TDD Example - Movie

▪ 영화 등급 조회

- Movie 클래스
- void rate(int rate) 메서드를 이용하여 등급을 부여
- int averageRate() 메서드를 이용하여 평균 등급을 조회

TDD Example - Movie

- **Movie 클래스, averageRating 메서드 생성**

Red

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class MovieTest {
  @Test
  public void testMovie() {
      Movie movie = new Movie();
      assertEquals(0, movie.averageRate());
  }
}
```

Green

```
public class Movie {
  public int averageRate() {
      return 0;
  }
}
```

Blue

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class MovieTest {
  @Test
  public void should_return_0_when_just_created() {
      Movie movie = new Movie();
      assertEquals(0, movie.averageRate());
  }
}
```

← 의미있는 이름 부여

TDD Example - Movie

- **등급으로 1을 줬을 때, 평균 등급으로 1을 반환**

Red

```
public class MovieTest {
  @Test
  public void should_return_0_when_just_created() {
      Movie movie = new Movie();
      assertEquals(0, movie.averageRate());
  }
  @Test
  public void should_return_1_when_was_rated() {
      Movie movie = new Movie();
      movie.rate(1);
      assertEquals(movie.averageRate(), 1);
  }
}
```

Green

```
public class Movie {
  public int averageRate() {
      return 0;
  }
  public void rate(int i) {
      // 컴파일을 통과하기 위한 최소 코드
  }
}
```

TDD Example - Movie

- **등급으로 1을 줬을 때, 평균 등급으로 1을 반환**

Green

```
public class Movie {
  private int sumOfRate = 0;
  private int countOfRate = 0;

  public int averageRate() {
     // 동작하기 위한 최소 코드
     return sumOfRate / countOfRate;
  }
  public void rate(int rate) {
     countOfRate ++;
     sumOfRate += rate;
  }
}
```

Green

```
public class Movie {
  private int sumOfRate = 0;
  private int countOfRate = 0;

  public int averageRate() {
     // ArithmeticException이 발생하지 안도록 수정
     if (countOfRate == 0)
        return 0;
     return sumOfRate / countOfRate;
  }
  public void rate(int rate) {
     countOfRate ++;
     sumOfRate += rate;
  }
}
```

TDD Example - Movie

- **등급으로 1을 줬을 때, 평균 등급으로 1을 반환**

Blue

```
public class MovieTest {
  private Movie movie;     // 중복(인스턴스 생성) 제거

  @Before
  public void setUp() {
      movie = new Movie();
  }
  @Test
  public void should_return_0_when_just_created() {
      assertEquals(0, movie.averageRate());
  }
  @Test
  public void should_return_1_when_was_rated() {
      movie.rate(1);
      assertEquals(movie.averageRate(), 1);
  }
}
```

TDD Example - Movie

- **등급으로 3, 5를 줬을 때, 평균 등급으로 4를 반환**

```
public class MovieTest {
  private Movie movie;

  @Before
  public void setUp() {
      movie = new Movie();
  }
  @Test
  public void should_return_0_when_just_created() {
      assertEquals(0, movie.averageRate());
  }
  @Test
  public void should_return_1_when_was_rated() {
      movie.rate(1);
      assertEquals(1, movie.averageRate());
  }
  @Test
  public void should_return_4_when_3_and_5_were_rated() {      ➔ 수정 없이 테스트 통과
      movie.rate(3);
      movie.rate(5);
      assertEquals(4, movie.averageRate());
  }
}
```

TDD Example - Vending Machine

- **Vending Machine**
  - 입금된 금액을 저장한다.
  - 상품을 선택하면 입금액에서 상품 가격을 차감한다.
  - 잔액을 보여준다.
  - 반환 버튼을 누르면 잔액을 반환한다.
  - 잔액은 1000원 지폐와 500원, 100원, 50원, 10원 동전을 사용한다.
  - 지폐로 지급할 수 있는 금액을 동전으로 지급하지 않는다.
  - 각각의 동전은 최소 개수를 사용한다.
  - 반환 결과를 화면에 보여준다.

