

Gilded Rose Kata

Autor: [Gustaw Napiórkowski](#)

Kroki główne:

Przeniesienie funkcjonalności zmiany wartości sell_in do osobnej funkcji

Została dodana nowa funkcja odpowiedzialna za zmianę wartości sell_in dla danego przedmiotu o -1 dla wszystkich przedmiotów, poza przedmiotem legendarnym o nazwie "Sifuras"

Funkcja ta wygląda następująco:

```
def sell_in_update(self, item):  
    if 'Sulfuras' not in item.name:  
        item.sell_in -= 1  
    item.sell_in -= 1
```

Restrukturyzacja funkcji update_quality

Na drodze planowania refaktoryzacji dla tej funkcji została podjęta decyzja o przekształceniu jej w funkcję wyższego rzędu wywołującą dwie inne - sell_in_update() oraz quality_update, gdzie ta druga jest odpowiednikiem starej funkcji update_quality.

Pętla iterująca przedmioty została także wyciągnięta do tej funkcji ze starej wersji update_quality w celu zwiększenia przejrzystości.

Nowa funkcja update_quality miała następujący design:

```
def update_quality(self):  
    for item in self.items:  
        self.sell_in_update(item)  
        self.quality_update(item)
```

Funkcja quality_update() - następca starej update_quality()

Funkcja `quality_update` została stworzona jako dokładna kopia pierwotnej funkcji `update_quality` jednak w trakcie pracy postanowiono pozostawić w niej tylko funkcjonalność identyfikacji przedmiotu po nazwie i wywołaniu odpowiedniej dla niego funkcji, która to przejmie konkretną aktualizację wartości *quality*.

Wygląda ona następująco:

```
def quality_update(self, item):
    #Special items' behaviour
    if "Aged Bride" in item.name:
        self.bride_qualit_update(item)
    elif "Sulfuras" in item.name:
        self.sulfuras_quality_update(item)
    return 1
    elif "Backstage passes" in item.name:
        self.backstage_quality_update(item)
    elif "Conjured" in item.name:
        self.conjured_quality_update(item)
    else:
        #Normal items' behaviour
        self.normal_quality_update(item)
```

Funkcje indywidualne dla przedmiotów

W związku z potencjalną potrzebą dodawania przedmiotów w przyszłości do programu postanowiono każdy przedmiot z funkcji `quality_update` wydzielić do osobnej funkcji, dzięki czemu zachowujemy klarowność mając nadal odwołanie się do aktualizacji każdego z nich w jednym miejscu, ale także przenosząc część kodu w inne miejsce - mamy wyższą przejrzystość.

Funkcja zmiany quality: Sulfuras

Z funkcji pierwonej `update_quality` została wyciągnięta i usunięta funkcjonalność dotycząca przedmiotu "Sulfuras" i przeniesiona do funkcji indywidualnej:

```
def sulfuras_quality_update(self, item):
    item.quality = 80
```

Funkcja zmiany quality: Backstage passes

Z funkcji pierwonej `update_quality` została wyciągnięta i usunięta funkcjonalność dotycząca przedmiotu "Backstage passes" i przeniesiona do funkcji indywidualnej:

```
def backstage_quality_update(self, item):
    if item.sell_in >= 10:
        item.quality += 1
    elif 10 > item.sell_in >= 5:
        item.quality += 2
    elif 5 > item.sell_in >= 0:
        item.quality += 3
    elif item.sell_in < 0:
        item.quality = 0
```

Funkcja zmiany quality: Aged Bride

Z funkcji pierwonej `update_quality` została wyciągnięta i usunięta funkcjonalność dotycząca przedmiotu "Aged Bride" i przeniesiona do funkcji indywidualnej:

```
def bride_qualit_update(self, item):
    if item.sell_in >= 0:
        item.quality += 1
    if item.sell_in < 0:
        item.quality += 2
```

Funkcja zmiany quality: Conjured items

Z funkcji pierwotnej `update_quality` została wyciągnięta i usunięta funkcjonalność dotycząca przedmiotu "Conjured items" i przeniesiona do funkcji indywidualnej:

```
def conjured_quality_update(self, item):
    self.normal_quality_update(item)
    self.normal_quality_update(item)
```

Funkcja zmiany quality: normal items

Z funkcji pierwotnej `update_quality` została wyciągnięta i usunięta funkcjonalność dotycząca przedmiotów normalnych i przeniesiona do funkcji indywidualnej:

```
def normal_quality_update(self, item):
```

```
if item.sell_in >= 0:
    item.quality -= 1
elif item.sell_in < 0:
    item.quality -= 2
```

Moduł testowy

Testy aplikacji stworzono od początku ze względu na manualny charakter poprzednich testów. Testy automatyczne programu zostały zaprojektowane w bardzo prosty sposób. Dodają one charakterystyczny przedmiot i aktualizują go sprawdzając zgodność danych w krytycznych punktach.

