


## 상속을 생각하기 조는 상속을 생각해 놓는다 ☆


quack() 메소드에서  
이미 했던 것처럼 fly() 메소드도  
그냥 오버라이드하면 되는 거였잖아.




**RubberDuck**

```
quack() { // 뽕뽕 }
display() { // 고무 오리 }
fly() {
  // 아무것도 하지 않도록 오버라이드
}
```

프로그램에 나무로 된 가짜 오리를  
추가하려면 어떻게 하지?  
날 수도 소리 낼 수도 없어야 할 텐데...





**DecoyDuck**

```
quack() {
  // 아무것도 하지 않도록 오버라이드
}

display() { // 가짜 오리 }

fly() {
  // 아무것도 하지 않도록 오버라이드
}
```

새로운 클래스를 추가했습니다.  
 RubberDuck과 마찬가지로  
 날 수 없는 데다가  
 아무 소리도 내지 못합니다.



**쓰면서 제대로 공부하기** Q. 문제 B는 왜 체크임? → 실제 작업에 나쁜 영향을 바꿀 수 X.

정답 071쪽

다음 중 Duck의 행동을 상속할 때 단점이 될 수 있는 요소를 모두 고르시오.

- |  |   |
|--|---|
| <p><input checked="" type="checkbox"/> A. 서브클래스에서 코드가 중복된다.</p> <p><input checked="" type="checkbox"/> B. 실행 시에 특징을 바꾸기 힘들다.</p> <p><input type="checkbox"/> C. 오리가 춤추게 만들 수 없다.</p> | <p><input checked="" type="checkbox"/> D. 모든 오리의 행동을 알기 힘들다.</p> <p><input type="checkbox"/> E. 오리가 날면서 동시에 꺽꺽거릴 수 없다.</p> <p><input checked="" type="checkbox"/> F. 코드를 변경했을 때 다른 오리들에게 원치 않은 영향을 끼칠 수 있다.</p> |
|--|---|

# 소프트웨어 개발 불변의 진리

모든 변화는 언제나 우리와 함께 합니다 ☆

소프트웨어 개발에서 절대로 바뀌지 않는 진리는 무엇일까요?

어디에서든 어떤 프로그래밍 언어를 쓰든 무엇을 만들든 항상 여러분을 따라다니는 진리는 무엇일까요?

## 변화

(거울에 비춰 보면 답을 찾을 수 있습니다)

아무리 디자인을 잘한 애플리케이션이라도 시간이 지남에 따라 변화하고 성장해야 합니다.  
그렇지 않으면 그 애플리케이션은 죽고 맙니다.



### 쓰면서 제대로 공부하기

정답 071쪽

변화의 원인은 수없이 많습니다. 여러분이 애플리케이션을 만드는 과정에서 코드를 바꿔야 했던 이유를 적어 보세요(일단 2가지 이유를 적어 놓았습니다).

- 1 고객이나 사용자가 다른 것을 요구하거나 새로운 기능을 원할 때
- 2 회사에서 데이터베이스 종류를 바꾸고 데이터도 전과 다른 데서 구입하기로 했는데, 그게 지금 사용하는 데이터 포맷과 완전히 다른 경우.  
생각만 해도 골치 아프네요
- 3 새로운 개발자가 팀에 가입해서 코드를 이해하려고 할 때.
- 4
- 5
- 6
- 7



## 무엇이든 물어보세요 Q&A

**Q1** 매번 애플리케이션을 구현하고 바뀔 수 있는 부분을 찾아낸 후에 바뀌는 것과 바뀌지 않는 것을 분리해서 캡슐화하는 식으로 작업해야 하나요?

**A1** 언제나 그렇게 해야 하는 것은 아닙니다. 애플리케이션을 디자인하는 과정에서 바뀔 수 있는 부분을 예측하고 대처해서 유연한 코드를 만들 수도 있습니다. 여기에서 설명하는 원칙과 패턴은 개발 라이프사이클의 어느 단계에서든지 적용할 수 있습니다.

