

# Java 프로그래밍 길잡이

✓ 원리를 알면 IT가 맞았다

**Java Programming for Beginners**



chapter 06.

---

## 클래스들의 관계

- 클래스들의 관계 및 종류에 대해서 학습한다.
- 상속의 개념 및 특징에 관하여 학습한다.
- 접근 지정자에 관하여 학습한다.
- 오버라이딩 메소드에 대해서 학습한다.
- 다형성 개념 및 특징에 관하여 학습한다.
- 최상위 클래스인 Object 클래스에 관하여 학습한다.

## □ 1) 클래스들의 관계 [ Class Relationship ]

- 현실세계에서는 객체간에 특별한 관계를 가지고 존재하게 된다. 회사에서는 상사와 부하직원 관계, 학교에서는 선생님과 친구관계, 가정에서는 부모와 자식 관계등으로 맺어져 있다.

가상세계에서도 클래스간에 특별한 관계를 맺으면서 처리가 된다. 이 관계를 'Class Relationship' 이라고 한다.

- 관계의 종류 2가지

### 가. has a 관계

한 객체가 다른 객체를 포함하는 관계이다. 전체(whole)와 부분(part)로 구성된다.

예> 트럭 has a 엔진, 트럭 has a 라디오

학생 has a 컴퓨터

### 나. is a 관계

같은 종(species)의 객체를 큰 개념과 작은 개념으로 설정한 관계이다.

예> 대학생 is a 학생, 관리자 is a 사원, 남자 is a 사람

### 가. 집합관계 ( aggregation relationship)

Whole과 part간의 LifeCycle이 서로 다르다.

예> 트럭 has a 라디오.



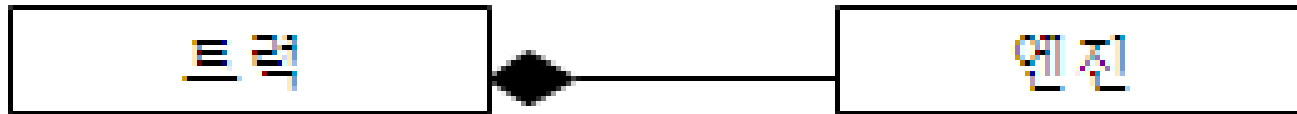
자바코드 표현방법은 다음과 같다.

```
public class 트럭{
    public void method(){
        라디오 radio; // 로컬변수 형태로 존재
    }
}
```

### 나. 구성관계 ( composition relationship)

Whole과 part간의 LifeCycle이 서로 일치한다.

예> 트럭 has a 엔진



자바코드 표현방법은 다음과 같다.

```
public class 트럭{
    엔진 engine; // 인스턴스 변수 형태로 존재
    public void method(){

    }
}
```

같은 종의 비슷한 속성 및 동작을 가진 객체들 간의 관계이다.

예를 들어 대학생, 중학생, 초등학생 객체들은 모두 공통된 개념의 학생 객체들이다. 개발자, 부장, 과장, 비서등은 모두 공통된 개념의 직원 객체들이다. 비슷한 개념들의 객체들은 모두 공통된 속성 및 동작을 가지고 있으며 이는 결국 공통점들이 중복되어 있다는 것이다.

따라서 공통점들을 추출해서 상위개념의 객체로 만들고, 하위 객체들에서는 상속 받아서 사용하게 하면 중복이 제거되고 재사용성도 향상될 수 있다.

이것이 자바의 ‘상속( inheritance)’ 개념이다.

자바의 상속을 적용시키기 위해서는 반드시 ‘is a’관계가 성립된 클래스들만 적용 가능하다.

