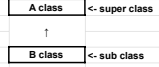


	<p>*생성자 호출 순서</p> <pre> graph TD Object --> Aclass[A class] Aclass --> Bclass[B class] Bclass --> Cclass[C class] </pre> <p>Object { - - } ↑ 아무것도 없으면 Object를 상속한다. 모든 클래스 생성자는 슈퍼 클래스 생성자를 호출해야 한다.</p> <p>A class { - v1 - A() { - - } super() <== 생성자의 "첫 문장"은 반드시 슈퍼클래스의 생성자 호출 문장이어야 한다. 예) super() // this() <= 둘다 동시에 호출할 수 없다. B class { - v2 - B() { - - } C class { - v3 - C() { - - } => 슈퍼클래스의 생성자 호출을 생략하면 컴파일러가 자동으로 추가한다.</p> <p>C obj = new C (); ==> C의 생성자 호출</p> <pre> graph LR Object[Object { super() }] -- 4 --> Aclass[A class { A() { super() } }] Aclass -- 3 --> Bclass[B class { B() { super() } }] Bclass -- 2 --> Cclass[C class { C() { super() } }] Cclass -- 1. 호출 --> 리턴 </pre>
	<p>*생성자 호출 2</p> <pre> graph TD Aclass[A class { v1 - A(int) B class { v2 - B() </pre> <p>B obj = new B(); ==> B의 기본 생성자 호출</p>
	<p>*다중 상속</p> <pre> graph TD Aclass[A class { v1 : int v2 : float b class { v2 : int v3 : int </pre> <p>C obj = new C ();</p> <p>v1 v2 v2 v3 A A B C</p> <p>obj.v2 = ?; A의 변수인지 B의 변수인지 구분할 수 없다. => 구분하려면 문법을 추가 => 언어가 복잡 => 컴파일러가 복잡 ↓ 그래서 자바는 다중상속을 허락하지 않는다.</p>
	<p>*상속 : specialization(전문화)</p> <pre> graph TD Car[Car { model maker capacity run Sedan[Sedan { run() { - } doSunroof() { - } Truck[Truck { run() { - } dump() { - } </pre> <p><- 처음에 직접 사용하려고 만든 클래스이다. <- 필요에 의해 추가로 정의한 클래스이다.</p> <p>상속받은 메서드를 서브클래스에 맞춰 재정의한다 = 오버라이딩 << 슈퍼클래스의 메서드를 현재 클래스의 역할에 맞게 재정의한다. ="Overriding" (오버라이딩) << 승용차에 맞는 기능을 덧붙인다.</p> <p>"더 특별한 자동차를 만든다 = specialization</p> <p><<트럭의 특성에 맞게 재정의 = Overriding <<트럭만의 고유기능</p>
	<p>*상속 : Generalization(일반화)</p> <pre> graph TD Car[Car { start() shutdown() run Sedan[Sedan { start() shutdown() stop() run() go-> doSunroof() Truck[Truck { launch() stop()-> go-> dump() run 메서드 재정의 : overriding run 메서드 재정의 : overriding </pre> <p><< 다른 자동차를 정의할 때 재사용 가능 : generalization 추출해서 일반적인 기능으로 만든다.</p> <p>자동차의 일반적인 기능 : Generalization (일반화)</p>

클래스 관계 cheat sheet

1. 상속(inheritance)



class B extend A { ~ }

2. 연관(association)



class B {
A obj;
}

3. 집합(aggregation)



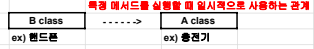
class B {
A obj;
}
Lifecycle
컴퓨터 ≠ 키보드
≠ 마우스
≠ 모니터

4. 합성(composition)



class B {
A obj;
}
Lifecycle
컴퓨터 = 그래픽카드
= ROM
= CPU

5. 의존(dependency)



class B {
void m(A obj){
}

특정 메서드를 실행할 때 일시적으로 사용하는 관계

