batteryLevel;
   public void charge(int amount) {
      batteryLevel += amount;
   }
}
```

- 상속을 정의하려면 자식 클래스 이름 뒤에 extends를 쓰고 부모 클래스 이름을 적으면 된다.
- “extends”는 확장(또는 파생)한다는 의미이다. 즉 부모 클래스를 확장하여서 자식 클래스를 작성한다는 의미가 된다.

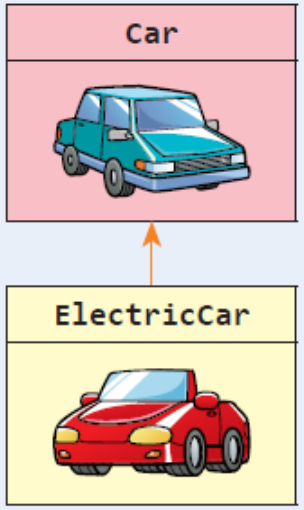
상속의 예

ElectricCar.java

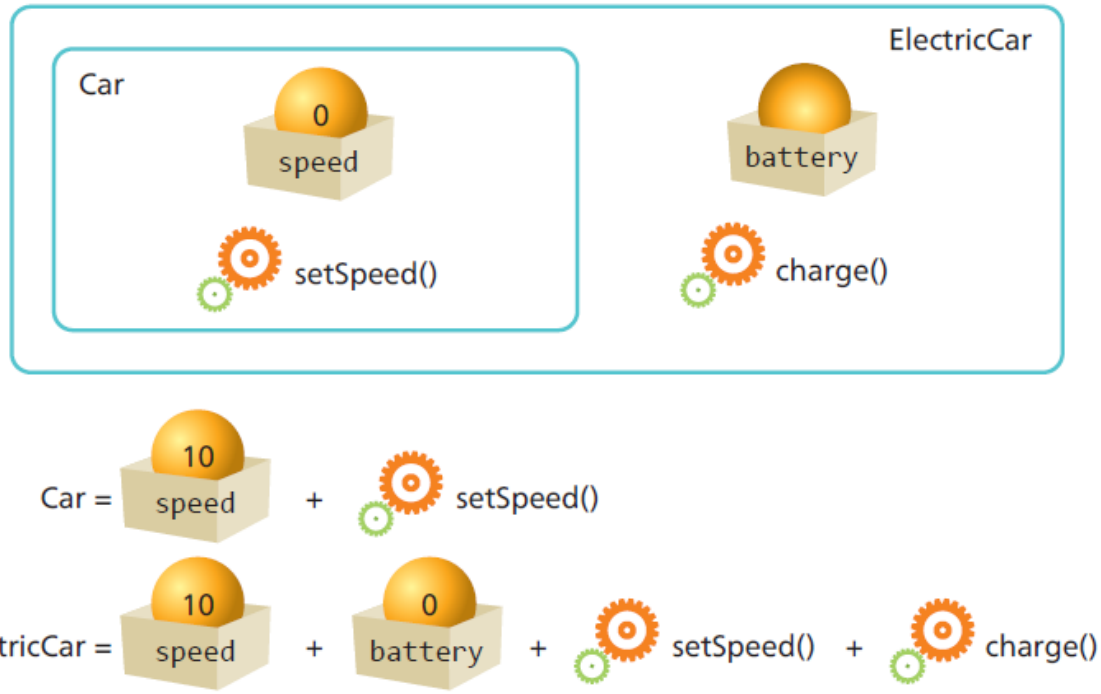
```
01  class Car {
02      int speed; // 속도
03      public void setSpeed(int speed) {     // 속도 변경 메소드
04          this.speed = speed;
05      }
06  }
07
08  class ElectricCar extends Car  {
09      int battery;
10
11      public void charge(imt amount) {      // 충전 메소드
12          battery += amount;
13      }
14  }
```

추가된 필드

추가된 메소드



무엇이 상속되는가?


ElectricCar
Car
0
speed
battery
setSpeed()
charge()
Car =
10
speed
+
setSpeed()
ElectricCar =
10
speed
+
0
battery
+
setSpeed()
+
charge()

상속받은 필드와 메소드 사용하기

ElectricCarTest.java

```
01  public class ElectricCarTest {
02     public static void main(String[] args) {
03        ElectricCar  obj = new ElectricCar();
04        obj.speed = 10;
05        obj.setSpeed(60);
06        obj.charge(10);
07     }
08  }
```

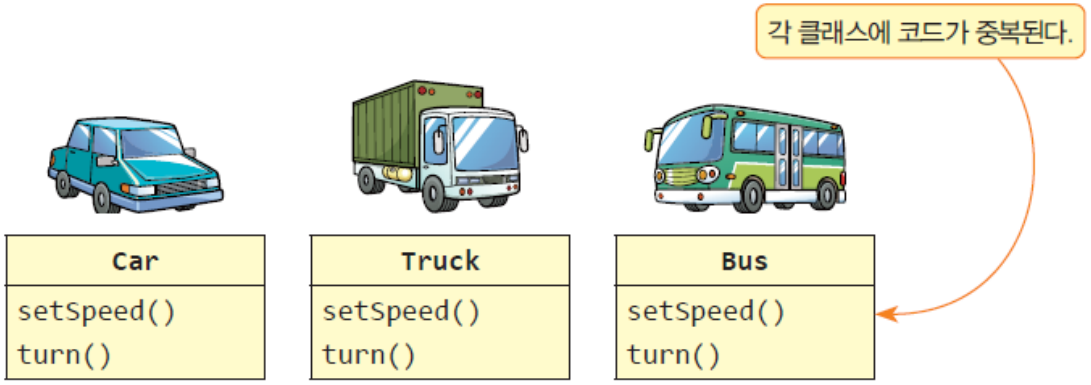
자식 클래스 객체 생성

부모 클래스의 필드와 메소드 사용

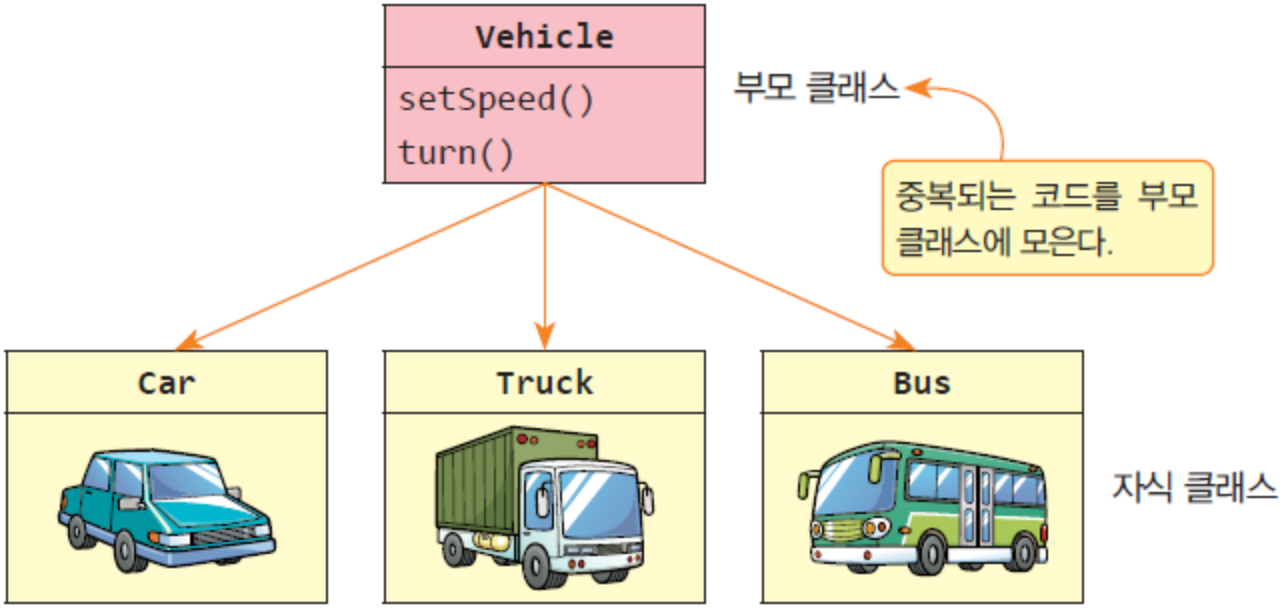
자체 메소드 사용

왜 상속을 사용하는가?

- 만약 우리가 원하는 코드를 가진 클래스가 이미 존재한다면 이 클래스를 상속받아서 이미 존재하는 클래스의 필드와 메소드를 재사용할 수 있다.
- 상속을 사용하면 중복되는 코드를 줄일 수 있다.



왜 상속을 사용하는가?


Vehicle
setSpeed()
turn()
부모 클래스
중복되는 코드를 부모 클래스에 모은다.
Car
Truck
Bus
자식 클래스

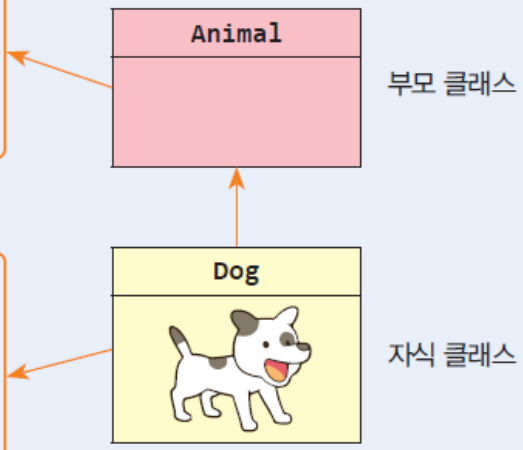
자바 상속의 특징

- 다중 상속을 지원하지 않는다. 다중 상속이란 여러 개의 클래스로부터 상속 받는 것이다. 자바에서는 금지되어 있다.
- 상속의 횟수에는 제한이 없다.
- 상속 계층 구조의 최상위에는 java.lang.Object 클래스가 있다.

예제: Animal 클래스와 Dog 클래스 만들어보기

DogTest.java

