테스트 주도 개발

- 5장 솔직히 말하자면
- 6장 돌아온 '모두를 위한 평등'
- 7장 사과와 오렌지
- 8장 객체 만들기

github/DiveTech

테스트 주도 개발 – 솔직히 말하자면

지금까지 우리는
오로지 초록 막대를 보기 위해
죄악을 저질러서라도 경주마처럼 달려왔다.

오직 테스트를 향상시키기 위해서만 개발된 기능을 사용하고
테스트와 코드 사이의 결합도를 낮추기 위해,
테스트하는 객체의 새 기능을 사용다.

테스트 주도 개발 – 솔직히 말하자면

```
class Dollar {

     private int amount;

     Dollar(int amount) {
          this.amount = amount;
     }

     Dollar times(int multiplier) {
          return new Dollar(amount * multiplier);
     }

     public boolean equals(Object object) {
          Dollar dollar = (Dollar)object;
          return amount == dollar.amount;
     }

}
```

```
public class DollarTest extends TestCase {

     public void test_multiplication() {
          Dollar five = new Dollar(5);
          assertEquals(new Dollar(10), five.times(2));
          assertEquals(new Dollar(15), five.times(3));
     }

     public void test_equality() {
          assertTrue(new Dollar(5).equals(new Dollar(5)));
          assertFalse(new Dollar(5).equals(new Dollar(6)));
     }

}
```

테스트 주도 개발 – 솔직히 말하자면

$5+10CHF=$10(환율이2:1일경우)
~~$5x2=$10~~
~~amount를 private으로 만들기~~
~~Dollar 부작용?~~
Money 반올림?
~~equals()~~
hashCode()
Equal null
Equal object
5CHF x 2=10CHF

테스트 주도 개발 – 솔직히 말하자면

Dollar 객체와 비슷하지만 달러 대신 프랑(Franc)을 표현할 수 있는
객체와 테스트도 같이 만들자.

```
class Franc {

     private int amount;

     Franc(int amount) {
          this.amount = amount;
     }

     Franc times(int multiplier) {
          return new Franc(amount * multiplier);
     }

     public boolean equals(Object object) {
          Franc franc = (Franc)object;
          return amount == franc.amount;
     }

}
```

```
public class FrancTest extends TestCase {

     public void test_multiplication() {
          Franc five = new Franc(5);
          assertEquals(new Franc(10), five.times(2));
          assertEquals(new Franc(15), five.times(3));
     }

     public void test_equality() {
          assertTrue(new Franc(5).equals(new Franc(5)));
          assertFalse(new Franc(5).equals(new Franc(6)));
     }

}
```

테스트 주도 개발 – 솔직히 말하자면

Franc을 만들기 위해..
설상가상으로 모델 코드까지 도매금으로 복사하고 수정해서 테스트를 통과했다.
중복이 사라지기 전에는 집에 가지 않겠다고 약속했다. (내가 언제..?)

테스트 주도 개발 – 솔직히 말하자면

1. 테스트 작성
2. 컴파일되게 하기
3. 실패하는지 확인하기 위해 실행
4. 실행하게 만듦

TDD를 함에 있어
빨리 진행해야 할 4개

5. 중복 제거

테스트 주도 개발 – 솔직히 말하자면

$5+10CHF=$1(0환율이2:1일경우)
~~$5x2=$10~~
~~amount를 private으로 만들기~~
~~Dollar 부작용?~~
Money 반올림?
~~equals()~~
hashCode()
Equal null
Equal object
~~5CHF x 2=10CHF~~
Dollar/Franc 중복
공용 equals
공용 times

새롭게 추가된 할일 3개

테스트 주도 개발 – 솔직히 말하자면

Dollar/Franc 중복

```
class Dollar {

     private int amount;

     Dollar(int amount) {
          this.amount = amount;
     }

     Dollar times(int multiplier) {
          return new Dollar(amount * multiplier);
     }

     public boolean equals(Object object) {
          Dollar dollar = (Dollar)object;
          return amount == dollar.amount;
     }

}
```

```
class Franc {

     private int amount;

     Franc(int amount) {
          this.amount = amount;
     }

     Franc times(int multiplier) {
          return new Franc(amount * multiplier);
     }

     public boolean equals(Object object) {
          Franc franc = (Franc)object;
          return amount == franc.amount;
     }

}
```

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

$5+10CHF=$1(0환율이2:1일경우)
~~$5x2=$10~~
~~amount를 private으로 만들기~~
~~Dollar 부작용?~~
Money 반올림?
~~equals()~~
hashCode()
Equal null
Equal object
~~5CHF x 2=10CHF~~
Dollar/Franc 중복
공용 equals
공용 times

이번에는 "공용 equals" 만들어보자.
조금 전에 중복 제거 하기 전까지는 집에 안간다고 했잖…

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

공용 메소드를 뽑으려고 보니 새로운 클래스를 추가해야하는데
이름은 뭐가 좋을까?

