Łukasz Piskor

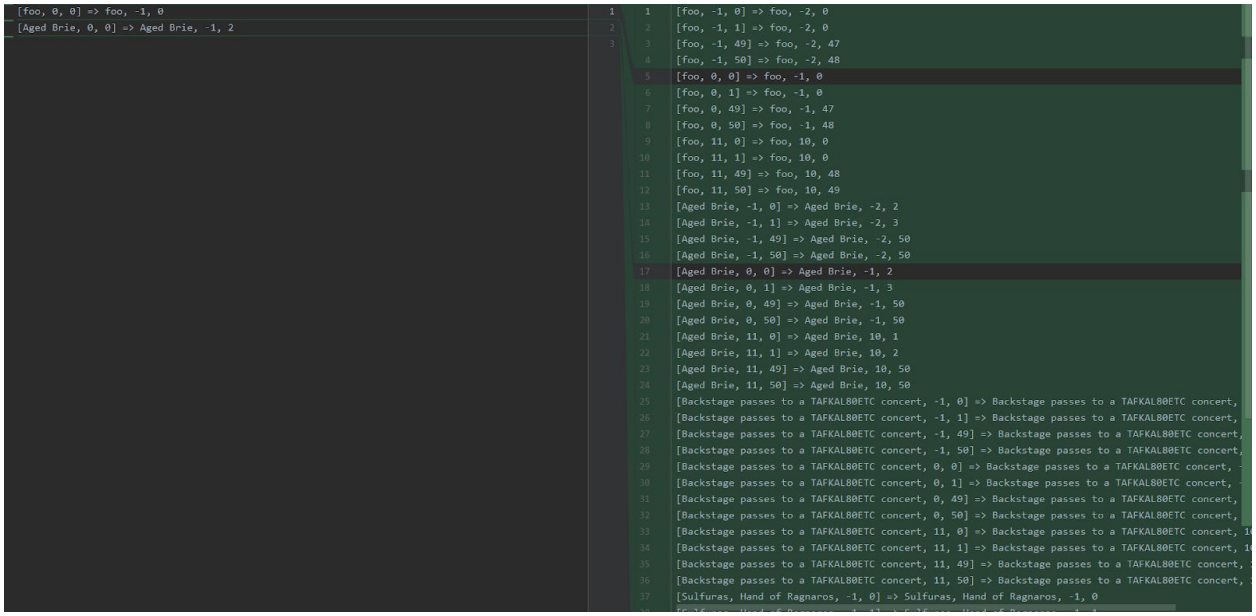
Gilded Rose Kata - Java

Po pobraniu niezbędnych plików ze wskazanego repozytorium załadowałem je jako Maven Project do środowiska InteliJ. Pracę z kodem rozpocząłem od testu “GildedRoseTest”. Nie działał poprawnie, jego wartość oczekiwała na wartość string “fixme” podczas, gdy wprowadzoną wartością było zmienna string “foo”. Pierwszym krokiem, który poczyniłem była zmiana oczekiwanej wartości na “foo”. Następnie zmieniłem nazwę metody z “foo” na “updateQuality”. Kolejno zaimportowałem ApprovalTests do pliku pom.xml. Korzystając z nowo dodanych zmian rozpocząłem proces upraszczania kodu od uproszczenia testu z użyciem funkcji “Approvals.verify” zamiast “assertEquals”. Przy próbie przeprowadzenia testu JUnit zwrócił błąd “junit.framework.ComparisonFailure:” dla zatwierdzenia zmian użyłem polecenia “move” z właściwą ścieżką pliku, komendę tą wprowadziłem w oknie terminalu. Następnie zmodyfikowałem wartość poddawaną weryfikacji na “app.items[0].toString()”, co wymagało ponownego użycia polecenia “move” w konsoli. Kolejno zmieniłem ustawienia InteliJ dla obsługi testów Edit Configurations -> Code Coverage -> Sampling na Edit Configurations -> Code Coverage -> Tracing. Odznaczyłem opcję Settings -> Build, Execution, Deployment -> Coverage -> Activate Coverage View oraz wybrałem opcję “Replace active suites with the new one”.