예> 대학생 is a 학생, 부장 is a 사원, 남자 is a 사람

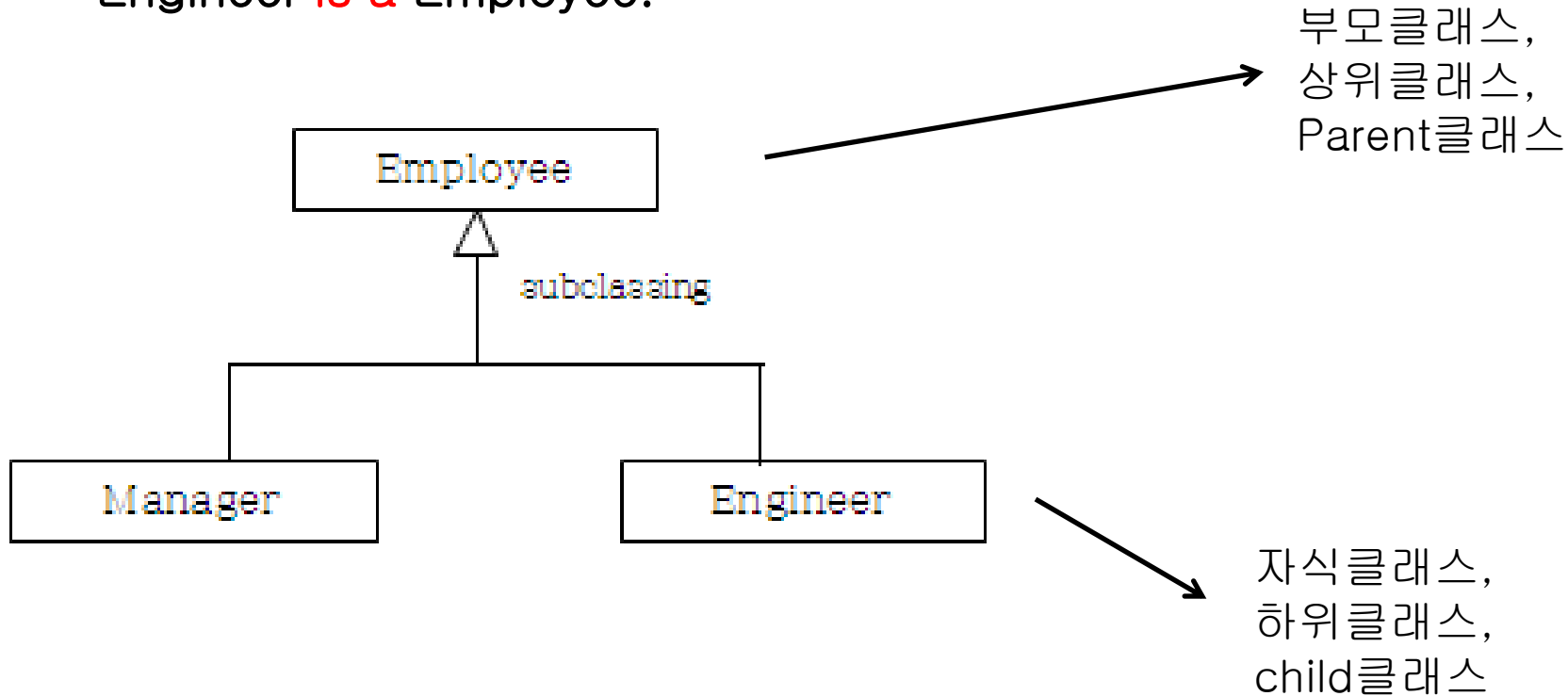
- 특징

- 반드시 객체간에 ‘is a’ 관계가 성립되어야 한다.  
즉 , 같은 종(종류)이어야 한다.
- 상속관계가 성립되면 하위 클래스에서는 부모 클래스의 멤버( 인스턴스 변수/ 메소드)를 선언없이 사용 가능하다.  
단, 생성자와 private로 지정된 멤버는 상속이 안된다.
- 자바는 단일 상속 ( single inheritance ) 만을 지원한다.
- extends 키워드를 사용하여 상속관계를 표현한다.
- 모든 클래스( 사용자 지정 클래스 및 API )는 계층구조인 상속관계로 되어있다.
- java.lang.Object 클래스가 모든 클래스의 최상위 클래스이다. 결국 Object 클래스의 멤버(인스턴스변수/메소드)는 모든 클래스에서 상속받아서 사용할 수 있다. extends 키워드로 지정하지 않은 클래스는 자동으로 extends Object가 추가된다.
- 문법:
  - public class 하위클래스 extends 상위클래스 { }



## □ 4) 상속 ( inheritance )

- Manager **is a** Employee.
- Engineer **is a** Employee.



- `public class Employee{}` => `public class Employee extends Object{}`
- `public class Manager extends Employee{}`
- `public class Engineer extends Employee{}`

- 생성자는 상속되지 않는다.
- 항상 모든 클래스는 부모 객체를 먼저 생성한 후에 자신이 생성된다. 따라서 항상 Object 클래스가 가장 먼저 생성되고 나머지 클래스들이 생성된다.
- 부모클래스를 생성하기 위해서는 부모 생성자가 호출되어야 된다. 자식 생성자 첫라인에서 부모생성자를 호출하는 코드가 자동으로 삽입이 되는데 형식은 다음과 같다.