**Q2** Duck 인터페이스도 만들어야 하나요?

**A2** 이 경우에는 그렇게 할 필요가 없습니다. 지금 살펴보고 있는 예제가 끝나면 알 수 있겠지만, Duck을 구상 클래스로 만들고 **MallardDuck** 같은 특정 오리 클래스를 만들 때 공통 속성과 메소드를 상속함으로써

얻는 장점도 있으니까요. 이제 Duck 상속 과정에서 바뀔 수 있는 부분을 제거했으므로 별문제 없이 장점을 취할 수 있습니다.

**Q3** 행동만 나타내는 클래스를 만든다는 게 좀 이상하게 느껴지네요. 클래스는 원래 어떤 대상을 나타내는 것 아닌가요? 클래스에는 상태와 행동이 모두 들어있어야 하지 않나요?

**A3** 객체지향 시스템에서는 질문한 내용이 맞습니다. 클래스는 일반적으로 상태(인스턴스 변수)와 메소드를 둘 다 가지고 있습니다. 그런데 이 경우에는 클래스가 '행동'을 가지고 있습니다. 하지만 행동에도 여전히 상태와 메소드가 들어있을 수 있습니다. 나는 행동에 속성(1분당 날개를 펄럭이는 횟수라든가 최고 높이, 속도 등)을 나타내는 인스턴스 변수를 넣을 수도 있으니까요.



## 쓰면서 제대로 공부하기

**①** 앞쪽에 나온 디자인을 활용해서 SimUDuck에 로켓의 추진력으로 날아가는 행동을 추가해야 한다면 어떻게 해야 할까요?

*flyBehaviour 구현체를 하나 추가하면 된다*

**②** 오리 클래스가 아닌 다른 클래스에서 Quack을 활용할 방법이 있는지 한번 생각해 봅시다.

*다른 클래스에서 Quack Behaviour를 활용하면 된다..*

*뭔... Duck 이 아닌..  
Penguin?*

### 정답

- 1 FlyBehavior 인터페이스를 구현하는 FlyRocketPowered 클래스를 만들면 되겠죠
- 2 오리 호출기(오리 소리를 내는 장치)를 만들 때 활용할 수 있겠죠?

## 캡슐화된 행동 살펴보기

캡슐화된 행동을 큰 그림으로 바라봅시다 ☆

지금까지 오리 시뮬레이터 디자인을 깊이 파헤쳐 왔습니다.

이제 잠깐 쉬면서 큰 그림을 한번 살펴보죠.

아래에 새롭게 구성한 클래스 구조가 있습니다. 지금까지 했던 일들이 정리되어 있습니다.

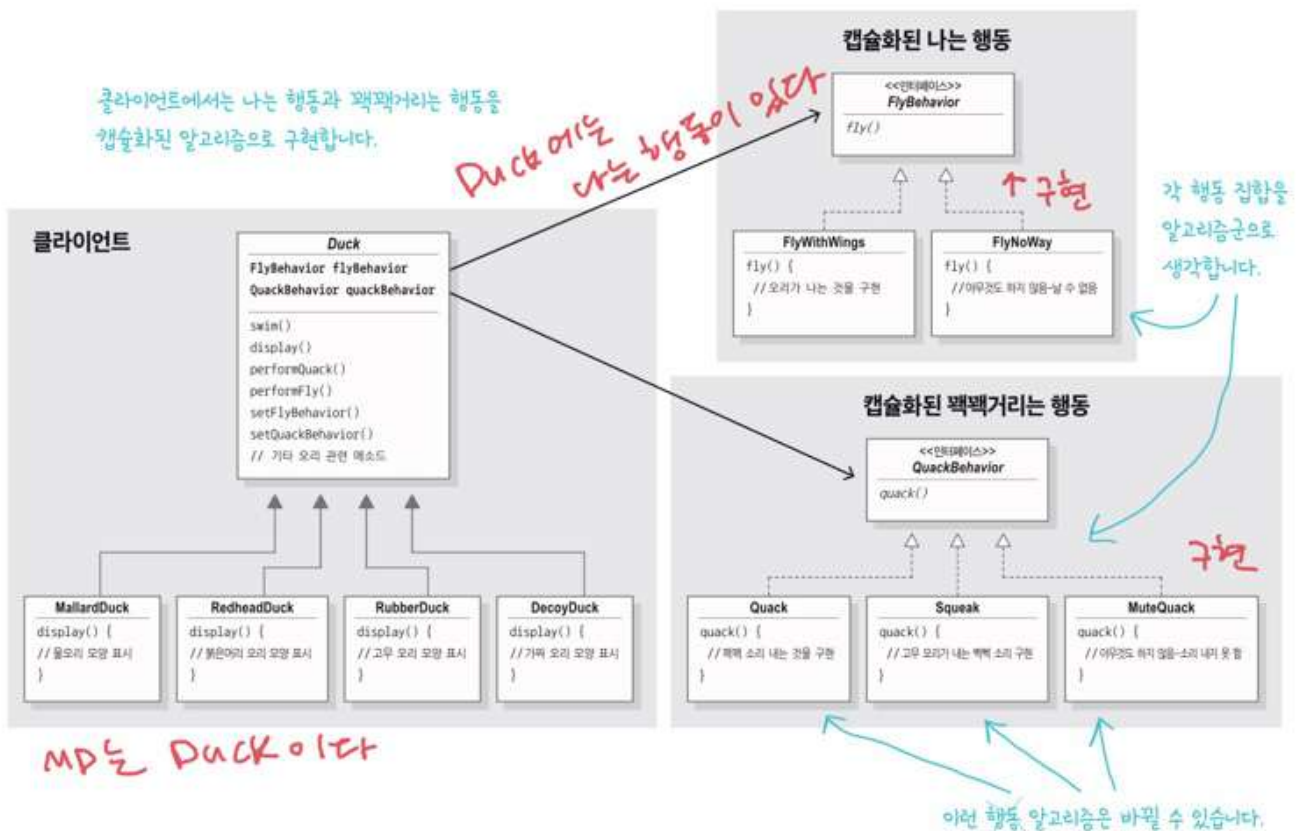
오리들은 모두 Duck을 확장해서 만들고, 나는 행동은 FlyBehavior를, 깹꾹거리는 행동은 QuackBehavior를 구현해서 만듭니다.

