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 {}
```

▶ 단일 상속과 다중 상속

✓ 단일 상속(Single Inheritance)

클래스간의 관계가 다중 상속보다 명확하고 신뢰성 있는 코드 작성

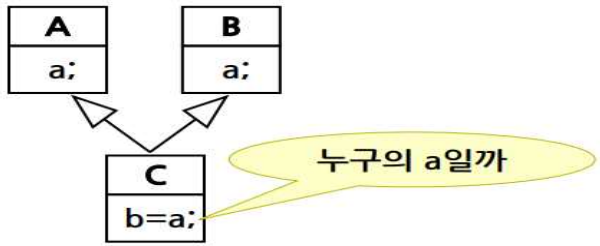
자바에서는 다중 상속 미지원 → **단일상속만 지원**

✓ 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



▶ 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

▶ Is a VS Has a

✓ Is a (상속관계)

"자식클래스는 (하나의) 부모 클래스이다. " 관계로, 부모 클래스를 자식 클래스가 상속한다.

```
public class Shape {
    private double area;
    public void calcArea() {
        // 넓이 구하기 메소드
    }
}
```

```
public class Circle extends Shape {
    private int x;  // 중심점의 x좌표
    private int y;  // 중심점의 y 좌표
    private int r;  // 반지름 Radius

    calcArea();
}
```

Circle **is a** Shape.

Circle 클래스**는** 하나의 Shape 클래스**이다.** (Is a 상속 관계)

▶ Is a VS Has a

일반화 관계 >> 상속관계
연관 관계 >> 포함관계

✓ Has a (포함관계)

한 클래스의 멤버변수로 다른 클래스 타입의 참조변수를 선언함

```
public class Point {
    private int x;  // 중심점의 x좌표
    private int y;  // 중심점의 y좌표
}
```

```
public class Circle {
    private Point center = new Point();
    private int r;  // 반지름 Radius
}
```

Circle **has a** Point

Circle 클래스**는** Point 클래스를 **가지고 있다.** (has a 포함 관계)

▶Is a VS Has a

✓ 다음 클래스의 관계를 정의해보자

```
public class Shape {
    private double area;
    public void calcArea() {
        // 넓이 구하기 메소드
    }
}
```

Is a

```
public class Circle extends Shape {  priv
    ate Point center = new Point();  pri
    vate int r;      // 반지름 Radius
}
```

Has a

```
public class Point {
    private int x;  // 중심점의 x좌표
    private int y;  // 중심점의 y좌표
}
```

Circle **is a** Shape. → Circle 클래스**는** 하나의 Shape 클래스**이다.**

Circle **has a** Point. → Circle 클래스**는** Point 클래스를 **가지고 있다.**

▶ 오버라이딩(Overriding)

자식 클래스가 상속 받은 부모 메소드를 재작성 하는 것
부모가 제공하는 기능을 후손이 일부 고쳐 사용하겠다는 의미로
자식 객체를 통한 실행 시 후손 것이 우선권을 가짐

✓ 특징

메소드 헤드라인 위에 반드시 Annotation, @Override 표시
접근 제어자를 부모 것보다 같거나 넓은 범위로 변경 가능
부모 메소드의 예외처리 클래스 처리범위보다 좁은 범위로 예외처리 클래스 수정 가능

▶ 오버라이딩(Overriding)

✓ 성립 조건

부모 클래스의 메소드와 자식 클래스의 메소드 비교

- 메소드 이름 동일
- 매개변수의 개수, 타입 동일
- 리턴 타입 동일
- private 메소드 오버라이딩 불가

▶ 오버로딩(Overloading)

한 클래스 내에서 같은 이름의 메소드를 여러 개 정의하는 것

✓ 성립 조건

같은 메소드 이름

다른 매개변수 선언부(매개변수 타입, 개수, 순서)

✓ 주의 사항

메소드의 리턴타입은 오버로딩 조건과 관계 없음

▶ 오버라이딩(Overriding)과 오버로딩(Overloading)

오버라이딩(Overriding)	오버로딩(Overloading)
하위 클래스에서 메소드 정의	같은 클래스에서 메소드 정의
메소드 이름 동일 매개변수 동일(개수, 타입) 리턴 타입 동일	메소드 이름 동일 매개변수 다름(개수, 타입) 리턴 타입 상관 없음
자식 메소드의 접근 범위가 부모 메소드의 접근 범위보다 넓거나 같아야 함	접근 제어자와 상관 없음
자식 메소드의 예외 수가 부모 메소드의 예외 수보다 적거나 범위가 좁아야 함	예외처리와 상관 없음

▶ final 예약어

✓ final 클래스

상속이 불가능한 클래스

```
public final class FinalClass {}
```

✓ final 메소드

상속 시 오버라이딩이 불가능한 메소드

```
public final void method() {}
```

▶ 대상에 따른 사용 가능한 제어자, 예약어

대상	사용 가능한 제어자/예약어
클래스	public, (default), final, abstract
메소드	모든 접근 제어자, final, abstract, static
변수	모든 접근 제어자, final, static
지역변수	final

✓ 유의 사항

클래스에 abstrac와 final 동시에 사용 불가능

메소드에 static과 abstract 동시에 사용 불가능

abstract 메소드의 접근제어자로 private 불가능