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

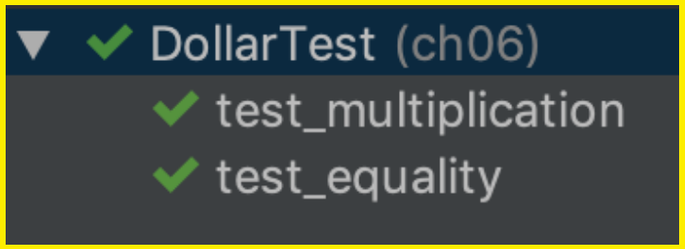
Dollar와 Franc가 속한 카테고리는 돈이므로
공통 상위 클래스 Money를 만들어 상속하자.

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

공통 equals를 만들기 위해 equals에서 사용하는 amount를 money로 올려야 한다.

```
class Dollar extends Money {

     Dollar(int amount) {
          this.amount = amount;
     }

     Dollar times(int multiplier) {
          return new Dollar(amount * multiplier);
     }

     public boolean equals(Object object) {
          Money money = (Dollar)object;
          return amount == money.amount;
     }


}
```

```
class Money {
     protected int amount;
}
```



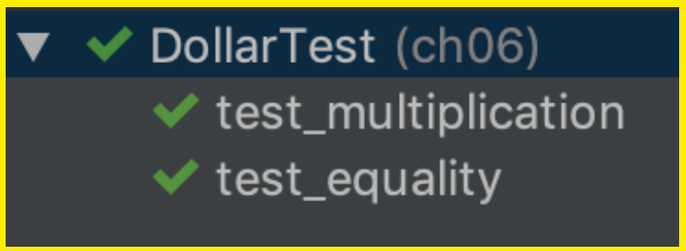
여전히 DollarTest는 깨지지 않고 초록막대가 유지된다.

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

공통 equals를 만들기 위해 equals에서 사용하는 amount를 money로 올려야 한다.

```
class Dollar extends Money {

     Dollar(int amount) {
          this.amount = amount;
     }

     Dollar times(int multiplier) {
          return new Dollar(amount * multiplier);
     }

}
```

```
class Money {
     protected int amount;

     public boolean equals(Object object) {
          Money money = (Money)object;
          return amount == money.amount;
     }
}
```



이번에도 역시 DollarTest는 깨지지 않고 초록막대가 유지된다.

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

Franc도 Dollar와 동일한 과정을 거친다.

- Money를 상속받고 테스트 수행
- amount field를 제거하고 테스트 수행
- equals()를 제거하고 테스트 수행

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

Money Class를 추가함으로써 Dollar와 Franc는 여전히 똑같은 기능을 하지만 기존과는 다른 코드가 되어가고 있다. (물론 좋은 의미로)

이것이 바로 리팩토링이다.

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

하지만 충분한 테스트 코드 지원이 없다면
우리는 불안에 휩싸이고 리팩토링이 실패하면서 리팩토링에 대한 안좋은 경험을 갖게 된다.
그러니 있으면 좋을 것 같은 테스트를 추가로 작성하자.

테스트 주도 개발 – 사과와 오렌지

```
public void test_equality() {
     assertTrue (new Dollar(5).equals(new Dollar(5)));
     assertFalse(new Dollar(5).equals(new Dollar(6)));
     assertTrue (new Franc(5) .equals(new Franc(5)));
     assertFalse(new Franc(5) .equals(new Franc(6)));

     // 테스트 실패 -> equals에서는 amount비교만 하므로 실제 true이다.
     assertFalse(new Franc(5) .equals(new Dollar(5)));
}
```

오직 금액과 클래스가 서로 동일할 때만 두 Money가 서로 같은 것으로 한다.

테스트 주도 개발 – 돌아온 '모두를 위한 평등'

$5+10CHF=$1(0환율이2:1일경우)
~~$5x2=$10~~
~~amount를 private으로 만들기~~
~~Dollar 부작용?~~
Money 반올림?
~~equals()~~
hashCode()
Equal null
Equal object
~~5CHF x 2=10CHF~~
Dollar/Franc 중복
~~공용 equals~~
공용 times
Franc과 Dollar 비교하기

그럼 이제 Money#equals()를 수정해보자.

