상속 : 실제 부모/자식 관계가 있는 다른 클래스의 특성과 행위를 상속
(is-a)

합성: 객체들을 사용해 또 다른 객체를 작성하는 일까지 포함
(has-a)

상속과 합성이 존재하는 주된 이유는 객체의 재사용

상속

상속을 이용하면 자식 클래스가 부모 클래스의 행위를 물려 받기 때문에 설계 및 코딩 시간과 테스트 시간을 절약할 수 있게 해준다.
또한 상속된 코드는 부모 클래스에서 수정하면 자식 클래스에서는 수정할 필요가 없어 유지보수가 간편하다.

하지만 상속을 사용시 자식 클래스에서 사용 불가능한 부모 클래스의 메서드를 상속 받을 수도 있다.

ex) 부모 클래스 bird의 fly는
자식 클래스 펭귄에서 호출될 수 있다.
하지만 펭귄은 날 수 없으며 override 하더라도
메서드의 이름은 fly 이며 사용해도 의미가 없다.

일반화와 특수화

상속 트리를 내려갈수록 특수화(구체화)되며
올라갈수록 일반화 된다.

설계 결정

이론적으로는 가능한 한 많은 공통성을 고려하는 편이 바람직하다.
하지만 이렇게 하면 복잡성이 늘어난다.
항상 미래를 염두에 두고 설계를 결정한다

합성 : 어떤 객체들은 그 밖의 객체들로 이루어져 있다.

캡슐화 : 클래스에서 무엇을 노출해야 하고
무엇을 노출해서는 안되는가.

상속이 캡슐화를 약화시키는 방법
: 상속으로 인해 그 밖에 클래스에 대해서는 강력하게
캡슐화되는 꼴이 되지만 슈퍼 클래스와 서브 클래스 사이의
캡슐화가 약해지는 경우를 말한다.

결론

이번 장에서는 상속과 합성이 무엇이고 서로 어떻게 다른지를 간략하게 살펴보았다. 존경받는 객체지향 설계자들은 될 수 있으면 기본적으로 합성을 사용하면서 필요할 때만 상속을 사용해야 한다고 했다.

그러나 이는 조금은 단순해 보이는 생각이다. 필자는 될 수 있으면 합성을 사용해야 한다고 생각하는 방식에는 현실적인 문제가 숨겨져 있다고 생각한다. 합성을 상속보다 많이 써야만 하는 경우라면 그런 생각이 더 적절할 수도 있다. 그런 현실을 고려한다고 해도 무조건 합성만 사용해야 하는 건 아니다. 합성을 쓰는 편이 적절한 경우가 더 많다고 해서 상속이 나쁘다고만은 말할 수 없다. 그러므로 합성과 상속을 상황에 맞게 적절히 섞어 쓰도록 하자.

이전에 나온 여러 장에서는 추상 클래스와 자바 인터페이스라는 개념을 여러 번 다뤘다. 8장에서는 개발 계약이라는 개념과 이러한 계약을 충족시키기 위해 추상 클래스와 자바 인터페이스가 어떻게 사용되는지를 살펴본다.