```
01  class Animal {
02     int age;
03     void eat() {
04        System.out.println("먹고 있음...");
05     }
06  }
07
08  class Dog extends Animal {
09     void bark() {
10        System.out.println("짖고 있음...");
11     }
12  }
13
14  public class DogTest {
15     public static void main(String args[]) {
16        Dog d = new Dog();
17        d.bark();
18        d.eat();
19     }
20  }
```



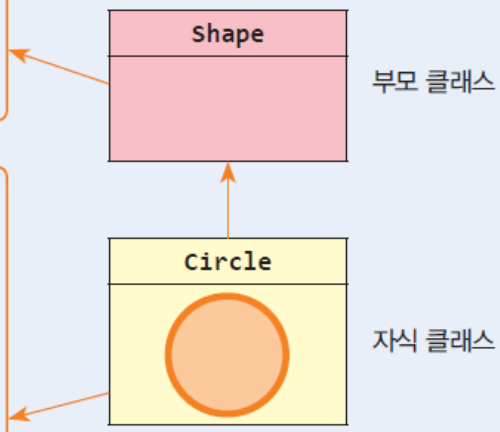

```
짖고 있음...
먹고 있음...
```

예제: 도형 예제

- 일반적인 도형을 나타내는 Shape 클래스를 작성하고 이것을 상속받아서 원을 나타내는 Circle 클래스를 작성해보자.

CircleTest.java

```
01  class Shape {
02     int x, y;
03  }
04
05  class Circle extends Shape {
06     int radius;
07
08     public Circle(int radius) {
09        this.radius = radius;
10        x = 0;
11        y = 0;
12     }
13
14     double getArea() {
15        return 3.14 * radius * radius;
16     }
17  }
```



예제: 도형 예제

```
18
19  public class CircleTest {
20     public static void main(String args[]) {
21        Circle obj = new Circle(10);
22        System.out.println("원의 중심: (" + obj.x + "," + obj.y + ")");
23        System.out.println("원의 면적: " + obj.getArea());
24     }
25  }
```

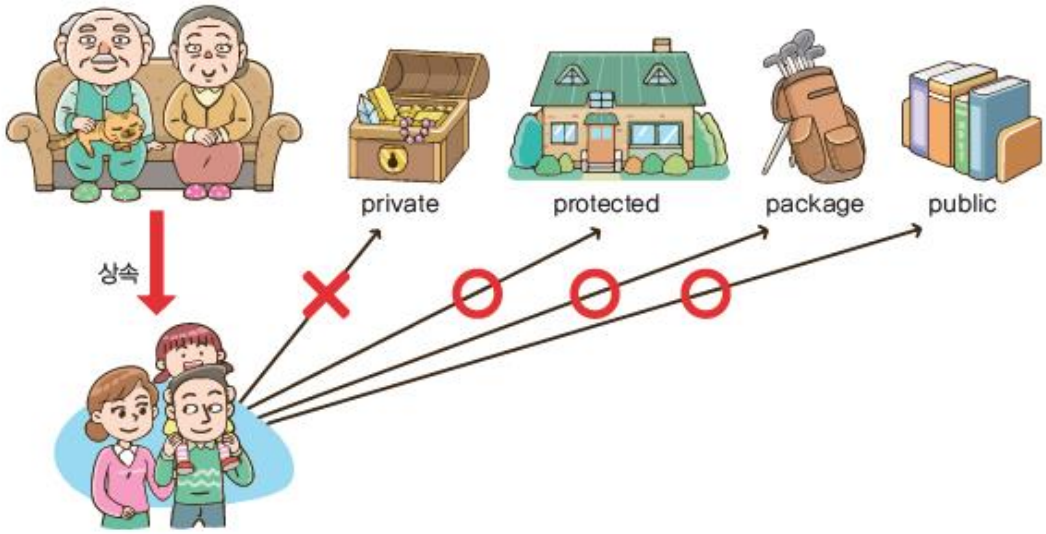
```
원의 중심: (0,0)
원의 면적: 314.0
```

중간점검

1. 상속의 장점은 무엇인가? 그리고 상속을 사용하는 이유는 무엇인가?
2. 프로그래머, 가수, 무용수를 클래스로 만들려고 한다. 부모 클래스를 어떻게 잡는 것이 코드의 길이를 줄일 수 있을까?
3. 자바에서 상속을 정의하는 키워드는 무엇인가?
4. 상속 계층도의 최상위 층에는 어떤 클래스가 있는가?
5. Animal 클래스[sleep(), eat()], Dog 클래스[sleep(), eat(), bark()], Cat 클래스[sleep(), eat(), play()]를 상속을 이용하여서 표현하면 어떻게 코드가 간결해지는가?
6. 일반적인 상자(box)를 클래스 Box로 표현하고, Box를 상속받는 서브 클래스인 ColorBox 클래스를 정의하여 보자. 적절한 필드(길이, 폭, 높이)와 메소드(부피 계산)를 정의한다.

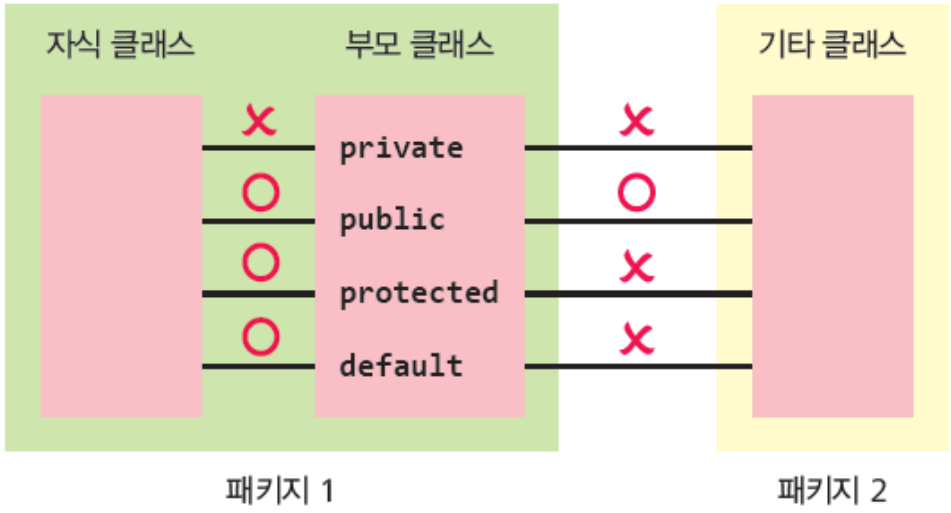
상속과 접근 지정자

- 자식 클래스는 부모 클래스의 public 멤버, protected 멤버, 디폴트 멤버(부모 클래스와 자식 클래스가 같은 패키지에 있다면)를 상속받는다. 하지만 부모 클래스의 private 멤버는 상속되지 않는다.



상속과 접근 지정자

	public	protected	default	private
동일한 클래스	O	O	O	O
동일한 패키지	O	O	O	X
자식 클래스	O	O	O	X
다른 패키지	O	X	X	X



예제 코드

Rectangle.java

protected로 선언되었다.

부모 클래스의 protected 멤버는 사용할 수 있다.

```
01  class Shape {
02     protected int x, y;
03     void print() {
04        System.out.println("x좌표: " + x + " y좌표: " + y);
05     }
06  }
07
08  public class Rectangle extends Shape {
09     int width, height;
10
11     double calcArea() {
12        return width * height;
13     }
14     void draw() {
15        System.out.println("(" + x + "," + y + ") 위치에 " + "가로: " + width
16              + " 세로: " + height);
17     }
18  }
```

예제: Person 클래스와 Student 클래스 만들어보기

- Person 클래스는 일반적인 사람을 나타낸다. Person 클래스를 상속받아서 Student 클래스를 작성해보자. Person 클래스 중에서 민감한 개인 정보는 private으로 지정한다. 예를 들어서 주민등록번호나 체중 같은 정보는 공개되면 안 된다. 민감하지 않은 정보는 protected로 지정한다. 공개해도 좋은 정보는 public으로 지정한다.

예제: Person 클래스와 Student 클래스 만들어보기

StudentTest.java

```
01  class Person {
02     private String regnumer;     // 주민등록번호, 자식 클래스에서 접근 불가
03     private double weight;       // 체중, 자식 클래스에서 접근 불가
04     protected int age;           // 나이, 자식 클래스에서 접근 가능
05     String name;                 // 이름, 어디서나 접근 가능
06
07     public double getWeight() {
08        return weight;
09     }
10     public void setWeight(double weight) {
11        this.weight = weight;
12     }
13  }
14
15  class Student extends Person {
16     int id;              // 학번
17  }
```

예제: Person 클래스와 Student 클래스 만들어보기

```
19  public class StudentTest {
20      public static void main(String args[]) {
21          Student obj = new Student();
22          //obj.regnumber = "123456-123456";        // 오류!!
23          //obj.weight = 75.0;                      // 오류!!
24          obj.age = 21;                             // OK
25          obj.name = "Kim";                         // OK
26          obj.setWeight(75.0);                      // OK
27      }
28  }
```

중간점검

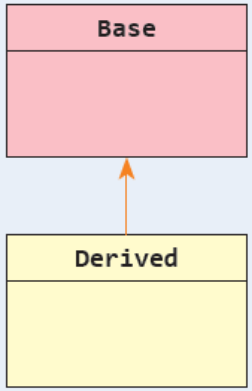
1. 부모 클래스와 자식 클래스만 사용하는 필드는 어떤 접근 지정자를 사용하는 것이 좋은가?
2. 자식 클래스가 부모 클래스에서 private로 선언된 필드에 접근할 수 있는가?
3. 필드 앞에 아무런 접근 지정자가 없다면 어디에 있는 클래스만 사용이 가능한가?

상속과 생성자

- 자식 클래스의 객체가 생성될 때, 자식 클래스의 생성자만 호출될까? 아니면 부모 클래스의 생성자도 호출되는가? 또 어떤 순서로 호출될까?

Test.java

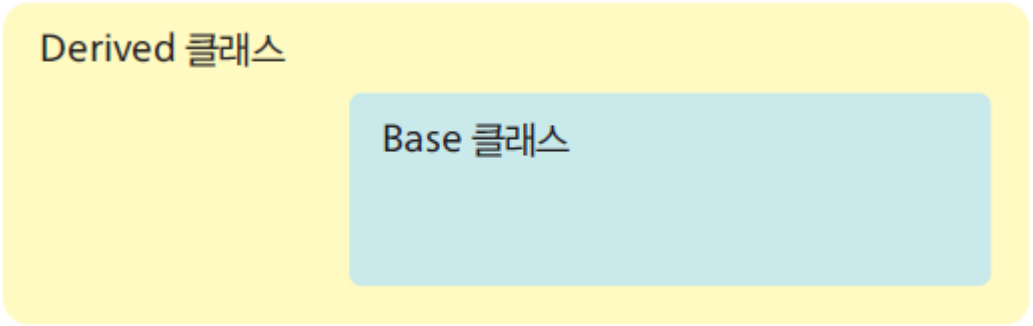
```
01  class Base {
02      public Base() {
03          System.out.println("Base() 생성자");
04      }
05  };
06
07  class Derived extends Base {
08      public Derived() {
09          System.out.println("Derived() 생성자");
10      }
11  };
12
13  public class Test {
14      public static void main(String[] args) {
15          Derived r = new Derived();
16      }
17  };
```




