

**상속**  
**(Inherit)**

## ▶ 상속

다른 클래스가 가지고 있는 멤버(필드, 메소드)들을 새로 작성할 클래스에서 직접 만들지 않고 상속을 받음으로써 새 클래스가 자신의 멤버처럼 사용할 수 있는 기능

### ✓ 상속의 목적

클래스의 재사용, 연관된 일련의 클래스들에 대한 공통적인 규약 정의

### ✓ 상속의 장점

1. 보다 적은 양의 코드로 새로운 클래스 작성 가능
2. 코드를 공통적으로 관리하기 때문에 코드의 추가 및 변경 용이
3. 코드의 중복을 제거하여 프로그램의 생산성과 유지보수에 크게 기여

# ▶ 상속의 특징

## 1. 모든 클래스는 Object클래스의 후손

Object클래스가 제공하는 메소드를 오버라이딩하여 메소드 재구현 가능  
ex) java.lang.String 클래스의 equals()와 toString()

## 2. 부모클래스의 생성자, 초기화 블록은 상속 안 됨

자식 클래스 생성 시, 부모 클래스 생성자가 먼저 실행

자식 클래스 생성자 안에서 부모 클래스 생성자 호출을 명시하고 싶으면 super() 활용

## 3. 부모의 private멤버는 상속은 되지만 직접 접근 불가

자식 객체 생성 시에 부모의 필드 값도 전달 받은 경우,

자식 생성자 안에서 부모의 private 필드에 직접 접근하여 대입 불가

super() 이용하여 전달받은 부모 필드 값을 부모 생성자 쪽으로 넘겨서 생성하거나  
setter, getter 메소드를 이용하여 접근

## ▶ 상속

### ✓ 방법

클래스 간의 상속 시에는 extends 키워드 사용

### ✓ 표현식

[접근제한자] class 클래스명 **extends** 클래스명 {}

**public class** Academy **extends** Company {}

# 상속의 가장 기본적인 특성

---

```
class Man {  
    String name;  
    public void tellYourName() {  
        System.out.println("My name is " + name);  
    }  
}
```

```
class BusinessMan extends Man {  
    String company;  
    String position;  
    public void tellYourInfo() {  
        System.out.println("My company is " + company);  
        System.out.println("My position is " + position);  
        tellYourName();  
    }  
}
```

```
BusinessMan man = new BusinessMan( );
```

man  
참조변수

String name : Man의 멤버  
String company;  
String position;  
void tellYourName( ) {..} : Man의 멤버  
void tellYourInfo( ) {..}

BusinessMan 인스턴스

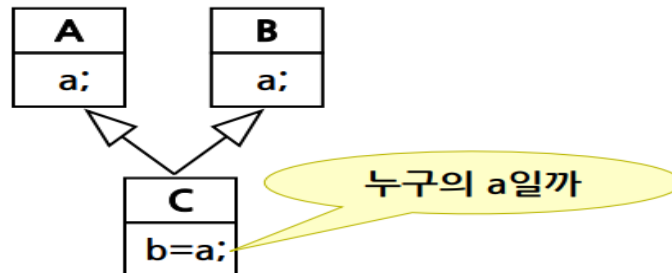
## ▶ 단일 상속과 다중 상속

### ✓ 단일 상속(Single Inheritance)

클래스간의 관계가 다중 상속보다 명확하고 신뢰성 있는 코드 작성  
자바에서는 다중 상속 미지원 → **단일상속만 지원**

### ✓ 다중 상속(Multiple Inheritance)

C++에서 가능한 기능으로 여러 클래스로부터 상속을 받으며  
복합적인 기능을 가진 클래스를 쉽게 작성 가능  
서로 다른 클래스로부터 상속 받은 멤버 간의 이름이 같은 경우 문제 발생



# 상속과 생성자1

---

```
class Man {  
    String name;  
  
    public Man(String name) {  
        this.name = name;  
    }  
  
    public void tellYourName() {  
        System.out.println("My name is " + name);  
    }  
}
```

BusinessMan 인스턴스 생성시 문제점은?

```
class BusinessMan extends Man {  
    String company;  
    String position;  
  
    public BusinessMan(String company, String position) {  
        this.company = company;  
        this.position = position;  
    }  
  
    public void tellYourInfo() {  
        System.out.println("My company is " + company);  
        System.out.println("My position is " + position);  
        tellYourName();  
    }  
}
```

