TDD 7~12장

7장. 사과와 오렌지

서로 다른 것을 비교할 수 없다

```
public void testEquality(){
    assertTrue(new Dollar( amount: 5).equals(new Dollar( amount: 5)));
    assertFalse(new Dollar( amount: 5).equals(new Dollar( amount: 6)));
    assertTrue(new Franc( amount: 5).equals(new Franc( amount: 5)));
    assertFalse(new Franc( amount: 5).equals(new Franc( amount: 6)));
    assertFalse(new Franc( amount: 5).equals(new Dollar( amount: 5)));
}
```

TestCase 실패

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

서로 다른 Dollar와 Franc의 비교를 위한 equals의 수정이 필요하다

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

But, 현재 이에 해당하는 통화 개념이 없고 이 개념을 도입할 동기가 불충분하다

일단 테스트만 통과하도록 하고 미뤄둠

핵심 : 미뤄야 할 때를 알아야 한다 – 충분한 동기가 있을 때 설계를 도입한다

8장. 객체 만들기

하위 클래스에 대한 참조 줄이기

Factory Method를 도입해서 하위 클래스에 대한 직접적인 참조를 줄인다.

Test Code 수정

구현 코드 수정

에러-times()가 Money에 정의되지 않음

에러 수정 & 팩토리 메소드 나머지 코드에 모두 적용

```
public void testMultiplication(){
    Money five = Money.dollar( amount: 5);
    assertEquals(Money.dollar( amount: 10),five.times( multiplier: 2));
    assertEquals(Money.dollar( amount: 15),five.times( multiplier: 3));

}
```

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

public static Money franc(int amount){
    return new Franc(amount);
}
```

```
abstract public Money times(int multiplier);
```

8장. 객체 만들기

하위 클래스에 대한 참조 줄이기

```
        Dollar five = new Dollar(5);
        assertEquals(new Dollar(10),five.times(2));
        assertEquals(new Dollar(15),five.times(3));

    }

    @org.junit.Test
    public void testFrancMultiplication(){
        Franc five = new Franc(5);
        assertEquals(new Franc(10),five.times(2));
        assertEquals(new Franc(15),five.times(3));

    }

    @org.junit.Test
    public void testEquality(){
        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)));
        assertTrue(new Franc(5).equals(new Dollar(5)));
    }
```

```
13 14
14 15
15 16
16 17
17 18
18 19
19 20
20 21
21 22
22 23
23 24
24 25
25 26
26 27
27 28
28 29
29 30
30 31
31 32
32 33
33 34
34 35
```

```
        Money five = Money.dollar(5);
        assertEquals(Money.dollar(10),five.times(2));
        assertEquals(Money.dollar(15),five.times(3));

    }

    @org.junit.Test
    public void testFrancMultiplication(){
        Franc five = new Franc(5);
        assertEquals(Money.franc(10),five.times(2));
        assertEquals(Money.franc(15),five.times(3));

    }

    @org.junit.Test
    public void testEquality(){
        assertTrue(Money.dollar(5).equals(Money.dollar(5)));
        assertFalse(Money.dollar(5).equals(Money.dollar(6)));
        assertTrue(Money.franc(5).equals(Money.franc(5)));
        assertFalse(Money.franc(5).equals(Money.franc(6)));
        assertTrue(Money.franc(5).equals(Money.dollar(5)));
    }
```

9장. 우리가 사는 시간

통화 개념 도입

Test Code 추가

하위 클래스에 필요 메소드 & 변수 추가

동일 부분 상위 클래스로 올리기

생성자에 인자 추가 & Currency 문자열 정적 팩토리 메서드로 올리기

중복되는 부분을 상위 클래스의 생성자로 올리기

```
public class Franc extends Money {

    private String currency;

    Franc(int amount) {
        this.amount = amount;
        currency = "CHF";
    }

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

    String currency() {
        return currency;
    }

}
```

```
public class Dollar extends Money {

    private String currency;

    Dollar(int amount) {
        this.amount = amount;
        currency = "USD";
    }

    Money times(int multiplier) {
        return new Dollar( amount: this.amount * multiplier);
    }

    String currency() {
        return currency;
    }

}
```

Money

```
protected String currency;

new *
public String currency(){
    return currency;
}
```

```
static Dollar dollar(int amount) {
    return new Dollar(amount, currency: "USD");
}

static Franc franc(int amount) {
    return new Franc(amount, currency: "CHF");
}
```

```
Dollar(int amount, String currency) {
    this.amount = amount;
    this.currency = currency;
}
```