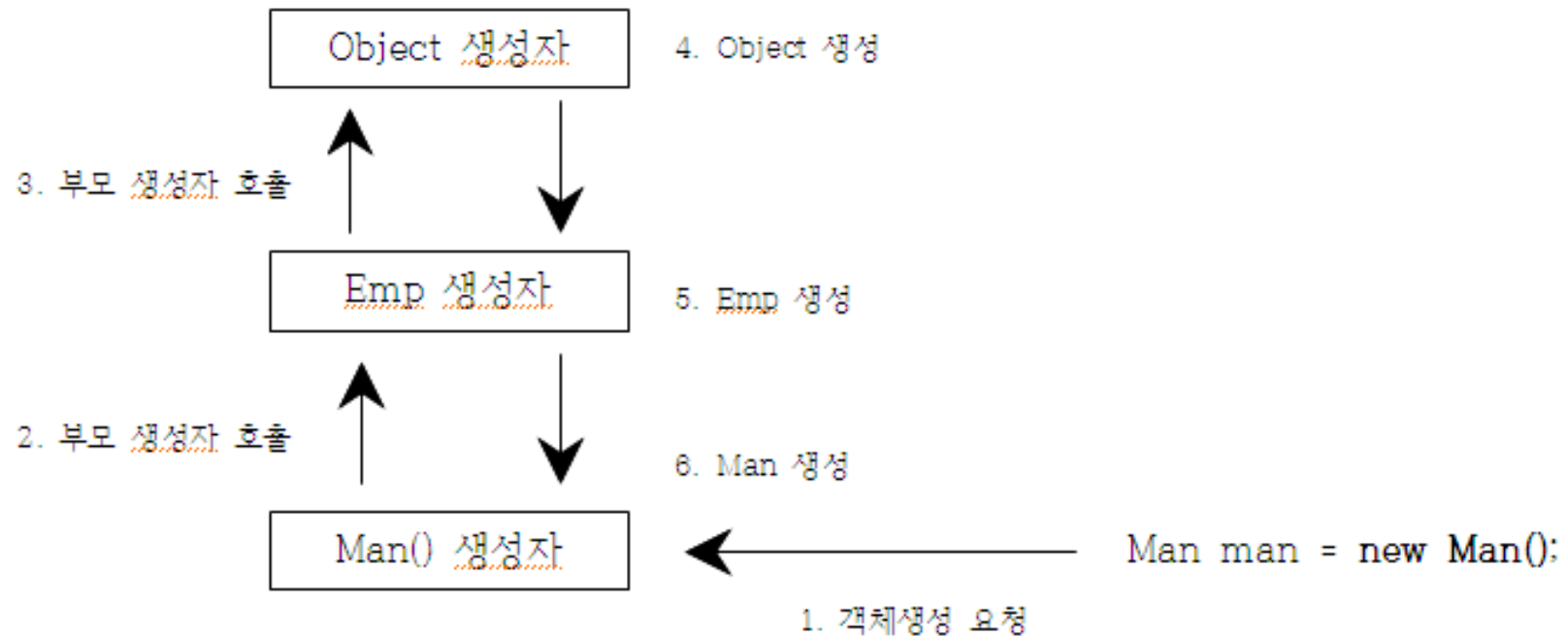
`super();` // 인자없는 부모생성자 호출하는 형태

- 필요에 의해서 명시적으로 부모 생성자를 호출할 수도 있다.

단, 반드시 생성자 첫라인에서 사용해야 된다.

```
예> super();  
    super("홍길동");  
    super("홍길동", 20 );
```

## □ 4) 상속과 생성자



super 키워드는 부모의 인스턴스를 가리킨다.  
(this 키워드는 자신의 인스턴스를 가리킨다.)

자식입장에서는 굳이 명시적으로 super 키워드를 사용하여 부모를 가리키지 않더라도 상속받기 때문에 부모의 멤버를 사용할 수 있다.  
하지만 명시적으로 super 키워드를 사용하여 부모를 가리키는 경우가 있다.

다음과 같은 2가지 가 대표적인 경우이다.

가. 부모 클래스의 멤버와 자식클래스의 멤버가 이름이 동일한 경우.