```
Base() 생성자
Derived() 생성자
```

왜 Derived 객체를 생성했는데 Base 생성자까지 호출되는 것일까?

- 자식 클래스 객체 안에는 부모 클래스에서 상속된 부분이 들어 있다. 따라서 자식 클래스 안의 부모 클래스 부분을 초기화하기 위하여 부모 클래스의 생성자도 호출되는 것이다.



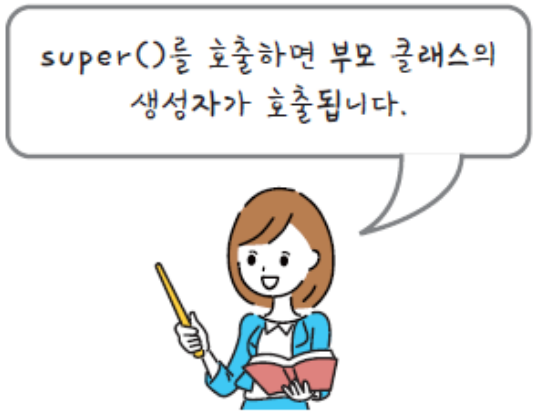
생성자의 호출 순서

- (부모 클래스의 생성자) -> (자식 클래스의 생성자) 순서이다.

명시적인 생성자 호출

```
class Base {
   public Base() {
      System.out.println("Base 생성자()");
   }
};
```

```
class Derived extends Base {
   public Derived() {
      super();
      System.out.println("Derived 생성자()");
   }
};
```



묵시적인 생성자 호출

```
class Base {
   public Base() {
      System.out.println("Base 생성자()");
   }
};
```

```
class Derived extends Base {
   public Derived() {
      System.out.println("Derived 생성자()");
   }
};
```

오류가 발생하는 경우

- 묵시적인 부모 클래스 생성자 호출을 사용하려면 부모 클래스에 기본 생성자(매개 변수가 없는생성자)가 반드시 정의되어 있어야 한다.

```
class Base {
   public Base(int x) {
      System.out.println("Base 생성자()");
   }
};
```

```
class Derived extends Base {
   public Derived() {
      System.out.println("Derived 생성자()");
   }
};
```

```
public class Test {
   public static void main(String[] args) {
      Derived obj = new Derived();
   }
};
```

부모 클래스의 생성자 선택

```
class TwoDimPoint {
   int x, y;

   public TwoDimPoint() {
      x = y = 0;
   }

   public TwoDimPoint(int x, int y) {
      this.x = x;
      this.y = y;
   }
};
```

```
class ThreeDimPoint extends TwoDimPoint {
   int z;
   public ThreeDimPoint(int x, int y, int z) {
      super(x, y);
      this.z = z;
   }
};
```

예제: Person 클래스와 Employee 클래스 만들어보기

- Person 클래스는 일반적인 사람을 나타낸다. Person 클래스를 상속받아서 직원을 나타내는 Employee 클래스를 작성해보자.

Employee.java

```
01  class Person {
02     String name;
03     public Person()   {          }
04     public Person(String theName)   {      this.name = theName;    }
05  }
06
07  class Employee extends Person {
08     String id;
09     public Employee()    {        super();      }
10     public Employee(String name)    {        super(name);      }
11     public Employee(String name, String id)    {
12        super(name);
13        this.id = id;
14     }
15     @Override
16     public String toString()
17                          { return "Employee [id=" + id + " name="+name+"]"; }
18  }
```

```
19  public class EmployeeTest {
20     public static void main(String[] args) {
21        Employee e = new Employee("Kim", "20210001");
22        System.out.println(e);
23     }
24  };
```

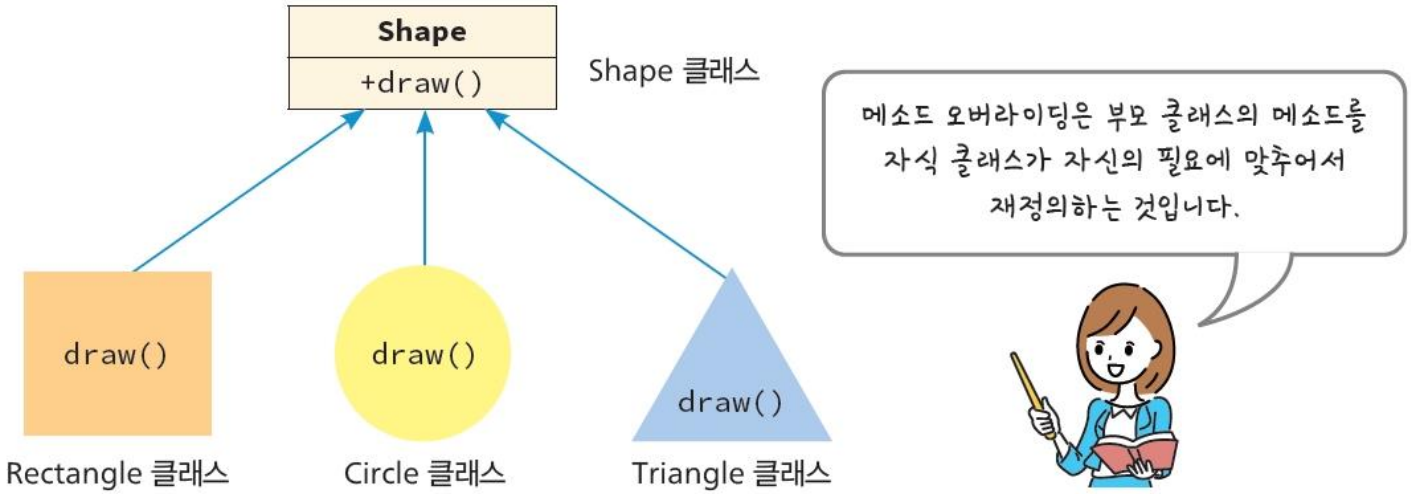
```
Employee [id=20210001 name=Kim]
```

중간점검

1. 상속에서 부모 생성자와 자식 생성자는 어떤 순서대로 호출되는가?
2. 부모 클래스에 기본 생성자가 없는 경우에, 묵시적인 생성자 호출을 사용하면 어떻게 되는가?

메소드 오버라이딩

- 메소드 오버라이딩(method overriding)은 자식 클래스가 부모 클래스의 메소드를 자신의 필요에 맞추어서 재정의하는 것이다. 이때 메소드의 이름이나 매개 변수, 반환형은 동일하여야 한다.



예제

ShapeTest.java

```
01  class Shape{
02     public void draw()  {    System.out.println("Shape");    }
03  }
04
05  class Circle extends Shape{
06  @Override
07     public void draw()  {     System.out.println("Circle을 그립니다.");    }
08  }
09
10  class Rectangle extends Shape{
11  @Override
12     public void draw()  {    System.out.println("Rectangle을 그립니다.");    }
13  }
14
15  class Triangle extends Shape{
16  @Override
17     public void draw()  {    System.out.println("Triangle을 그립니다.");    }
18  }
```

예제

```
20  public class ShapeTest {
21     public static void main(String[] args) {
22        Rectangle s = new Rectangle();
23        s.draw();
24     }
25  }
```

```
Rectangle을 그립니다.
```

Rectangle 클래스의 객체에 대하여 draw()가 호출되면 Rectangle 클래스 안에서 오버라이딩된 draw()가 호출된다. Shape의 draw()가 호출되는 것이 아니다.

경고

철자를 잘못 쓰는 경우, 컴파일러는 이것을 새로운 메소드 이름으로 인식한다(인공지능은 없다). 따라서 메소드 오버라이드가 일어나지 않는다.

이것을 방지하기 위해서 오버라이딩된 메소드 이름 앞에는 @Override 어노테이션을 붙이는 것이 좋다. 만약 부모 클래스에 그런 이름의 메소드가 없다면 컴파일러가 오류를 발생한다.

```
class Square extends Shape{
   @Override
   public void draw()  {  System.out.println("Square를 그립니다.");  }
};
```

키워드 super를 사용하여 부모 클래스 멤버 접근

- 키워드 super는 상속 관계에서 부모 클래스의 메소드나 필드를 명시적으로 참조하기 위하여 사용된다.
- 만약 부모 클래스의 메소드를 오버라이딩한 경우에 super를 사용하면 부모 클래스의 메소드를 호출할 수 있다.

예제

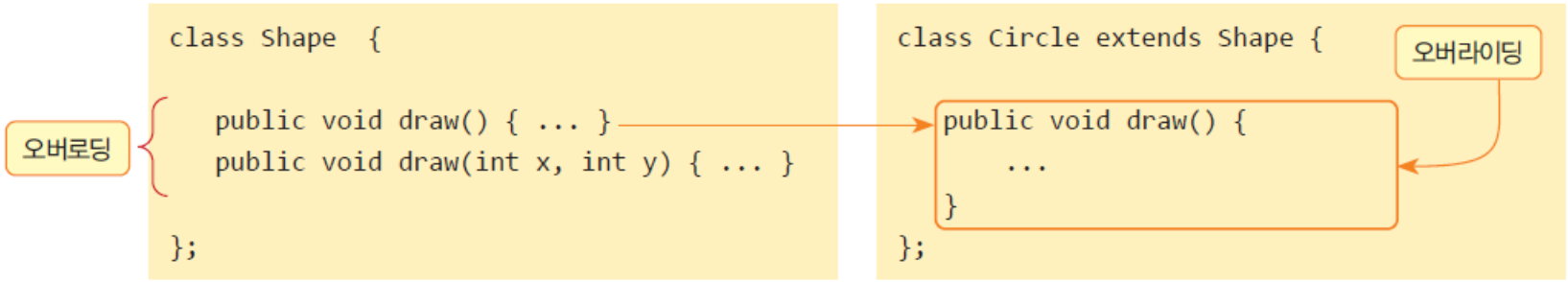
ShapeTest.java