Ponownie uruchomiłem test korzystając z opcji “Run with coverage”, co po przejściu do pliku GildedRose.java pozwoliło mi na obserwację jak dobrze test pokrywa działania wykonywane przez program oznaczając je przy pomocy 3 kolorów(zielony, żółty, czerwony) zależnie od stopnia pokrycia oraz wyrażając, gdzie leży problem po najechaniu na odpowiednie miejsce. Rozpocząłem sprawdzanie błędów od “żółtych” sprawdzając linijki kodu od góry. Komunikat błędu wskazywał na fakt, iż wartość “items[i].name.equals("Aged Brie" po naciśnięciu zawsze zwróci wartość true, nigdy nie będzie fałszywa. Aby uwzględnić to w moim teście utworzyłem zmienną typu string “itemStr” przechowującą dane, które wcześniej podałem jako wartość do sprawdzenia przez funkcję “Approvals.verify”. Również zmienne zawarte w wartościach tablicy zapisałem w postaci zmiennych przed jej wywołaniem. Pozwoliło mi to na utworzenie metody dla uproszczenia zapisu programu oraz użycie funkcji `CombinationApprovals.verifyAllCombinations(this::doUpdate, new String[]{name}, new Integer[]{sellIn}, new Integer[]{quality});`, co wymagało ponownego przeniesienia pliku w terminalu z użyciem “move” oraz późniejszego zastąpienia zmiennych name, sellIn, quality wartościami do nich przypisanych korzystając z funkcji InteliJ “Inline Variable”. Wykonanie zastąpienia “Inline” pozwoliło mi na usunięcie niepotrzebnych dłużej zmiennych przechowujących wartości, a także “itemStr” wraz z pozbyciem się wcześniej używanej `Approvals.verify(itemStr);`.

```java
package com.gildedrose;

import ...

public class GildedRoseTest {

    @Test
    public void updateQuality() throws Exception{
        CombinationApprovals.verifyAllCombinations(
                this::doUpdate,
                new String[]{"foo"},
                new Integer[]{0},
                new Integer[]{0});
    }

    private String doUpdate(String name, int sellIn, int quality) {
        Item[] items = new Item[] { new Item(name, sellIn, quality) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        return app.items[0].toString();
    }
```

W ten sposób przygotowany test wzbogacam o kolejne wartości wpisując je po kolei zaczynając od pierwszej “żółtej” linijki, aby zwiększyć pokrycie testu: “Aged Brie”, "Backstage passes to a TAFKAL80ETC concert", wartość quality większą od 0 (w moim przypadku 1), "Sulfuras, Hand of Ragnaros", wartość quality większa równa 50, wartość sellIn większa równa 11, wartość quality równa 49, wartość sellIn równą 6, wartość sellIn równa -1. Po dodaniu wszystkich wymienionych wartości test w pełni pokrywa kod GildedRose.java. Pozwala mi to na odnalezienie dwóch wartości różnych od oczekiwanych.



Błąd udaje się naprawić, po raz kolejny używając polecenia “move”. Następnie testowałem ręcznie poprzez zmianę wartości dodanej lub odejmowanej w pojedynczych liniach kodu(mutacji) czego skutkiem okazał się brak błędu w 32 linii kodu wynikający z braku dostatecznej liczby wartości pośrednich dla sellIn, co naprawiam poprzez wstawienie do tablicy wartości 2, zatwierdzam zmiany używając polecenia “move”.Ostateczna forma testu prezentuje się następująco:

```
package com.gildedrose;

import ...

public class GildedRoseTest {

    @Test
    public void updateQuality() throws Exception{
        CombinationApprovals.verifyAllCombinations(
                this::doUpdate,
                new String[]{"foo", "Aged Brie", "Backstage passes to a TAFKAL80ETC concert", "Sulfuras, Hand of Ragnaros"},
                new Integer[]{-1, 0, 2, 6, 11},
                new Integer[]{0, 1, 49, 50});
    }

    private String doUpdate(String name, int sellIn, int quality) {
        Item[] items = new Item[] { new Item(name, sellIn, quality) };
        GildedRose app = new GildedRose(items);
        app.updateQuality();
        return app.items[0].toString();
    }
```

Dobre pokrycie kodu przez test pozwala mi na przejście do refaktoryzacji właściwej i uproszczenie jego składni. Zacząłem od utworzenia zmiennej lokalnej Item item = items[i]; i zamienienia wszystkich wartości “items[i]” na item. Cały kod zaczynając od pierwszej pętli if spakowałem jako metodę “doUpdate(items[i]);”. Następnie przeszedłem w dół rozpoczynając od 18 linii zaznaczyłem obszar do końca i spakowałem jako metodę “foo(item)” nad którą stworzyłem warunek :

```
    private void doUpdate(Item item) {
        if (item.name.equals("Aged Brie")) {
            foo(item);
        } else {
            foo(item);
        }
    }
    private void foo(Item item) {
```

Następnie zaznaczyłem “foo” w pętli i użyłem narzędzia inline. Po uruchomieniu testu można było zauważyć, że część kodu jest nieaktywna. Usunąłem z pierwszej zagnieżdżonej pętli część o przedmiocie Sulfuras oraz ustawiłem wartość drugiego if’a sprawdzającego czy wartość jest

nierówna “Aged Brie” jako fałszywą (Wartość będzie zawsze fałszywa, if przed nią sprawdza czy wartość jest równa “Aged Brie”). Środowisko podkreśla warunek false oraz następujące po nim sprawdzenie dając mi możliwość uproszczenia i pozbycia się martwego kodu występującego po wartości “false”, a to z kolei pozwala na odwrócenie warunku przez program do wartości “true” oraz ponowne jego uproszczenie.