나. 자식클래스에서 명시적으로 부모 생성자를 호출하는 경우.

➔ 부모에서 선언된 변수인 경우에는 자식에서 초기화하지 않고  
super를 이용하여 부모에게 초기화하도록 유도할 수 있다. (권장)

클래스들은 서로간 상호작용을 통해서 동작된다.

클래스들간에 상호 작용시 접근을 제어하는 용도로 ‘접근 지정자’를 사용한다.

‘접근 지정자’를 사용함으로써 올바른 데이터를 설정할 수 있고, 또한 접근을 제한함으로써 접근대상에 관한 복잡성을 감소시킬수 있다.

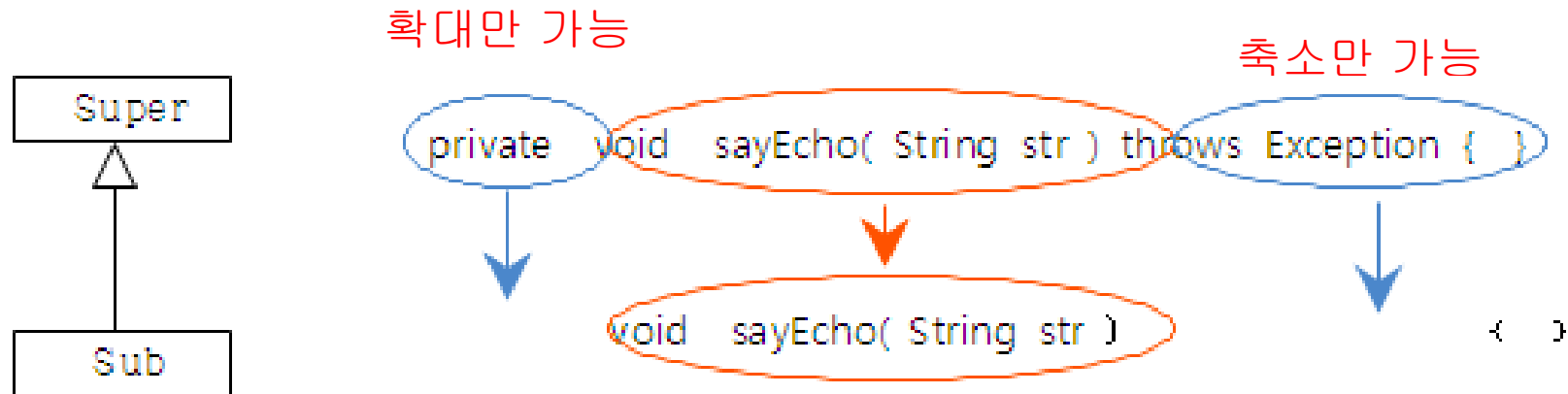
이 개념을 은닉화(encapsulation)라고 한다.

접근지정자	같은 클래스	같은 패키지	상속 관계	다른 패키지
public	가능	가능	가능	가능
protected	가능	가능	가능	불가
(friendly)	가능	가능	불가	불가
private	가능	불가	불가	불가

- 부모클래스의 메서드를 자식이 입맛에 맞게 수정할 수 있다.  
이렇게 부모클래스의 메서드를 자식이 수정한 메서드를 ‘오버라이딩 메소드 (overriding method)’라고 한다.

규칙:

- 상속전제
- 메소드 이름이 반드시 동일
- 메소드 리턴타입이 반드시 동일
- 메소드 인자리스트가 반드시 동일
- 접근지정자는 같거나 확대만 가능
- 예외클래스는 같거나 축소만 가능
- static , final, private로 지정된 메소드는 overriding 불가.**



- 오버로딩 메소드( overloading method)?  
같은 클래스내에 동일한 이름의 메소드가 여러 개 존재 가능.  
규칙은 반드시 인자리스트가 달라야 된다.

다형성( polymorphism )?

- 하나의 참조변수가 여러 다른 데이터형을 참조할 수 있는 능력.
- 상속이 전제되고 , 형변환도 가능하다.

- 예> // 다형성 미적용

```
Employee e = new Employee();
```

```
Manager m = new Manager();
```

```
Engineer eng = new Engineer();
```

//다형성 적용

```
Employee emp = new Employee();
```

```
emp = new Manager();
```

```
emp = new Engineer();
```



### \* 기본형 데이터 (정수형)

long            큰 타입  
int  
short  
byte            작은 타입

예>

```
int num = 10;  
long num2 = num;
```

### \* 참조형 데이터 (직원)

Employee            큰 타입  
  
Manager            Engineer    작은 타입

예>

```
Manager m = new Manger();  
Employee e = m;
```

- 다형성을 적용하는 용도는 다음 2가지 방법이 대표적이다.