```
01  class Shape{
02      public void draw()    {
03        System.out.println("Shape 중의 하나를 그릴 예정입니다.");
04     }
05  }
06
07  class Circle extends Shape{
08     @Override
09     public void draw()    {
10        super.draw();              // 부모 클래스의 draw() 호출
11        System.out.println("Circle을 그립니다.");
12     }
13  }
14
15  public class ShapeTest {
16     public static void main(String[] args) {
17        Circle s = new Circle();
18        s.draw();
19     }
20  }
```

```
Shape 중의 하나를 그릴 예정입니다.
Circle을 그립니다.
```

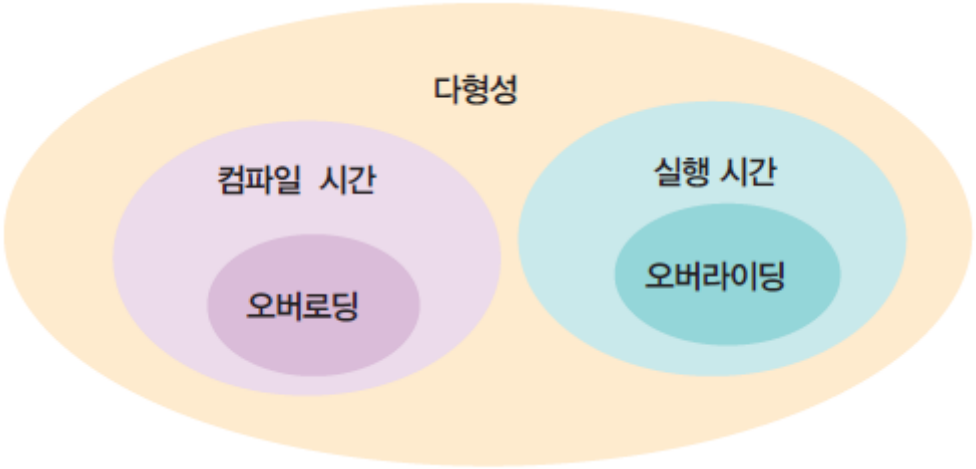
오버라이딩 vs 오버로딩

- 오버로딩(overloading)이란 같은 메소드명을 가진 여러 개의 메소드를 작성하는 것이다.
- 오버라이딩(overriding)은 부모 클래스의 메소드를 자식 클래스가 다시 정의하는 것을 의미한다.



다형성

- 이들은 모두 다형성과 관련이 있다. 이름을 재사용하는 것은 같다. 오버로딩은 컴파일 시간에서의 다형성을 지원한다.
- 메소드 오버라이딩을 사용하면 실행 시간에서의 다형성을 지원할 수 있다.



정적 메소드를 오버라이드하면 어떻게 될까?

- 자식 클래스가 부모 클래스의 정적 메소드와 동일한 정적 메소드를 정의하는 경우, 어떤 참조 변수를 통하여 호출되는지에 따라 달라진다.

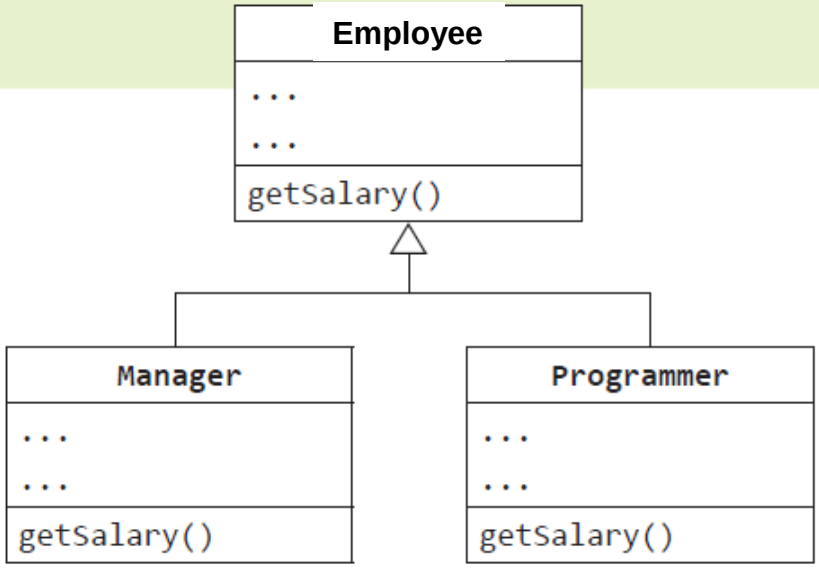
```
01  class Animal {
02     public static void A() {
03        System.out.println("static method in Animal");
04     }
05  }
06  public class Dog extends Animal {
07     public static void A() {
08        System.out.println("static method in Dog");
09     }
10     public static void main(String[] args) {
11        Dog dog = new Dog();
12        Animal a  = dog;
13        a.A();
14        dog.A();
15     }
16  }
```

```
static method in Animal
static method in Dog
```

Lab: Employee 클래스

- 아래와 같은 상속 계층도를 가정하자. 일반 직원은 Employee 클래스로 모델링한다. Employee 클래스를 상속받아서 관리자를 나타내는 Manager 클래스와 프로그래머를 나타내는 Programmer 클래스를 작성한다.

```
관리자의 월급: 5000000
프로그래머의 월급: 6000000
```



Lab: Employee 클래스

Test.java

```
01  class Employee {
02     public int baseSalary = 3000000;          // 기본금
03     int getSalary()    {    return baseSalary;    }
04  }
05
06  class Manager extends Employee {
07     @Override    int getSalary()    {    return (baseSalary + 2000000);    }
08  }
09
10  class Programmer extends Employee {
11     @Override    int getSalary()    {    return (baseSalary + 3000000);    }
12  }
13
14  public class Test {
15     public static void main(String[] args)    {
16        Manager obj1 = new Manager();
17        System.out.println("관리자의 월급: "+obj1.getSalary());
18
19        Programmer obj2 = new Programmer();
20        System.out.println("프로그래머의 월급: "+obj2.getSalary());
21     }
22  }
```

중간점검

1. 메소드 오버로딩과 메소드 오버라이딩은 어떻게 다른가?
2. 다음과 같은 코드에서 오버라이딩이 발생하는가? 그 이유는 무엇인가?

```
public class A {
   public void add(){ }
   public void add(int a){ }
}

public class B extends A {
   public float add(int b){ }
}
```

3. @Override의 의미는 무엇인가?

다형성이란?

- 다형성(polymorphism)이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것

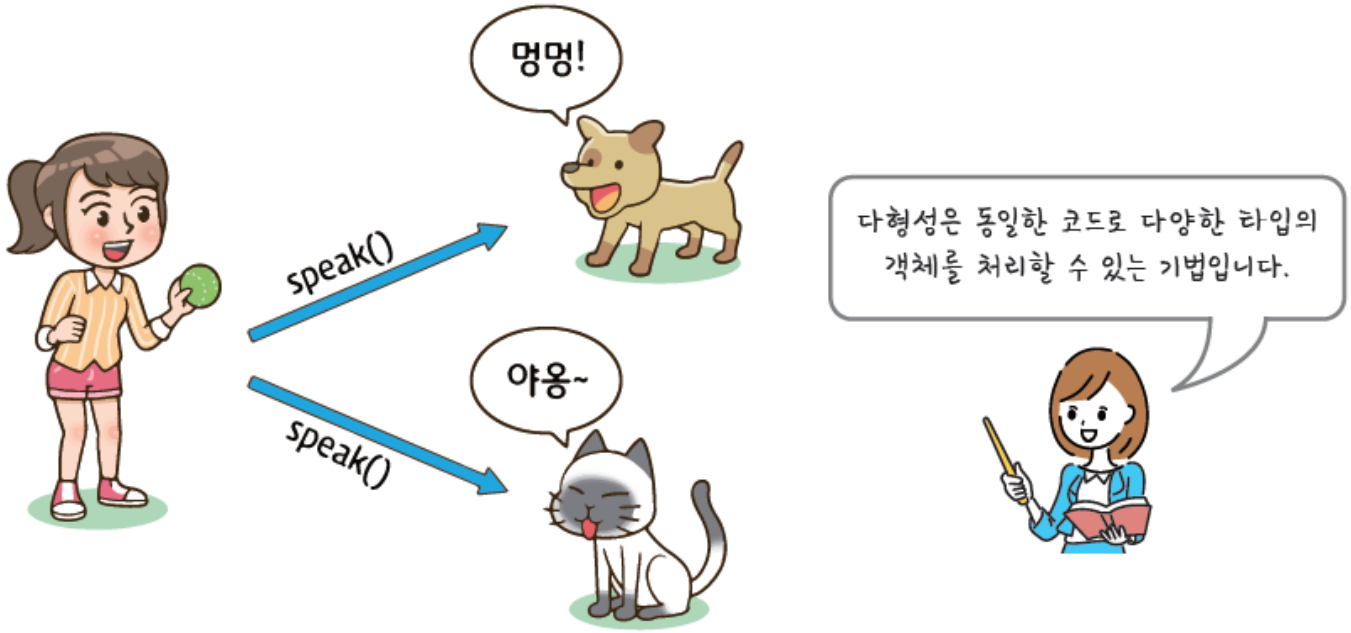
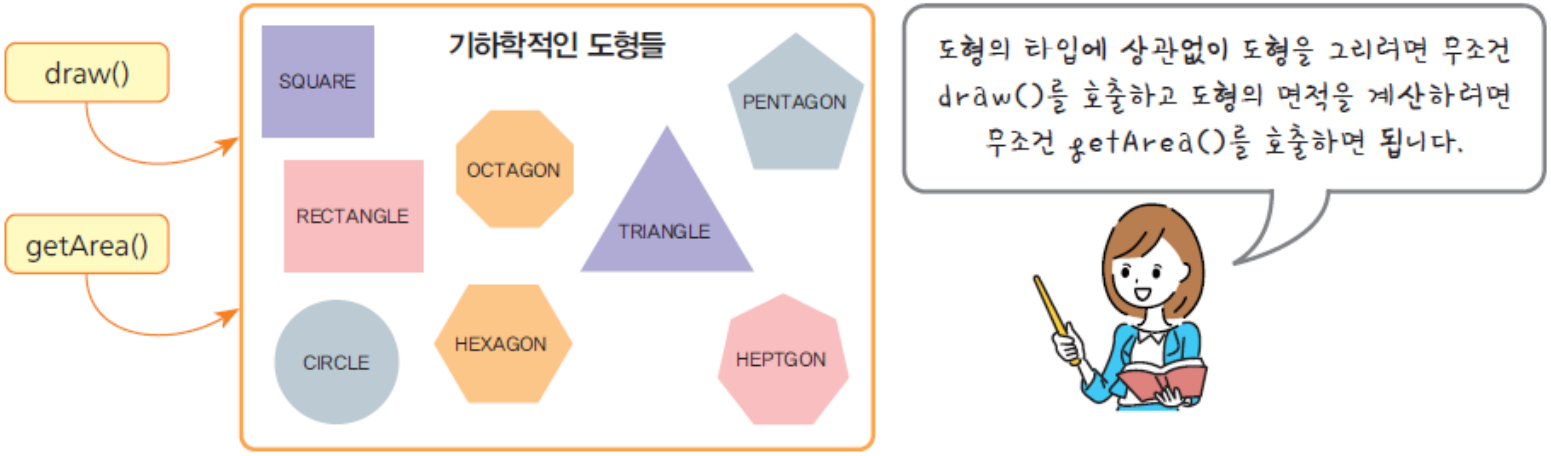


그림 6.1 다형성의 개념

다형성을 어떻게 사용할 수 있을까?

- 사각형, 삼각형, 원과 같은 다양한 타입의 도형 객체들이 모여 있다고 하자. 이 도형들을 그리고 싶으면 각 객체에 draw 메시지를 보내면 된다. 각 도형들은 자신의 모습을 화면에 그릴 것이다. 즉 도형의 타입을 고려할 필요가 없는 것이다



업캐스팅

- 하나의 예로 Rectangle, Triangle, Circle 등의 도형 클래스가 부모 클래스인 Shape 클래스로부터 상속되었다고 가정하자.

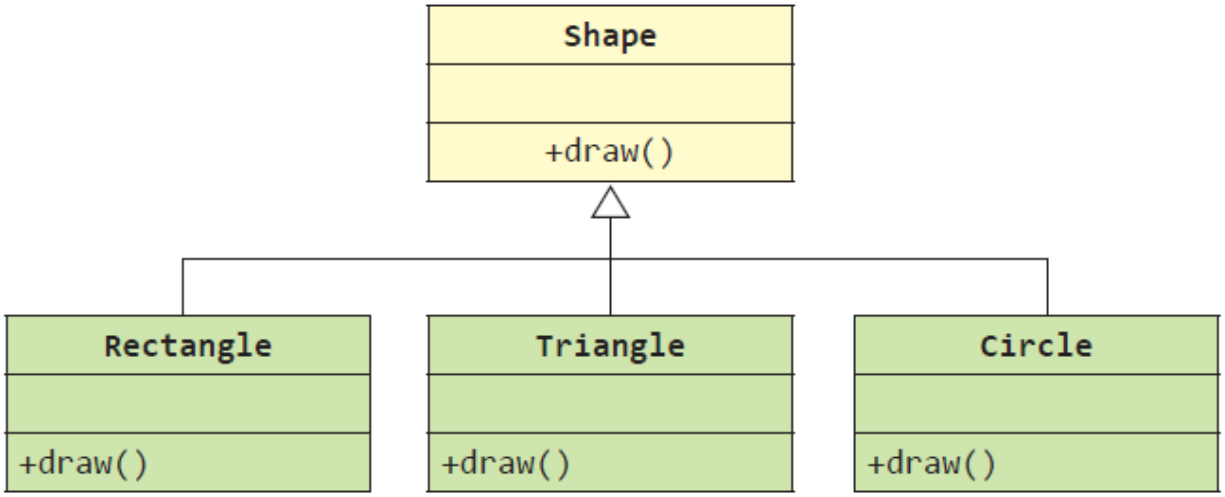


그림 6.2 도형의 상속 구조

업캐스팅

각 도형들은 2차원 공간에서 도형의 위치를 나타내는 기준점 (x, y)을 가진다. 이것은 모든 도형에 공통적인 속성이므로 부모 클래스인 Shape에 저장한다.

ShapeTest.java