Z kolejnym warunkiem oznaczonym przez środowisko na żółto dotyczącym biletów postępuję alegorycznie, sprawdzam uwagę mówiącą, że wg testów wartość będzie zawsze fałszywa i zmieniam warunek if na “false”, środowisko sugeruje uproszczenie kodu poprzez usunięcie warunku. Schodząc do kolejnego zagnieżdżenia mówiącego, że wartość "Sulfuras, Hand of Ragnaros" jest nieprawdziwa otrzymujemy z testów wartość zawsze true, co umożliwia rozpakowanie if’a. W ten sam sposób sprawdzam wszystkie warunki oznaczone na żółto nadając im odpowiednie wartości dla if, następnie upraszczając, aż do momentu 100% pokrycia testu czego wynikiem jest uproszczona metoda dla “Aged Brie”. Powtarzam proces pakowania metody dla następnego przedmiotu(Backstage passes, później Sulfuras) następnie wykonując wszystkie kroki wykonywane dla “Aged Brie”. Po ich ukończeniu zmieniam warunki rozpoczynające z else do else if i całą metodę z pętli warunkowej składającej się z ifów na switch. Dodałem “Conjured Mana Cake” do switcha oraz uwzględniłem go w testach.

Następnie zająłem się skróceniem updateQuality, co rozpocząłem od pozbycia się powtarzających elementów. Nowa metoda plusQuality przejęła kontrolę nad sprawdzaniem czy wartość quality nie jest zbyt duża oraz zwiększał jej wartość o odpowiednią ilość. Metoda outOfDate sprawdzała czy produkty nie są przeterminowane i obniżała ich wartość jeśli były. Kolejną utworzoną metodą jest endOfDay odliczający upłynięcie dnia poprzez zmniejszenie wartości SellIn. Ostatnią z metod mających eliminację powtórzeń jest qualityLoss, która sprawdza wartość quality i obniża ją, gdy ta jest wyższa od 0.

Ostatnim krokiem w mojej refaktoryzacji było utworzenie reprezentacji dla samych przedmiotów. W tym przypadku również posłużyłem się metodami o nazwach odpowiadających obsługiwanym przedmiotom tj. “AgedBrie”, “BackstagePasses”, oraz “BasicAndConjured”, która to obsługuje zarówno “zwykłe” przedmioty nie mające dodatkowych warunków, jak i “zaczarowane”. Każda z tych metod wywołuje przydzielone jej w procesie eliminacji powtórzeń metody, co pozwala na drastyczne skrócenie metody doUpdate.

```
    private void doUpdate(Item item) {
        switch (item.name) {
            case "Aged Brie":
                AgedBrie(item);
                break;
            case "Backstage passes to a TAFKAL80ETC concert":
                BackstagePasses(item);
                break;
            case "Sulfuras, Hand of Ragnaros":
                break;
            case "Conjured Mana Cake":
                BasicAndConjured(item, i: 2);
                break;
            default:
                BasicAndConjured(item, i: 1);
        }
    }
```

Program przeszedł testy pomyślnie i zwrócił oczekiwane wartości. Poniżej zamieszczam zrzuty ekranu, które prezentują wartości produktów dla kilku kluczowych dni.

```
-------- day 0 --------
name, sellIn, quality
+5 Dexterity Vest, 10, 20
Aged Brie, 2, 0
Elixir of the Mongoose, 5, 7
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 15, 20
Backstage passes to a TAFKAL80ETC concert, 10, 49
Backstage passes to a TAFKAL80ETC concert, 5, 49
Conjured Mana Cake, 3, 6

-------- day 1 --------
name, sellIn, quality
+5 Dexterity Vest, 9, 19
Aged Brie, 1, 1
Elixir of the Mongoose, 4, 6
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 14, 21
Backstage passes to a TAFKAL80ETC concert, 9, 50
Backstage passes to a TAFKAL80ETC concert, 4, 50
Conjured Mana Cake, 2, 4
```