그리고 이번에는 생각하는 방식을 조금 바꿔 봤습니다. **오리의 행동들을 일련의 행동으로 생**

**각하는 대신, 알고리즘군(family of algorithms)으로 생각하는 거죠.** SimUDuck 디자인에 서 알고리즘은 오리가 하는 행동(다른 방식으로 깹꾹거리고 나는 행동)을 나타내지만, 지역 에 따라 달라지는 **세금 계산 방식을 구현하는 클래스에도 활용할 수 있습니다.**

클래스 사이의 관계에도 관심을 기울여 봅시다. 클래스 다이어그램에 있는 각 화살표와 클래스 스들이 **어떤 관계인지** (“A는 B이다” 관계인지, “A에는 B가 있다” 관계인지, 아니면 “A가 B를 구현하는” 관계인지) 직접 연필로 써 보세요.

정말 꼭 해 보세요!







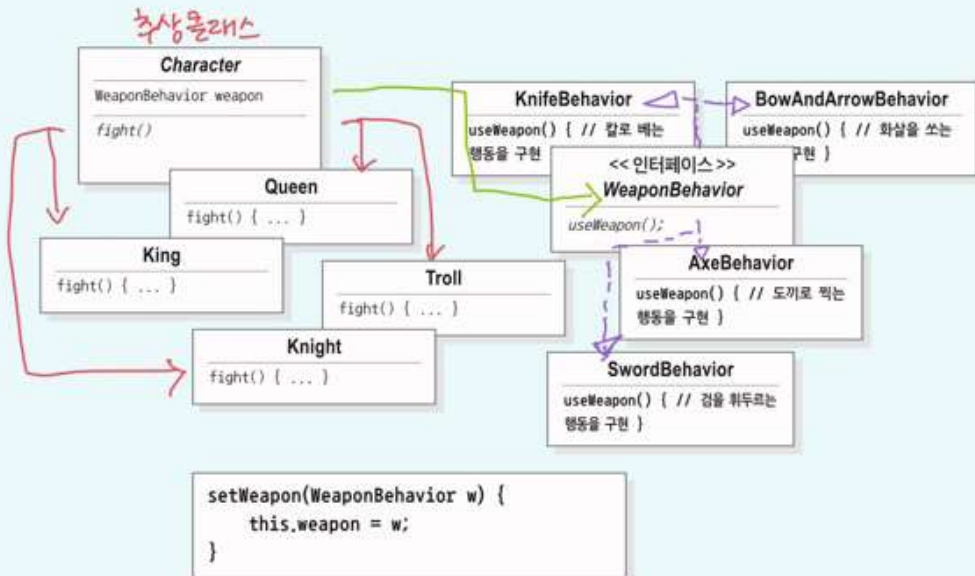
## 디자인 퍼즐

정답 070쪽

아래를 보면 액션 어드벤처 게임에 사용할 클래스와 인터페이스가 어지럽게 흩어져 있습니다. 게임 캐릭터용 클래스와 캐릭터가 사용할 무기의 행동 클래스를 찾을 수 있습니다. 각 캐릭터는 한 번에 한 가지 무기만 사용할 수 있지만, 게임 도중에 무기를 바꿀 수 있습니다. 이 클래스와 인터페이스를 잘 정돈해 봅시다.

### 해야 할 일

- ❶ 클래스를 정돈합니다.
- ❷ 추상 클래스 1개, 인터페이스 1개, 그리고 클래스 8개를 골라냅니다.
- ❸ 클래스들을 화살표로 연결합니다.
  - a. 상속(확장)은 이런 화살표를 사용하세요.
  - b. 인터페이스(구현)는 이런 화살표를 사용하세요.
  - c. "A에는 B가 있다" 관계는 이런 화살표를 사용하세요.
- ❹ 적당한 클래스에 setWeapon() 메소드(무기 설정 메소드)를 추가하세요.



*Character 추상클래스에 추가하고 ...*



여러분의 오른쪽 뇌에도 뭔가 할 일을 줘야겠죠?  
그냥 평범한 날말 퀴즈입니다.  
정답은 모두 1장에 나온 단어입니다.

단어는 영어 알파벳으로 되어 있습니다.  