```
01  class Shape {
02     protected int x, y;
03     public void draw() {    System.out.println("Shape Draw");    }
04  }
05
06  class Rectangle extends Shape {
07     private int width, height;
08     public void draw() {    System.out.println("Rectangle Draw");    }
09  }
10
11  class Triangle extends Shape {
12     private int base, height;
13     public void draw() {    System.out.println("Triangle Draw");    }
14  }
15
16  class Circle extends Shape {
17     private int radius;
18     public void draw() {    System.out.println("Circle Draw");    } }
19  }
```

이어서 Shape에서 상속받아서 사각형을 나타내는 클래스 Rectangle을 정의하여 보자. Rectangle은 추가적으로 width와 height 변수를 가진다. Shape 클래스의 draw()를 사각형을 그리도록 재정의한다. 물론 실제 그래픽은 아직까지 사용할 수 없으므로 화면에 사각형을 그린다는 메시지만을 출력한다.

서브 클래스인 Triangle을 Shape 클래스에서 상속받아 만든다.

업캐스팅

ShapeTest.java

```
01  public class ShapeTest {
02     public static void main(String arg[]) {
03        Shape s1, s2;
04
05        s1 = new Shape();        // ① 당연하다.
06        s2 = new Rectangle();    // ② Rectangle 객체를 Shape 변수로 가리킬 수 있을까?
07     }
08  }
```

부모 클래스 변수로 자식 클래스 객체를 참조할 수 있다. 이것을 업캐스팅(upcasting, 상형 형변환)이라고 한다.

자식 클래스와 부모 클래스의 포함 관계

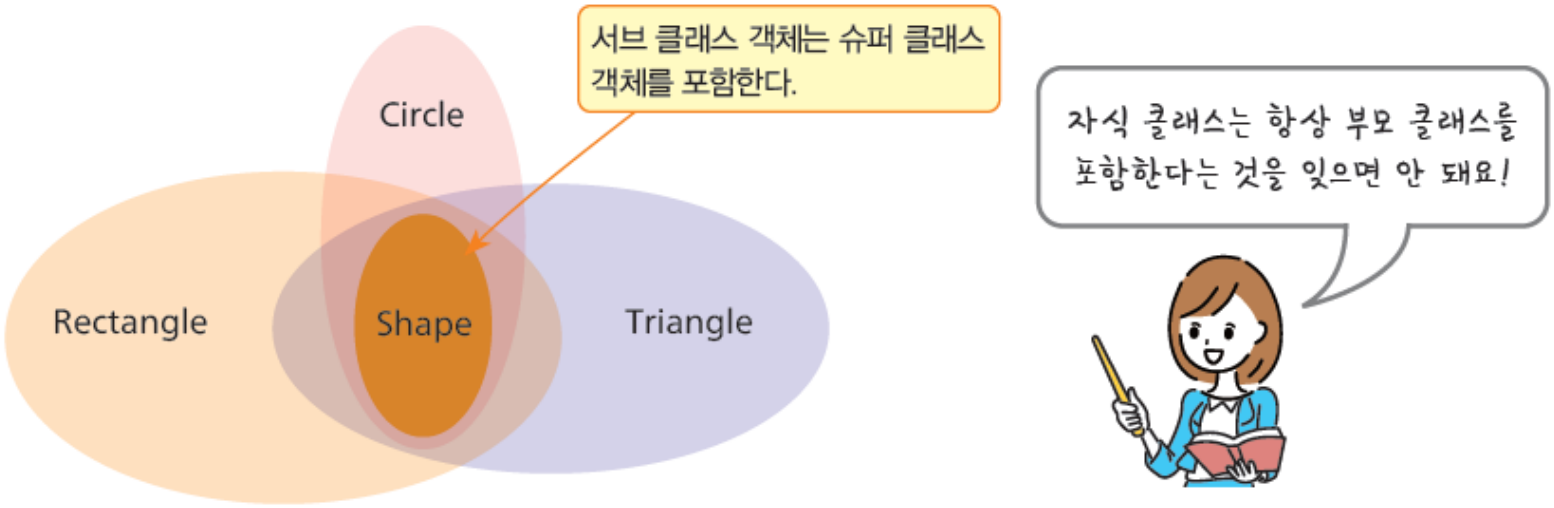


그림 6.3 자식 클래스와 부모 클래스의 포함 관계

업캐스팅

ShapeTest.java

```
01  ...
02  public class ShapeTest {
03     public static void main(String arg[]) {
04        Shape s = new Rectangle();
05        Rectangle r = new Rectangle();
06        s.x = 0;
07        s.y = 0;
08        s.width = 100;
09        s.height = 100;
10     }
11  }
```

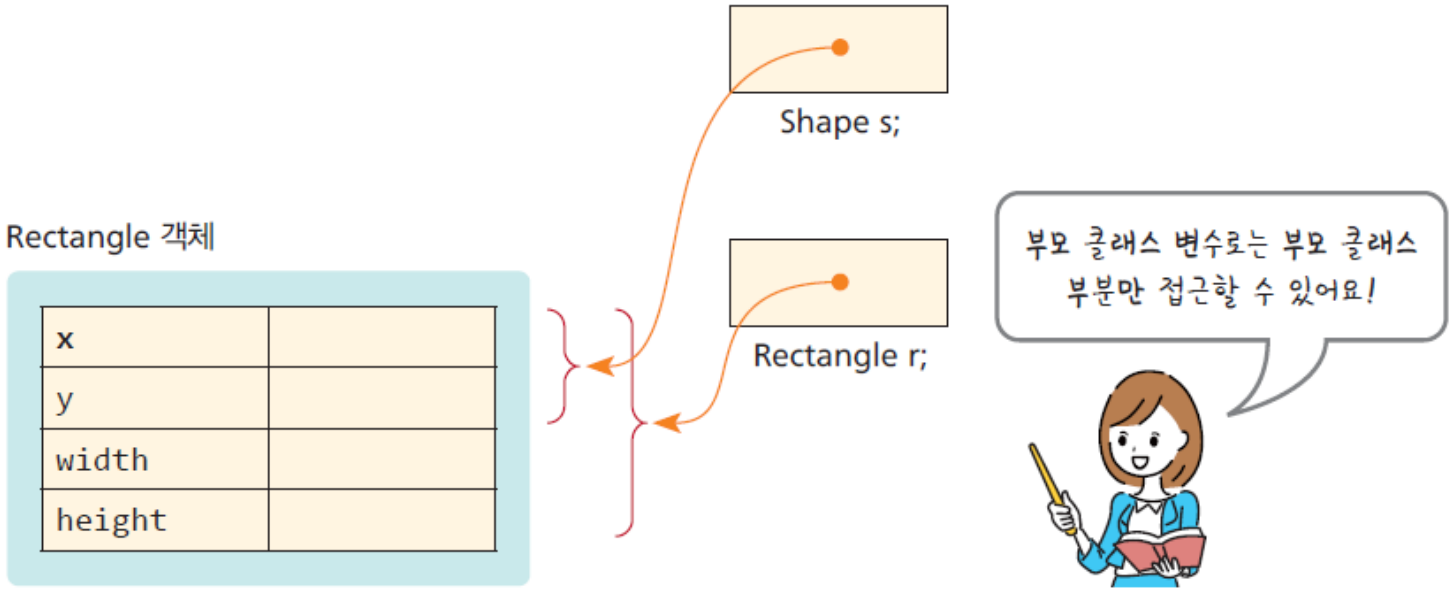
부모 클래스의 변수로 자식 클래스의 객체를 가리키는 것은 합법적이다.

Shape 클래스의 필드와 메소드에 접근하는 것은 OK

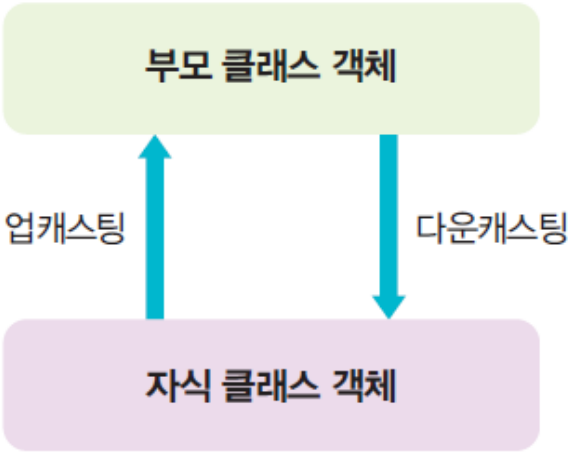
컴파일 오류가 발생한다. s를 통해서는 Rectangle 클래스의 필드와 메소드에 접근할 수 없다.