### 가. 배열

일반적으로 배열은 같은 타입만을 저장할 수 있다. 하지만 다형성을 적용시키면 다른 타입도 저장 가능하다.

```
예> Object [] obj = { “홍길동”, new Person(), new Employee() };
```

```
Employee [] emp = { new Manager(“홍길동”),  
                    new Manager(“이순신”),  
                    new Engineer(“유관순”) };
```

```
for( Employee e : emp ){  
    if( e instanceof Manager){ ... }  
}
```


## □ 8] 다형성 [ polymorphism ]

나. 메소드 인자

예> 모든 직원의 세금을 구하려고 한다.

Employee는 10%, Manger는 20% , Engineer 는 5%

```
public class Employee{  
    public void taxRate( Employee emp){  
  
        if( emp instanceof Manager){  
            Manager m = (Manager)emp;  
            // 20%  
        }else if( emp instanceof Engineer ){  
            Engineer eng = (Engineer)emp;  
            // 5%  
        }else if ( emp instanceof Employee){  
            // 10%  
        }  
    }  
}
```



```
Manager m = new Manager();  
taxRate(m);  
  
Engineer eng = new Engineer();  
taxRate(eng);
```

- 동적 바인딩 ( Dynamic Binding )
  - 바인딩은 실제 실행할 메소드 코드와 호출하는 코드를 연결시키는 것을 의미한다. 프로그램이 실행되기 전에 바인딩이 일어날 경우 ‘정적 바인딩’이라고 한다.
  - 동적 바인딩은 실행할 당시의 객체 타입을 기준으로 바인딩되는 것을 의미한다. ( late 바인딩, 런타임 바인딩 이라고도 한다. )

컴파일러는 실행될 객체의 타입을 모르지만 메소드를 호출하는 메커니즘을 통해 실제 객체의 타입을 찾아서 적합한 메소드를 호출한다.
  - 동적 바인딩은 다형성에서 핵심적인 개념이다.

– 모든 클래스의 최상위 클래스이다.

명시적으로 extends 하지 않아도 자동으로 상속받는다.

Object 클래스의 메소드중에서 다음 2가지 메소드는 반드시 숙지하기 바람.

가. equals 메소드

참조변수가 참조하는 인스턴스의 실제값을 동등비교할 때 사용한다.

기본데이터는 == 연산자로 비교하면 된다.

```
예> int num = 3;  
    int num2 = 3;
```

```
if( num == num2 ){}
```

```
Manager m = new Manager("홍길동");  
Manager m2 = new Manager("홍길동");
```

```
if( m.equals(m2) ){ } // 실제값 비교
```

```
if( m == m2 ) { } //주소값 비교
```

Object의 equals 메소드는 내부적으로 == 연산자로 동등 비교하게 구현되어 있다. 따라서 실제값을 비교할 목적으로 equals 메소드를 사용해도 원하는 true값이 아닌 false값이 나온다.

이것을 해결하기 위해서 equals 메소드를 실제값 비교하게 Overriding 해야 된다.

대부분의 API는 Object 클래스의 equals 메소드를 실제값 비교로 처리하게 Overriding 되어 있다.

가장 많이 사용하는 클래스는 String 클래스이다. ( 중요 )

```
예> String name = "홍길동";  
      String name2 = "홍길동";
```

```
if( name.equals( name2 ) ){ }
```

### 나. toString 메소드

참조변수가 참조하는 인스턴스 의 위치값을 문자열로 변경시키는 메소드이다.  
이 메소드는 참조변수를 println(참조변수) 할때 자동으로 호출된다.  
따라서 다음 두 문장은 동일하다.

```
예> Person p = new Person( "홍길동" );  
      System.out.println( p );  
      System.out.println( p.toString() );
```

대부분의 API는 모두 toString 메소드를 Overriding 해서 사용한다.

용도: 디버깅용으로 사용한다.

( 재사용 클래스의 인스턴스 변수가 제대로 설정되었는지 확인하거나 한꺼번에 특정 포맷으로 출력시 사용됨 )



**Thank you**

---