날말 퀴즈 옆에 있는 단어 리스트를 참고해서 풀어 보세요!



- 옵저버 OBSERVER
- 인터페이스 INTERFACE
- 프레임워크 FRAMEWORKS
- 뻑뻑 소리 SQUEAK
- APis
- 전략 STRATEGY
- 가짜오리 DECOYDUCK
- 경험 EXPERIENCE
- 잭 베니 JACKBENNY
- 캡슐화 ENCAPSULATE
- 용어 VOCABULARY
- 머리 BRAIN
- 구성 COMPOSITION
- 재사용 REUSED
- 유연한 FLEXIBLE
- 변화 CHANGE
- 원칙 PRINCIPLES
- 마우이 MAUI

- 가로**
1. 패턴은 \_\_\_\_\_ 애플리케이션을 만드는 데 도움이 됩니다.
  4. 전략 패턴을 쓰면 코드를 \_\_\_\_\_ 할 수 있습니다.
  7. 상속보다는 \_\_\_\_\_ 이 더 낫죠.
  8. 개발에 있어서 바뀌지 않는 것!
  9. 자바 IO, 네트워킹, 사운드 \_\_\_\_\_
  10. 패턴은 대부분 객체지향 \_\_\_\_\_ 을 따릅니다.
  12. 디자인 패턴은 서로 공유하는 \_\_\_\_\_ 입니다.
  14. 고수준 라이브러리를 \_\_\_\_\_ 라고 부릅니다.
  15. 다른 사람의 \_\_\_\_\_ 으로부터 배울 수 있어야 합니다.
  17. 시뮬레이터에 있는 문제를 \_\_\_\_\_ 패턴으로 해결했죠?
  18. 구현보다는 \_\_\_\_\_ 에 맞춰서 프로그래밍해야 합니다.

- 세로**
2. 패턴을 \_\_\_\_\_ 에 넣어 주세요!
  3. 뻑뻑 소리를 낼 수 없는 오리는?
  5. 고무 오리는 \_\_\_\_\_ 를 냅니다.
  6. 바뀔 수 있는 부분은 \_\_\_\_\_ 해야 합니다.
  11. 베이컨이 들어있는 구운 치즈 샌드위치를 \_\_\_\_\_ 라고 부릅니다.
  13. 아까 오창이 \_\_\_\_\_ 패턴을 좋아했죠?
  16. SimUduck 데모를 선보였던 주주총회가 열렸던 곳의 지명은?