```
width cannot be resolved or is not a field
height cannot be resolved or is not a field
```

업캐스팅


Shape s;
Rectangle 객체
x
y
width
height
Rectangle r;
부모 클래스 변수로는 부모 클래스 부분만 접근할 수 있어요!

업캐스팅 vs 다운캐스팅



- 업캐스팅(upcasting): 자식 객체를 부모 참조 변수로 참조하는 것이다. 업캐스팅은 묵시적으로 수행될 수 있다. 업캐스팅을 사용하면 부모 클래스의 멤버에 접근할 수 있다. 하지만 자식 클래스의 멤버는 접근이 불가능하다.
- 다운캐스팅(downcasting): 부모 객체를 자식 참조 변수로 참조하는 것이다. 이것은 묵시적으로는 안 되고 명시적으로 하여야 한다.

업캐스팅 vs 다운캐스팅

Casting.java

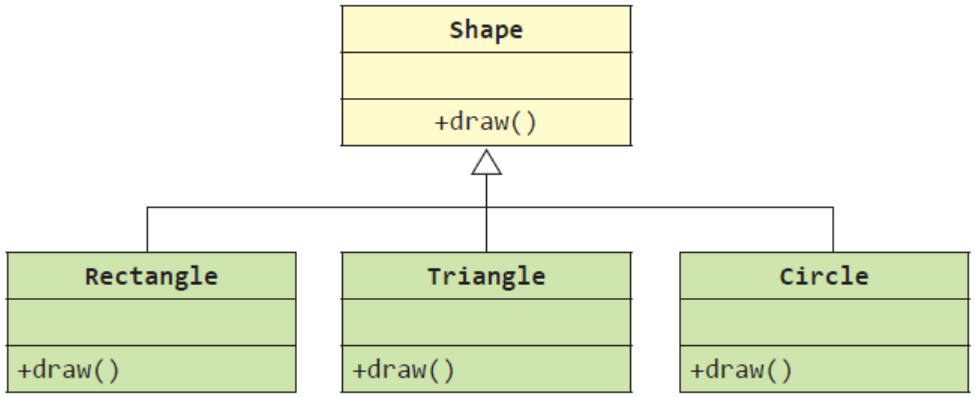
```java
class Parent {
    void print()    {    System.out.println("Parent 메소드 호출");    }
}

class Child extends Parent {
    @Override void print()   {    System.out.println("Child 메소드 호출");    }
}

public class Casting {
   public static void main(String[] args)    {
      Parent p = new Child();     // 업캐스팅: 자식 객체를 부모 객체로 형변환
      p.print();                  // 동적 메소드 호출, 자식의 print() 호출

     // Child c = new Parent();   // 이것은 컴파일 오류이다.

      Child c = (Child)p;         // 다운캐스팅: 부모 객체를 자식 객체로 형변환
      c.print();                  // 메소드 오버라이딩, 자식 객체의 print() 호출
   }
}
```

```
Child 메소드 호출
Child 메소드 호출
```

동적 바인딩

- “부모 참조 변수를 가지고 자식 객체를 참조하는 것이 도대체 어디에 필요한가요?” ->여러 가지 타입의 객체를 하나의 자료 구조 안에 모아서 처리하려는 경우에 필요하다.

- 모든 도형 클래스는 화면에 자신을 그리기 위한 메소드를 포함하고 있다고 가정한다. 이 메소드의 이름을 draw()라고 하자. 각 도형을 그리는 방법은 당연히 도형에 따라 다르다. 따라서 도형의 종류에 따라 서로 다른 draw()를 호출해야 한다. Shape가 draw() 메소드를 가지고 있고 Rectangle, Triangle, Circle 클래스들이 이 draw() 메소드를 오버라이딩하였다고 하자.



동적 바인딩

ShapeTest.java

```
...
public class ShapeTest {
   public static void main(String arg[]) {
        Shape[] arrayOfShapes;
        arrayOfShapes = new Shape[3];

        arrayOfShapes[0] = new Rectangle();
        arrayOfShapes[1] = new Triangle();
        arrayOfShapes[2] = new Circle();

        for (int i = 0; i < arrayOfShapes.length; i++) {
             arrayOfShapes[i].draw();
        }
   }
}
```

Shape의 배열 arrayOfShapes[]를 선언한다.

배열 arrayOfShapes의 각 원소에 객체를 만들어 대입한다.
다형성에 의하여 Shape 객체 배열에 모든 타입의 객체를 저장할 수 있다.

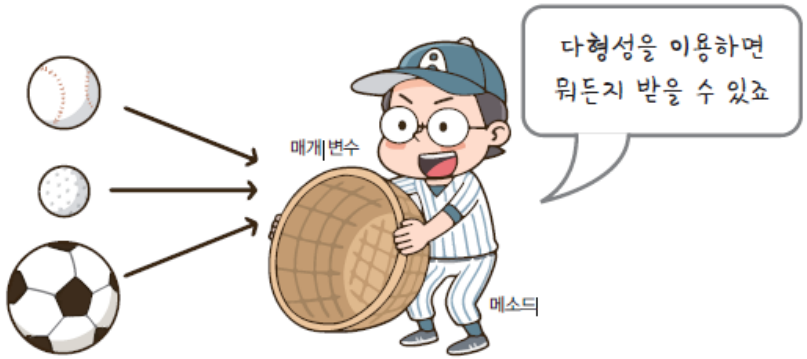
배열 arrayOfShapes[] 길이만큼 루프를 돌면서 각 배열 원소를 사용하여 draw() 메소드를 호출해본다. 어떤 draw()가 호출될까? 각 원소가 실제로 가리키고 있는 객체에 따라 서로 다른 draw()가 호출된다.

```
Shape Draw
Rectangle Draw
Triangle Draw
Circle Draw
```

업캐스팅의 활용

- 메소드의 매개 변수를 부모 타입으로 선언하면 훨씬 넓은 범위의 객체를 받을 수 있다. 예를 들어서 메소드의 매개 변수를 Rectangle 타입으로 선언하는 것보다 Shape 타입으로 선언하면 훨씬 넓은 범위의 객체를 받을 수 있다.

```
public static void printLocation(Shape s) {
   System.out.println("x=" + s.x + " y=" + s.y);
}
```



업캐스팅의 활용

ShapeTest.java

```
01  ...
02  public class ShapeTest {
03
04     public static void print(Shape s) {
05        System.out.println("x=" + s.x + " y=" + s.y);
06     }
07
08     public static void main(String arg[]) {
09        Rectangle s1 = new Rectangle();
10        Triangle s2 = new Triangle();
11        Circle s3 = new Circle();
12
13        print(s1);
14        print(s2);
15        print(s3);
16     }
17  }
```

Shape에서 파생된 모든 클래스의 객체를 다 받을 수 있다.

instanceof 연산자

- 변수가 가리키는 객체의 실제 타입을 알고 싶으면 instanceof 연산자를 사용하면 된다.

```
public class ShapeTest4 {
   public static void print(Shape obj) {
      if (obj instanceof Rectangle)
         System.out.println("실제 타입은 Rectangle");
      if (obj instanceof Triangle)
         System.out.println("실제 타입은 Triangle");
      if (obj instanceof Circle)
         System.out.println("실제 타입은 Circle");
   }
```

종단 클래스과 종단 메소드

- 종단 클래스(final class)는 상속을 시킬 수 없는 클래스를 말한다.
- 종단 클래스가 필요한 이유는 주로 보안상의 이유 때문이다.