테스트 주도 개발 – 사과와 오렌지

```
class Money {
     protected int amount;

     public boolean equals(Object object) {
          Money money = (Money)object;
          return amount == money.amount;
     }
}
```

↓

```
class Money {
     protected int amount;

     public boolean equals(Object object) {
          Money money = (Money)object;
          return amount == money.amount && getClass().equals(money.getClass());
     }
}
```

테스트 주도 개발 – 객체 만들기

$5+10CHF=$1(0환율이2:1일경우)
~~$5x2=$10~~
~~amount를 private으로 만들기~~
~~Dollar 부작용?~~
Money 반올림?
~~equals()~~
hashCode()
Equal null
Equal object
~~5CHF x 2=10CHF~~
Dollar/Franc 중복
~~공용 equals~~
공용 times
~~Franc과 Dollar 비교하기~~
통화?

테스트 주도 개발 – 객체 만들기

Dollar/Franc의 중복을 해결하고
공용 times를 해결하자!

테스트 주도 개발 – 객체 만들기

하위 클래스에 대한 직접적인 참조가 적어지도록
Money에 Dollar를 반환하는 factory method를 도입해보자.

```
public void test_multiplication() {
     Dollar five = Money.dollar(5);
     assertEquals(new Dollar(10), five.times(2));
     assertEquals(new Dollar(15), five.times(3));
}
```

```
class Money {
     protected int amount;

     public boolean equals(Object object) {
          Money money = (Money)object;
          return amount == money.amount && getClass().equals(money.getClass());
     }

     static Dollar dollar(int amount) {
          return new Dollar(amount);
     }

}
```

테스트 주도 개발 – 객체 만들기

테스트에서 Dollar에 대한 참조가 사라지길 원하므로
자바의 다형성에 의해 Dollar대신 Money로 대체한다.
하지만 Money에는 times() 메소드가 없어서 에러가 발생한다.

```
public void test_multiplication() {

    Money five = Money.dollar(5);

    assertEquals(new Dollar(10), five.times(2));
    assertEquals(new Dollar(15), five.times(3));

}
```

테스트 주도 개발 – 객체 만들기

$5+10CHF=$1(0환율이2:1일경우)
~~$5x2=$10~~
~~amount를 private으로 만들기~~
~~Dollar 부작용?~~
Money 반올림?
~~equals()~~
hashCode()
Equal null
Equal object
~~5CHF x 2=10CHF~~
Dollar/Franc 중복
~~공용 equals~~
공용 times
~~Franc과 Dollar 비교하기~~
통화?

테스트 주도 개발 – 객체 만들기

이참에 Money는 추상클래스로 변경하는게 좋을 것 같다.
그리고 times는 추상메소드로 정의하여 구현체에서 각각에 맞게 구현하도록 한다.

```
abstract class Money {

     . . .

     abstract Money times(int multiplier);

}
```

테스트 주도 개발 – 객체 만들기

결과 적으로 우리의 DollarTest는 Dollar 참조가 하나도 이루어지지 않은채 테스트가 성공적으로 되었다.

```
public class DollarTest extends TestCase {

     public void test_multiplication() {
          Dollar five = new Dollar(5);
          assertEquals(new Dollar(10), five.times(2));
          assertEquals(new Dollar(15), five.times(3));
     }

     public void test_equality() {
          assertTrue(new Dollar(5).equals(new Dollar(5)));
          assertFalse(new Dollar(5).equals(new Dollar(6)));
     }

}
```

↓

```
public class DollarTest extends TestCase {

     public void test_multiplication() {
          Money five = Money.dollar(5);
          assertEquals(Money.dollar(10), five.times(2));
          assertEquals(Money.dollar(15), five.times(3));
     }

     public void test_equality() {
          assertTrue(Money.dollar(5).equals(Money.dollar(5)));
          assertFalse(Money.dollar(10).equals(Money.dollar(15)));
     }

}
```

테스트 주도 개발 – 객체 만들기

지금까지 우리가 한일👩🏻‍💻

- Franc 계산을 위해 객체 추가
- 중복 제거를 위해 공통 클래스 Money를 추가
- 공통 equals 분리를 위해 Money로 올림
- Dollar/Franc을 팩토리 메서드를 적용하여 생성 주체를 Money로 올림
- 클라이언트는 각 화폐 객체 존재를 모름
- Money를 abstract class로 변경, abstract method – times 추가
- Dollar와 Franc에서 각각 추상메서드 times를 구현하도록 함

끗