Zwiększenie wartości dla “Backstage passes” w momencie, gdy wartość SellIn jest mniejsza, niż 11

```
-------- day 5 --------
name, sellIn, quality
+5 Dexterity Vest, 5, 15
Aged Brie, -3, 8
Elixir of the Mongoose, 0, 2
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 10, 25
Backstage passes to a TAFKAL80ETC concert, 5, 50
Backstage passes to a TAFKAL80ETC concert, 0, 50
Conjured Mana Cake, -2, 0

-------- day 6 --------
name, sellIn, quality
+5 Dexterity Vest, 4, 14
Aged Brie, -4, 10
Elixir of the Mongoose, -1, 0
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 9, 27
Backstage passes to a TAFKAL80ETC concert, 4, 50
Backstage passes to a TAFKAL80ETC concert, -1, 0
Conjured Mana Cake, -3, 0
```

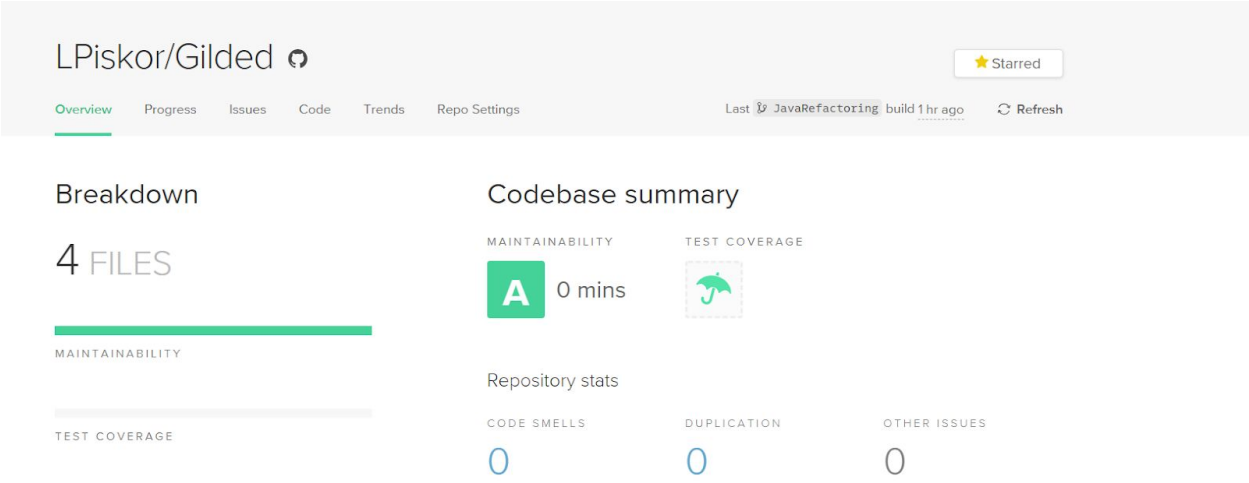
Skok wartości dla tego samego przedmiotu w momencie, gdy SellIn < 6.

```
-------- day 10 --------
name, sellIn, quality
+5 Dexterity Vest, 0, 10
Aged Brie, -8, 18
Elixir of the Mongoose, -5, 0
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 5, 35
Backstage passes to a TAFKAL80ETC concert, 0, 50
Backstage passes to a TAFKAL80ETC concert, -5, 0
Conjured Mana Cake, -7, 0

-------- day 11 --------
name, sellIn, quality
+5 Dexterity Vest, -1, 8
Aged Brie, -9, 20
Elixir of the Mongoose, -6, 0
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 4, 38
Backstage passes to a TAFKAL80ETC concert, -1, 0
Backstage passes to a TAFKAL80ETC concert, -6, 0
Conjured Mana Cake, -8, 0
```

Test przeprowadzono dla 29 dni, wyniki w ostatnim dniu prezentują się następująco:

```
-------- day 29 --------
name, sellIn, quality
+5 Dexterity Vest, -19, 0
Aged Brie, -27, 50
Elixir of the Mongoose, -24, 0
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, -14, 0
Backstage passes to a TAFKAL80ETC concert, -19, 0
Backstage passes to a TAFKAL80ETC concert, -24, 0
Conjured Mana Cake, -26, 0
```

Pracę z Gilded Rose zakończyłem aktualizacją danych na Githubie oraz sprawdzeniu jakości kodu poprzez narzędzie Code Climate sparowane z repozytorium.



Poprawiony przeze mnie kod uzyskał ocenę A (Projekty o technicznych wskaźnikach zadłużenia poniżej 5%)(Wskaźnik łatwości utrzymania kodu).