```
final class String {
   ...
}
```

```
class Baduk {
   enum BadukPlayer { WHITE, BLACK }
   ...
   final BadukPlayer getFirstPlayer() {
      return BadukPlayer.BLACK;
   }
}
```

서브 클래스에서 재정의할 수 없도록 final로 지정한다.

중간점검

1. 상속과 다형성은 어떤 관련이 있는가?
2. 어떻게 자식 클래스 참조 변수로 부모 클래스의 객체를 참조할 수 있는 것인가?
3. 업캐스팅과 다운캐스팅은 무엇인가?
4. 동적 바인딩이란 무엇인가?
5. 다형성의 장점은 무엇일까?

Lab: 도형 면적 계산하기

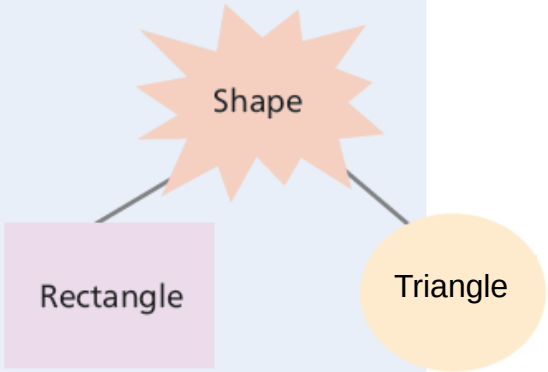
- 상속 계층 구조에서 Shape 클래스의 getArea()를 오버라이드하여서 각 도형에 맞는 면적을 계산해보자

```
Rectangle: 200.0
Triangle: 100.0
```

Sol: 도형 면적 계산하기

ShapeAreaTest.java

```
01  class Shape {
02     public double getArea() {          return 0; }
03     public Shape() {          super(); }
04  }
05
06  class Rectangle extends Shape {
07     private double width, height;
08     public Rectangle(double width, double height) {
09        super();
10        this.width = width;
11        this.height = height;
12     }
13     public double getArea() {          return width*height;  }
14  }
```



Sol: 도형 면적 계산하기

```
16  class Triangle extends Shape {
17     private double base, height;
18     public double getArea() {         return 0.5*base*height;    }
19     public Triangle(double base, double height) {
20        super();
21        this.base = base;
22        this.height = height;
23     }
24  }
25
26  public class ShapeAreaTest {
27     public static void main(String args[]) {
28        Shape obj1 = new Rectangle(10.0, 20.0);
29        Shape obj2 = new Triangle(10.0, 20.0);
30
31        System.out.println("Rectangle: " + obj1.getArea());
32        System.out.println("Triangle: " + obj2.getArea());
33     }
34  }
```

Lab: 동물 다형성

- 강아지와 고양이를 나타내는 클래스를 작성하자. 이들 클래스의 부모 클래스로 Animal 클래스를 정의한다. 강아지와 고양이 클래스의 speak() 메소드를 호출하면 각 동물들의 소리가 출력되도록 프로그램을 작성해보자.

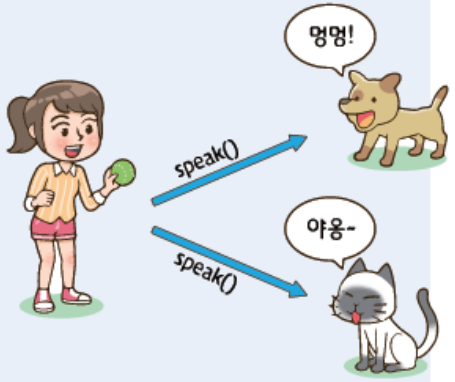
```
Animal 클래스의 speak()
멍멍
야옹
```

Sol:

DynamicCallTest.java

```
01  class Animal  {
02     void speak() {     System.out.println("Animal 클래스의 sound()");    }
03  }
04
05  class Dog extends Animal {
06     void speak() {     System.out.println("멍멍");    }
07  }
08
09  class Cat extends Animal {
10     void speak() {     System.out.println("야옹");    }
11  }
12
13  public class DynamicCallTest {
14     public static void main(String args[]) {
15        Animal a1 = new Dog();
16        Animal a2 = new Cat();
17
18        a1.speak();
19        a2.speak();
20     }
21  }
```

어떤 sound()가 호출될 것인지는 실행 시간에 참조되는 객체의 타입에 따라서 결정된다.



상속 vs 구성

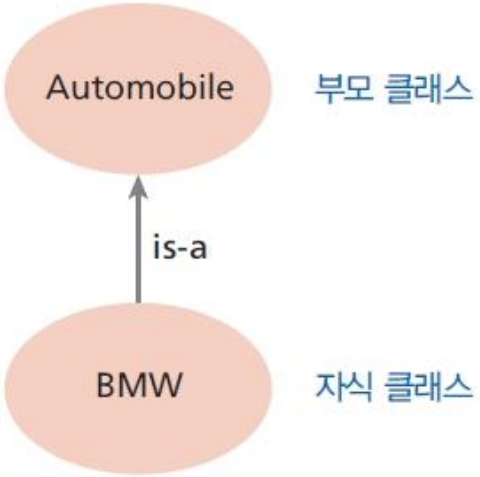
- 구성은 클래스가 다른 클래스의 인스턴스를 클래스의 필드로 가지는 디자인 기법이다. 반면에 상속은 한 객체가 클래스를 상속받아서 부모 객체의 속성과 동작을 획득할 수 있는 기법이다.
- 구성 및 상속은 모두 클래스를 연결하여 코드 재사용성을 제공한다. 상속과 구성은 객체 지향 개발자가 사용하는 두 가지의 중요한 프로그래밍 기술이다.
- 상속은 한 클래스를 다른 클래스에서 파생시키는 반면 구성은 하나의 클래스를 다른 클래스의 합으로정의한다.

상속 vs 구성

상속	구성
상속은 “is-a” 관계이다.	구성은 “has-a” 관계 이다.
상속에서는 하나의 클래스만 상속할 수 있으므로 하나의 클래스에서만 코드를 재사용할 수 있다.	여러 클래스에서 코드를 재사용할 수 있다.
상속은 컴파일 시간에 결정된다.	구성은 실행 시간에 결정될 수 있다.
final로 선언된 클래스의 코드를 재사용할 수 없다.	final로 선언된 클래스에서도 코드 재사용이 가능하다.
부모 클래스의 public 및 protected 메소드를 모두 노출한다.	아무것도 노출되지 않는다. 공개 인터페이스만을 사용하여 상호작용한다.

is-a 관계

- is-a 관계는 “A는 B의 일종이다.”라고 말하는 것과 같다. 예를 들어 사과는 과일의 일종이고, 자동차는 차량의 일종이다. is-a 관계가 있으면 상속을 사용한다. 상속은 단방향이다.
  - 자동차는 탈것이다(Car is a Vehicle).
  - 사자, 개, 고양이는 동물이다.
  - 집은 건축물의 일종이다.
  - 벤츠는 자동차의 일종이다.

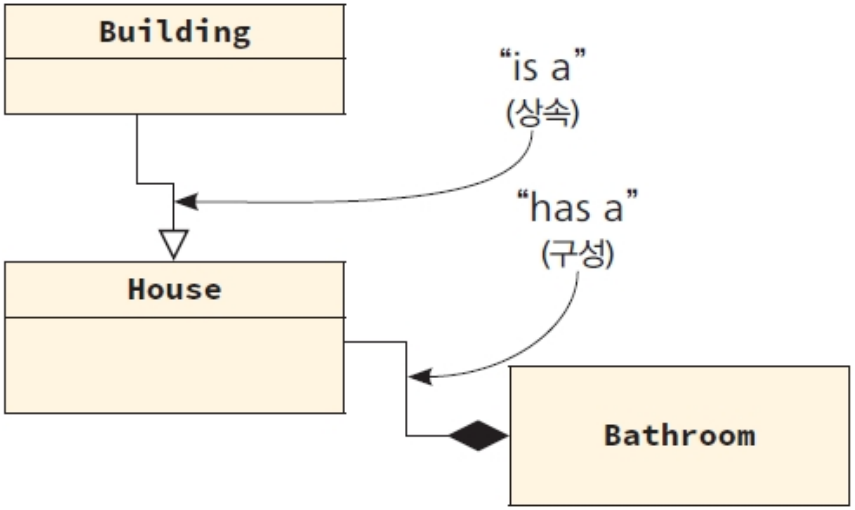


is-a 관계

```
01  class Animal {
02      void eat() {
03          System.out.println("음식을 먹습니다.");
04      }
05  }
06  public class Dog extends Animal {
07      void bark() {
08          System.out.println("멍멍");
09      }
10      public static void main(String args[]) {
11          Dog obj = new Dog();
12          obj.eat();
13          obj.bark();
14      }
15  }
```

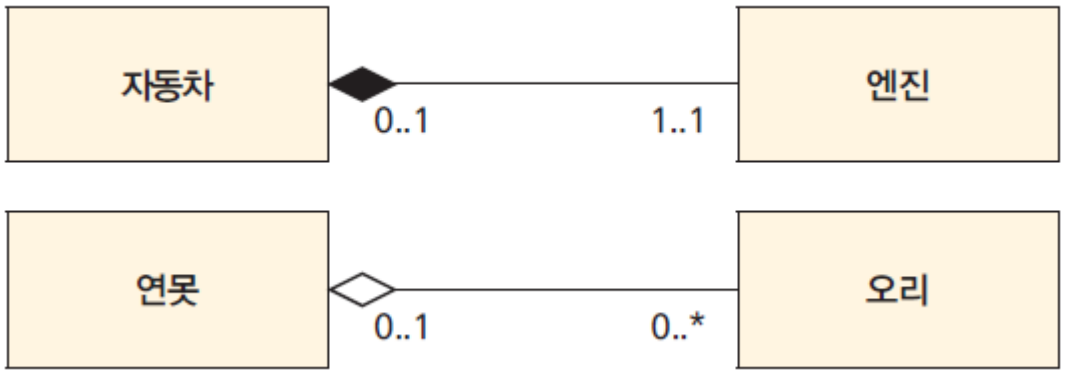
has-a 관계

- 만약 "~은 ~을 가지고 있다"와 같은 has-a(포함) 관계가 성립되면 이 관계는 상속으로 모델링을하면 안 된다. 이 경우에는 구성을 사용하여야 한다.
  - 자동차는 엔진을 가지고 있다.
  - 도서관은 책을 가지고 있다(Library has a book).
  - 집은 욕실을 가지고 있다.



has-a 관계

- 객체 지향 프로그래밍에서 has-a 관계는 구성 관계(composition) 또는 집합 관계(aggregation)를 나타낸다.



상단 그림에서 자동차는 엔진을 가지고 있다. 검정색 다이아몬드 표시는 구성(composition)을 나타낸다. 자동차가 엔진으로 구성되었다고 생각하여도 된다.

하단 그림에서 연못은 오리를 가지고 있다. 이때는 다이아몬드가 흰색인데 이것은 집합(aggregation)을 의미한다. 집합은 다이아몬드가 있는 객체가 다른 객체를 소유하고 있다는 것을 의미한다

has-a 관계

```
01  class Vehicle { }
02  class Engine  { }
03  class Brake  { }
04
05  public class Car extends Vehicle {
06     private Engine e;
07     private Brake b;
08     public Car() {
09        this.e = new Engine();
10        this.b = new Brake();
11     }
12  }
```

구성 vs 집합

- 구성: House에는 하나 이상의 Room이 있다. 이것은 구성이다. Room은 House 없이는 존재하지 않으므로 Room의 수명은 House에 의해 제어된다.