Nowe testy wyglądają następująco:

```
def update(items, days):
    for day in range(1,days+1):
        GildedRose(items).update_quality()

def testNormalItems():
    items = [Item(name='+5 Dexterity Vest', sell_in = 2, quality = 8)]
    update(items, 2)
    for item in items:
        assert item.quality == 6, 'Normal item, normal rate quality issue'
        assert item.sell_in == 0, 'Normal item, sell_in issue'
    update(items, 2)
    for item in items:
        assert item.quality == 2, 'Normal item, fast rate quality issue'
        assert item.sell_in == -2, 'Normal item, sell_in issue'

def testLegendaryItems():
    items = [Item(name='Sulfuras, Hand of Ragnaros', sell_in = 2, quality = 75)]
    update(items, 2)
    for item in items:
        assert item.quality == 80, 'Legendary item, normal rate quality issue'
        assert item.sell_in == 0, 'Legendary item, sell_in issue'
    update(items, 2)
    for item in items:
        assert item.quality == 80, 'Legendary item, fast rate quality issue'
        assert item.sell_in == 0, 'Legendary item, sell_in issue'

def testAgedBride():
```

```
items = [Item(name='Aged Bride', sell_in = 2, quality = 8)]

update(items, 2)
for item in items:
    assert item.quality == 10, 'Aged Bride item, normal rate quality issue'
    assert item.sell_in == 0, 'Aged Bride item, sell_in issue'
update(items, 2)
for item in items:
    assert item.quality == 14, 'Aged Bride item, fast rate quality issue'
    assert item.sell_in == -2, 'Aged Bride item, sell_in issue'

def testBackstagePasses():
    items = [Item(name='Backstage passes to a TAFKAL80ETC concert', sell_in =

update(items, 2)
for item in items:
    assert item.quality == 10, 'BackstagePasses item, normal rate quality'
    assert item.sell_in == 10, 'BackstagePasses item, sell_in issue'
update(items, 5)
for item in items:
    assert item.quality == 20, 'BackstagePasses item, fast rate quality is'
    assert item.sell_in == 5, 'BackstagePasses item, sell_in issue'
update(items, 5)
for item in items:
    assert item.quality == 35, 'BackstagePasses item, fast rate quality is'
    assert item.sell_in == 0, 'BackstagePasses item, sell_in issue'
update(items, 5)
for item in items:
    assert item.quality == 0, 'BackstagePasses item, fast rate quality iss'
    assert item.sell_in == -5, 'BackstagePasses item, sell_in issue'

def testCojuredItems():
    items = [Item(name='Conjured Mana Cake', sell_in = 2, quality = 12)]

update(items, 2)
for item in items:
    assert item.quality == 8, 'Conjured item, normal rate quality issue'
    assert item.sell_in == 0, 'Conjured item, sell_in issue'
update(items, 2)
for item in items:
    assert item.quality == 0, 'Conjured item, fast rate quality issue'
    assert item.sell_in == -2, 'Conjured item, sell_in issue'

testNormalItems()
testLegendaryItems()
testAgedBride()
testBackstagePasses()
```

```
testCojuredItems()

print('-'*150, 'PASSED', sep = '\n')
```

Refactoring analysis

Dokonano analizy kodu po refaktoryzacji przy użyciu narzędzia *Code Climate* - wyniki poniżej:

| Ogólna ocena: | Duppikaty | Smród kodu | Inne |
|---------------|-----------|------------|------|
| A | 0 | 0 | 0 |

Oceny poszczególnych plików oraz ilość linii kodu:

| Nazwa pliku | linie kodu | ocena |
|-------------------------|------------|-------|
| gilded_rose.py | 59 | A |
| gilded_rose_new_test.py | 69 | A |

Refaktoryzację uznano za pomyślną.