TDD Example - Vending Machine

- 입금된 금액을 저장한다. 잔액을 보여준다.

Red

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class VendingMachineTest {
  @Test
  public void testPrintMoney() {
      VendingMachine vm = new VendingMachine();
      vm.inputMoney(2000);
      assertEquals(2000, vm.showMoney());
  }
}
```

Green

```
public class VendingMachine {
  public void inputMoney(int money) {

  }
  public int showMoney() {
      return 2000;
  }
}
```

TDD Example - Vending Machine

- **상품을 선택하면 입금액에서 상품 가격을 차감한다.**

Red

```
public class VendingMachineTest {
  @Test
  public void testInputMoney() {
      VendingMachine vm = new VendingMachine();
      vm.inputMoney(2000);
      assertEquals(2000, vm.showMoney());
  }
  @Test
  public void testChooseProduct() {
      VendingMachine vm = new VendingMachine();
      vm.inputMoney(2000);
      vm.chooseProduct(1350);
      assertEquals(2000-1350, vm.showMoney());
  }
}
```

Green

```
public class VendingMachine {
  private int money = 0;

  public void inputMoney(int money) {
      this.money += money;
  }
  public int showMoney() {
      return this.money;
  }
  public void chooseProduct(int price) {
      this.money -= price;
  }
}
```

TDD Example - Vending Machine

- **상품을 선택하면 입금액에서 상품 가격을 차감한다.**

Blue

```
public class VendingMachineTest {
  private VendingMachine vm;

  @Before
  public void setUp() {
      vm = new VendingMachine();
  }
  @Test
  public void testInputMoney() {
      vm.inputMoney(2000);
      assertEquals(2000, vm.showMoney());
  }
  @Test
  public void testChooseProduct() {
      vm.inputMoney(2000);
      vm.chooseProduct(1350);
      assertEquals(2000-1350, vm.showMoney());
  }
}
```

TDD Example - Vending Machine

- **지폐로 지급할 수 있는 금액을 동전으로 지급하지 않는다.**

Red

```
public class VendingMachineTest {
      :
  @Test
  public void 잔액이_1000원이상인_경우_1000원지폐_지급() {
      vm.inputMoney(3000);
      vm.chooseProduct(1350);
      int bills = (3000-1350) / 1000;
      assertEquals(bills, vm.billsToReturn());
      int coins = (3000-1350) - (1000*bills);
      assertEquals(coins, vm.showMoney());
  }
  @Test
  public void 잔액이_1000원미만인_경우_1000원지폐_없음() {
      vm.inputMoney(3000);
      vm.chooseProduct(2350);
      assertEquals(0, vm.billsToReturn());
  }
}
```

Green

```
public class VendingMachine {
  private int money = 0;

  public void inputMoney(int money) {
      this.money += money;
  }
  public int showMoney() {
      return this.money;
  }
  public void chooseProduct(int price) {
      this.money -= price;
  }
  public int billsToReturn() {
      int bills = this.money / 1000;
      this.money -= 1000 * bills;
      return bills;
  }
}
```

TDD Example - Vending Machine

- 각각의 동전은 최소 개수를 사용한다.

Red

```
public class VendingMachineTest {
      :
  @Test
  public void 잔액이_1000원미만_500원이상인_경우_500원동전_지급() {
      vm.inputMoney(1000);
      vm.chooseProduct(350);
      int coinsOf500 = (1000-350) / 500;
      assertEquals(coinsOf500, vm.coinsOf500ToReturn());
      assertEquals(150, vm.showMoney());
  }
  @Test
  public void 잔액이_500원미만인_경우_500원동전_없음() {
      vm.inputMoney(500);
      vm.chooseProduct(350);
      assertEquals(0, vm.coinsOf500ToReturn());
  }
}
```

Green