```
public class A {
   private B b = new B();

   public A() {     }
}
```

구성 vs 집합

- 집합: Block을 모아서 ToyHouse를 만들 수 있다. 이것은 집합이다. ToyHouse가 분해되더라도 Block들은 남는다.

```
public class A {
   private B b;

   public A( B b ) {    this.b = b;   }
}
```

잘못 사용하는 경우

```
01  import java.util.ArrayList;
02
03  public class Test extends ArrayList<Object> {
04      public static void main(String[] argv) {
05          Test obj = new Test();
06          obj.add("Kim");
07          obj.add("Park");
08      }
09  }
```

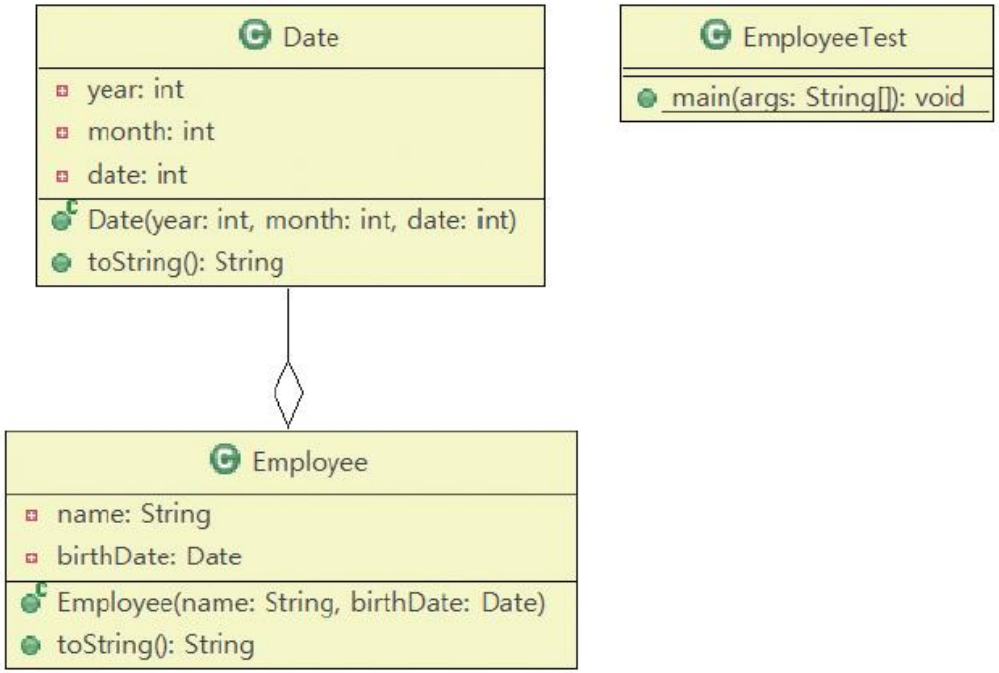
Test 클래스와 ArrayList 클래스 간에는 전혀 관련이 없으므로 “is-a” 관계가 아니다.

올바른 사용

```
01  import java.util.ArrayList;
02
03  public class Test2 {
04      static ArrayList<String> list = new ArrayList<>();
05
06      public static void main(String argv[]) {
07          list.add("Kim");
08          list.add("Park");
09      }
10  }
```

예제: 구성과 상속

- 간단한 예제를 살펴보자. 날짜를 나타내는 Date 클래스가 있다. 그리고 직원을 나타내는 Employee 클래스가 있다. Employee 클래스는 직원의 생일을 나타내는 필드인 birthDate를 가지고 있는데 이것은 Date 클래스의 참조 변수이다. 따라서 Employee 클래스는 Date 클래스를 가지고 있다고 할 수 있다. 전체를 UML 클래스 다이어그램으로 그리면 왼쪽과 같다.



예제: 구성과 상속

EmployeeTest.java

```
01  class Date {
02     private int year, month, date;
03
04     public Date(int year, int month, int date) {
05        this.year = year;
06        this.month = month;
07        this.date = date;
08     }
09
10     @Override
11     public String toString() {
12        return "Date [year=" + year + ", month=" + month + ", date=" + date + "]";
13     }
14  }
15  class Employee {
16     private String name;
17     private Date birthDate;
18
```

예제: 구성과 상속

```
19      public Employee(String name, Date birthDate) {
20          this.name = name;
21          this.birthDate = birthDate;
22      }
23
24      @Override
25      public String toString() {
26          return "Employee [name=" + name + ", birthDate=" + birthDate + "]";
27      }
28  }
29  public class EmployeeTest {
30      public static void main(String[] args) {
31          Date birth = new Date(1990, 1, 1);
32          Employee employee = new Employee("홍길동", birth);
33          System.out.println(employee);
34      }
35  }
```

```
Employee [name=홍길동, birthDate=Date [year=1990, month=1, date=1]]
```

중간점검

1. 어떤 경우에 상속을 사용해야 하는가?
2. 어떤 경우에 구성을 사용해야 하는가?
3. 구성과 결합의 차이점은 무엇인가?
4. 위임(delegation)이란 무엇인가?

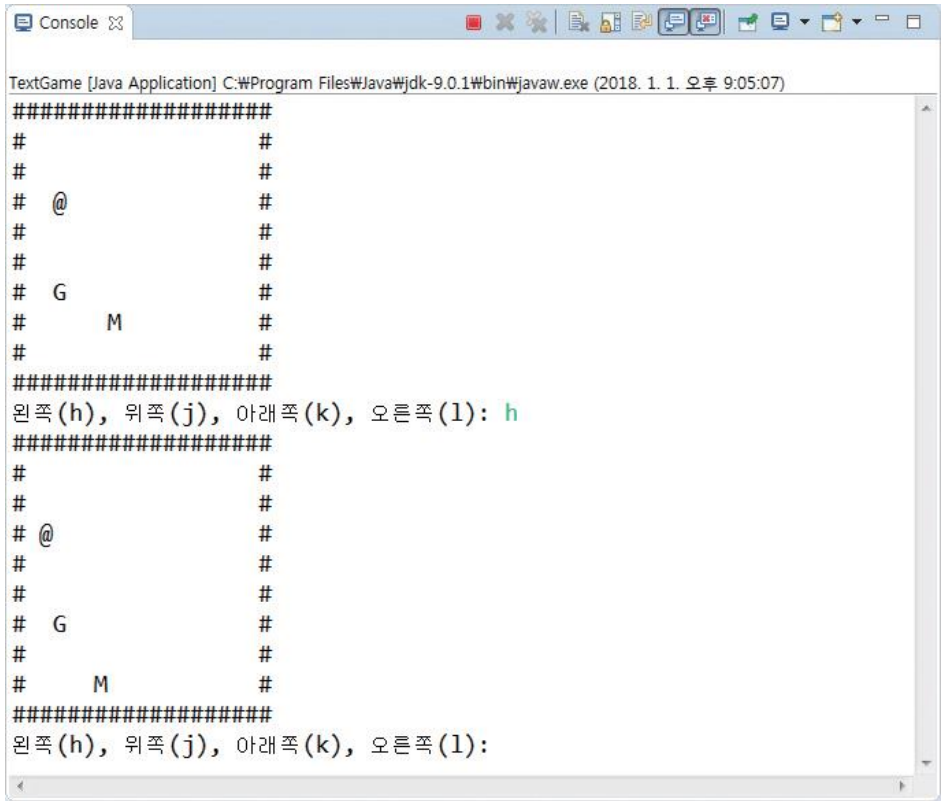
Mini Project: 카드(Card)와 덱(Deck)

- 카드를 나타내는 Card 클래스를 작성하고 52개의 Card 객체를 가지고 있는 Deck 클래스를 작성한다. 각 클래스의 __str__() 메소드를 구현하여서 덱 안에 들어 있는 카드를 다음과 같이 출력한다.

```
['클럽 에이스', '클럽 2', '클럽 3', '클럽 4', '클럽 5', '클럽 6', '클럽 7', '클럽 8', '클럽 9', '클럽 10', '클럽 잭', '클럽 퀸', '클럽 킹', '다이아몬드 에이스', '다이아몬드 2', '다이아몬드 3', '다이아몬드 4', '다이아몬드 5', '다이아몬드 6', '다이아몬드 7', '다이아몬드 8', '다이아몬드 9', '다이아몬드 10', '다이아몬드 잭', '다이아몬드 퀸', '다이아몬드 킹', '하트 에이스', '하트 2', '하트 3', '하트 4', '하트 5', '하트 6', '하트 7', '하트 8', '하트 9', '하트 10', '하트 잭', '하트 퀸', '하트 킹', '스페이드 에이스', '스페이드 2', '스페이드 3', '스페이드 4', '스페이드 5', '스페이드 6', '스페이드 7', '스페이드 8', '스페이드 9', '스페이드 10', '스페이드 잭', '스페이드 퀸', '스페이드 킹']
```

Mini Project: 텍스트 기반 게임

- 주인공을 움직여서 몬스터를 피하고 황금을 차지하는 게임이다. 화면에는 3가지의 스프라이트가 나타난다. '@'은 주인공 캐릭터이고 'M'은 몬스터, 'G'는 황금을 나타낸다. 주인공 캐릭터는 키보드의 h, j, k, l 키를 이용하여 상하좌우로 움직인다. 몬스터는 난수만큼 이동한다. 황금은 움직이지 않는다.



Summary

- 상속(inheritance)은 기존에 존재하는 클래스로부터 필드와 메소드를 이어받고, 필요한 기능을 추가할 수 있는 기법이다.
- 상속을 이용하면 여러 클래스에 공통적인 코드들을 하나의 클래스로 모을 수 있어서 코드의 중복을 줄일 수 있다.
- 자바에서는 extends 키워드를 이용하여 상속을 나타낸다. 상속하는 클래스를 부모 클래스(슈퍼 클래스)라고 하고 상속받는 클래스를 자식 클래스(서브 클래스)라고 한다.
- 자식 클래스는 부모 클래스의 public 멤버, protected 멤버, 디폴트 멤버(부모 클래스와 자식 클래스가 같은 패키지에 있다면)를 상속받는다.
- (부모 클래스의 생성자) -> (자식 클래스의 생성자) 순으로 호출된다.
- 메소드 오버라이딩(method overriding)은 자식 클래스가 부모 클래스의 메소드를 자신의 필요에 맞추어서 재정의하는 것이다.
- 오버로딩(overloading)이란 같은 메소드명을 가진 여러 개의 메소드를 작성하는 것이다.

Q & A