# 상속과 생성자2

---

```
class BusinessMan extends Man {
    String company;
    String position;

    public BusinessMan(String name, String company, String position) {
        // 상위 클래스 Man의 멤버 초기화
        this.name = name;

        // 클래스 BusinessMan의 멤버 초기화
        this.company = company;
        this.position = position;
    }

    public void tellYourInfo() { . . . }
}
```

```
class Man {
    String name;

    public Man(String name) {
        this.name = name;
    }
    . . .
}
```

모든 멤버의 초기화는 이루어진다. 그러나  
생성자를 통한 초기화 원칙에는 어긋남!

```
BusinessMan man =
    new BusinessMan("YOON", "Hybrid ELD", "Staff Eng.");
```



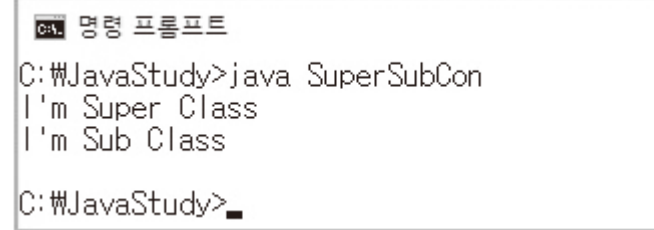
# 상속과 생성자3: 생성자 호출 관계 파악하기

---

```
class SuperCLS {  
    public SuperCLS() {  
        System.out.println("I'm Super Class");  
    }  
}
```

```
class SubCLS extends SuperCLS {  
    public SubCLS() {  
        System.out.println("I'm Sub Class");  
    }  
}
```

```
class SuperSubCon {  
    public static void main(String[] args) {  
        new SubCLS();  
    }  
}
```



```
명령 프롬프트  
C:\JavaStudy>java SuperSubCon  
I'm Super Class  
I'm Sub Class  
C:\JavaStudy>
```

상위 클래스의 생성자 실행 후  
하위 클래스의 생성자 실행 됨

호출할 상위 클래스의 생성자 명시하지 않으면 void 생성자 호출 됨

## ▶ **super()와 super.**

### ✓ **super()**

부모 객체의 생성자를 호출하는 메소드로 기본적으로 후손 생성자에 부모 생성자 포함 후손 객체 생성 시에는 부모부터 생성이 되기 때문에 후손 클래스 생성자 안에는 부모 생성자를 호출하는 `super()`가 첫 줄에 존재 (부모 생성자가 가장 먼저 실행되어야 하기 때문에 명시적으로 작성 시에도 반드시 첫 줄에만 작성)  
매개변수 있는 부모 생성자 호출은 `super(매개변수, 매개변수)`를 넣으면 됨

### ✓ **super.**

상속을 통한 자식 클래스 정의 시 해당 자식 클래스의 부모 객체를 가리키는 참조변수 자식 클래스 내에서 부모 클래스 객체에 접근하여 필드나 메소드 호출 시 사용

# 상속과 생성자4: 상위 클래스의 생성자 호출 지정

---

```
class SuperCLS {  
    public SuperCLS() {  
        System.out.println("...");  
    }  
  
    public SuperCLS(int i) {  
        System.out.println("...");  
    }  
  
    public SuperCLS(int i, int j) {  
        System.out.println("...");  
    }  
}
```

```
class SubCLS extends SuperCLS {  
    public SubCLS() {  
        System.out.println("...");  
    }  
  
    public SubCLS(int i) {  
        super(i);  
        System.out.println("...");  
    }  
  
    public SubCLS(int i, int j) {  
        super(i, j);  
        System.out.println("...");  
    }  
}
```

키워드 `super`를 통해 상위 클래스의 생성자 호출을 명시할 수 있음

# 적절한 생성자 정의의 예

---

```
class Man {
    String name;

    public Man(String name) {
        this.name = name;
    }
    public void tellYourName() {
        System.out.println("My name is " + name);
    }
}

class BusinessMan extends Man {
    String company;
    String position;

    public BusinessMan(String name, String company, String position) {
        super(name);
        this.company = company;
        this.position = position;
    }
    public void tellYourInfo() {
        System.out.println("My company is " + company);
        System.out.println("My position is " + position);
        tellYourName();
    }
}
```

상속이 도움이 되는 상황의 예

# 단순한 인맥 관리 프로그램: 관리 대상이 둘!

---

```
class UnivFriend {    // 대학 동창
    private String name;
    private String major;    // 전공
    private String phone;

    public UnivFriend
        (String na, String ma, String ph) {
        name = na;
        major = ma;
        phone = ph;
    }
    public void showInfo() {
        System.out.println("이름: " + name);
        System.out.println("전공: " + major);
        System.out.println("전화: " + phone);
    }
}
```

대학 동창      이름, 전공, 전화번호 정보 저장 및 관리

직장 동료      이름, 부서, 전화번호 정보 저장 및 관리

```
class CompFriend {    // 직장 동료
    private String name;
    private String department;    // 부서
    private String phone;

    public CompFriend
        (String na, String de, String ph) {
        name = na;
        department = de;
        phone = ph;
    }
    public void showInfo() {
        System.out.println("이름: " + name);
        System.out.println("부서: " + department);
        System.out.println("전화: " + phone);
    }
}
```

관리 대상이 둘이므로  
두 개의 클래스가 정의되었다.