```
public class VendingMachine {
        :

  public int coinsOf500ToReturn() {
      int coins = this.money / 500;
      this.money -= 500 * coins;
      return coins;
  }
}
```

TDD Example - Vending Machine

- 각각의 동전은 최소 개수를 사용한다.

Red

```
public class VendingMachineTest {
      :
  @Test
  public void 잔액이_500원미만_100원이상인_경우_100원동전_지급() {
      vm.inputMoney(500);
      vm.chooseProduct(350);
      int coinsOf100 = (int) (500-350) / 100;
      assertEquals(coinsOf100, vm.coinsOf100ToReturn());
      assertEquals(50, vm.showMoney());
  }
  @Test
  public void 잔액이_100원미만인_경우_100원동전_없음() {
      vm.inputMoney(400);
      vm.chooseProduct(350);
      assertEquals(0, vm.coinsOf100ToReturn());
  }
}
```

Green

```
public class VendingMachine {
        :
  public int coinsOf100ToReturn() {
      int coins = this.money / 100;
      this.money -= 100 * coins;
      return coins;
  }
}
```

TDD Example - Vending Machine

- 각각의 동전은 최소 개수를 사용한다.

Red

```
public class VendingMachineTest {
      :
  @Test
  public void 잔액이_100원미만_50원이상인_경우_50원동전_지급() {
      vm.inputMoney(400);
      vm.chooseProduct(330);
      int coinsOf50 = (int) (400-330) / 50;
      assertEquals(coinsOf50, vm.coinsOf50ToReturn());
      assertEquals(20, vm.showMoney());
  }
  @Test
  public void 잔액이_50원미만인_경우_50원동전_없음() {
      vm.inputMoney(350);
      vm.chooseProduct(330);
      assertEquals(0, vm.coinsOf50ToReturn());
  }
}
```

Green

```
public class VendingMachine {
        :
  public int coinsOf50ToReturn() {
      int coins = this.money / 50;
      this.money -= 50 * coins;
      return coins;
  }
}
```

TDD Example - Vending Machine

- 각각의 동전은 최소 개수를 사용한다.

Red

```
public class VendingMachineTest {
      :
  @Test
  public void 잔액이_50원미만인_경우_10원동전_지급() {
      vm.inputMoney(350);
      vm.chooseProduct(330);
      int coinsOf10 = (int) (350-330) / 10;
      assertEquals(coinsOf10, vm.coinsOf10ToReturn());
  }
}
```

Green

```
public class VendingMachine {
        :
  public int coinsOf10ToReturn() {
      int coins = this.money / 10;
      this.money -= 10 * coins;
      return coins;
  }
}
```

TDD Example - Vending Machine

- 각각의 동전은 최소 개수를 사용한다.

Red

```
public class VendingMachineTest {
      :
  @Test
  public void 잔액이_1000원이상인_경우() {
      vm.inputMoney(3000);
      vm.chooseProduct(1340);
      vm.calcurate();
      vm.display();
  }

  @Test
  public void 잔액이_1000원미만인_경우() {
      vm.inputMoney(2000);
      vm.chooseProduct(1340);
      vm.calcurate();
      vm.display();
  }
}
```

Green

```
public class VendingMachine {
        :
  private int billsOf1000 = 0;
  private int coinsOf500 = 0;
  private int coinsOf100 = 0;
  private int coinsOf50 = 0;
  private int coinsOf10 = 0;
  public void calcurate() {
      this.billsOf1000 = billsToReturn();
      this.coinsOf500 = coinsOf500ToReturn();
      this.coinsOf100 = coinsOf100ToReturn();
      this.coinsOf50 = coinsOf50ToReturn();
      this.coinsOf10 = coinsOf10ToReturn();
  }
  public void display() {
      System.out.println("1000원 : " + this.billsOf1000);
      System.out.println("500원 : " + this.coinsOf500);
      System.out.println("100원 : " + this.coinsOf100);
      System.out.println("50원 : " + this.coinsOf50);
      System.out.println("10원 : " + this.coinsOf10);
  }
}
```