```
Money(int amount, String currency) {
    this.amount = amount;
    this.currency = currency;
}
```

10장. 흥미로운 시간

하나의 클래스로 money 나타내기

아직 times()의 구현이 다르며, 동일하게 만들 방법이 없다
다시 이전 버전처럼 팩토리 메소드를 인라인시키자

```
public Money times(int multiplier) {
    return Money.franc( amount: amount*multiplier);
}
public Money times(int multiplier) {
    return new Franc( amount: amount*multiplier,currency);
}
```

```
public Money times(int multiplier) {
    return Money.dollar( amount: amount*multiplier);
}
public Money times(int multiplier) {
    return new Dollar( amount: amount*multiplier,currency);
}
```

Times에서 Dollar를 리턴해야할 지 Money를 리턴해야할 지 모르겠다면 컴퓨터에게 물어봐라

```
Expected :code.Dollar@2a742aa2
Actual   :code.Money@3cb1ffe6
```

```
Expected :code.Dollar<10    USD>
Actual   :code.Money<10    USD>
```

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

```
public Money times(int multiplier) {
    return new Money( amount: amount*multiplier,currency);
}
```

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

11장. 모든 악의 근원

하위 클래스에 대한 참조 제거하기

하위 클래스를 참조하는 곳을 제거했다

```
public static Money dollar(int amount){
    return new Dollar(amount, currency: "USD");
}

SeoungJun
public static Money franc(int amount){
    return new Franc(amount, currency: "CHF");
}
```

```
public static Money dollar(int amount){
    return new Money(amount, currency: "USD");
}

SeoungJun *
public static Money franc(int amount){
    return new Money(amount, currency: "CHF");
}
```

테스트 코드에서 하위 클래스를 참조하는 곳을 제거했다 ; testEquality에서 동치성 테스트를 하고 있기 때문

```
@Test
public void TestDifferentClassEquality(){
    assertTrue(new Money( amount: 10, currency: "CHF").equals(
            new Franc( amount: 10, currency: "CHF")));
}
```

```
@Test
public void testEquality(){
    assertTrue(Money.dollar( amount: 5).equals(Money.dollar( amount: 5)));
    assertFalse(Money.dollar( amount: 5).equals(Money.dollar( amount: 6)));
    assertTrue(Money.franc( amount: 5).equals(Money.franc( amount: 5)));
    assertFalse(Money.franc( amount: 5).equals(Money.franc( amount: 6)));
    assertFalse(Money.franc( amount: 5).equals(Money.dollar( amount: 5)));
}
```

12장. 드디어 더하기

다중 통화 연산을 표현하는 방법

1. 모든 내부 값을 참조통화로 전환하는 방법 – 여러 환율을 쓰기 어려움

2. 외부 구현은 같으면서 내부 구현은 다른 객체(imposter)를 만들어서 사용
   - Money와 비슷하게 동작하지만, 두 Money의 합을 나타내는 객체를 만드는 것

   1. Money의 합을 지갑처럼 취급하는 방법 – 금액과 통화가 다른 여러 화폐가 Money에 들어갈 수 있음

   2. Money의 합을 수식으로 취급하는 방법 ($2 + 3CHF) * 5
      - Money를 수식의 가장 작은 단위로 볼 수 있음
      - 연산의 결과로 **Expression**이 생기고 **그 중 하나는 sum**임
      - 연산이 완료되면 환율을 이용해서 **Expression**을 단일 통화로 축약할 수 있음

12장. 드디어 더하기

Reduced는 Expression으로 Expression에 환율을 적용함으로써 얻어진다
Bank가 reduce의 책임을 가짐

```
public void testSimpleAddition(){
    Money five = Money.dollar( amount: 5);
    Expression sum = five.plus(five);
    Bank bank = new Bank();
    Money reduced = bank.reduce(sum,  to: "USD");
    assertEquals(Money.dollar( amount: 10),reduced);
}
```

왜 Bank에서 reduce에 대한 책임을 가지나?
A) 핵심 객체인 Expression이 가능한 오랫 동안 유연하게 하기 위해서
+ 테스트하기 쉽고, 재활용 & 이해가 쉬움
+ 추가 되는 기능을 생각하면 Expression에 몰릴 가능성이 있음
-> Expression이 다른 부분에 대해서는 모르도록 해야함

일단은 reduce에 대해 스텁 구현을 해놓음

```
public Money reduce(Expression source, String to){
    return Money.dollar( amount: 10);
}
```

결과적으로는, 컴파일이 가능하며
testCode가 전부 초록색임
리팩토링할 수 있음