# 두 클래스를 대상을 하는 코드

---

```
public static void main(String[] args) {
    UnivFriend[] ufrns = new UnivFriend[5];
    int ucnt = 0;

    CompFriend[] cfrns = new CompFriend[5];
    int ccnt = 0;

    ufrns[ucnt++] = new UnivFriend("LEE", "Computer", "010-333-555");
    ufrns[ucnt++] = new UnivFriend("SEO", "Electronics", "010-222-444");

    cfrns[ccnt++] = new CompFriend("YOON", "R&D 1", "02-123-999");
    cfrns[ccnt++] = new CompFriend("PARK", "R&D 2", "02-321-777");

    for(int i = 0; i < ucnt; i++) {
        ufrns[i].showInfo();
        System.out.println();
    }

    for(int i = 0; i < ccnt; i++) {
        cfrns[i].showInfo();
        System.out.println();
    }
}
```

■ 대학 동창 관련 코드

■ 직장 동료 관련 코드

이러한 클래스 디자인 기반에서 관리 대상이 넷, 다섯으로 늘어 난다면? 늘어나는 수 만큼 코드 복잡해짐

# 상속 기반의 문제 해결: 두 클래스 상속 관계로 묶기

---

```
class Friend {  
    protected String name;  
    protected String phone;  
  
    public Friend(String na, String ph) {  
        name = na;  
        phone = ph;  
    }  
    public void showInfo() {  
        System.out.println("이름: " + name);  
        System.out.println("전화: " + phone);  
    }  
}
```

“연관된 일련의 클래스들에 대해  
공통적인 규약을 정의 및 적용할 수 있습니다.”

“CompFriend와 UnivFriend 클래스에 대해  
Friend 클래스라는 규약을 정의하고 적용할 수 있습니다.”

```
class CompFriend extends Friend {  
    private String department;  
  
    public CompFriend(String na, String de, String ph) {  
        super(na, ph);  
        department = de;  
    }  
    public void showInfo() {  
        super.showInfo();  
        System.out.println("부서: " + department);  
    }  
}  
  
class UnivFriend extends Friend {  
    private String major;  
  
    public UnivFriend(String na, String ma, String ph) {  
        super(na, ph);  
        major = ma;  
    }  
    public void showInfo() {  
        super.showInfo();  
        System.out.println("전공: " + major);  
    }  
}
```



# 상속으로 묶은 결과

---

```
public static void main(String[] args) {  
    Friend[] frns = new Friend[10];  
    int cnt = 0;  
  
    frns[cnt++] = new UnivFriend("LEE", "Computer", "010-333-555");  
    frns[cnt++] = new UnivFriend("SEO", "Electronics", "010-222-444");  
    frns[cnt++] = new CompFriend("YOON", "R&D 1", "02-123-999");  
    frns[cnt++] = new CompFriend("PARK", "R&D 2", "02-321-777");  
  
    // 모든 동창 및 동료의 정보 전체 출력  
    for(int i = 0; i < cnt; i++) {  
        frns[i].showInfo();          // 오버라이딩 한 메소드가 호출된다.  
        System.out.println();  
    }  
}
```

이러한 클래스 디자인 기반에서 관리 대상이 넷, 다섯으로 늘어 난다면? 인스턴스 관리와 관련해서 코드가 복잡해지지 않는다.

static변수,  
static 메소드와 상속

# 클래스 변수, 메소드는 상속이 되는가?

---

```
class SuperCLS {  
    static int count = 0;    // 클래스 변수  
  
    public SuperCLS() {  
        count++;    // 클래스 내에서는 직접 접근이 가능  
    }  
}
```

} 프로그램 전체에서 딱 하나만 존재하는데 상속의 대상이 되겠는가?

```
class SubCLS extends SuperCLS {  
    public void showCount() {  
        System.out.println(count);    // 상위 클래스에 위치하는 클래스 변수에 접근  
    }  
}
```

} 그러나 하위 클래스에서 이름만으로 접근 가능하다!  
접근 수준 지시자에서 허용한다면!