Refactoring

Petr Pošík

Katedra kybernetiky, FEL ČVUT v Praze

OI, B4B33RPH - Řešení problémů a hry, 2017

Refactoring: co to je?

- Změna formátování či organizace zdrojového kódu, která nemění funkci kódu.
- Obvykle za účelem zlepšení kvality (čistoty, čitelnosti, ...) kódu.

Příklad: Gilded Rose

Gilded Rose je malý obchůdek (v počítačové hře), který se zabývá **nákupem a prodejem zboží**. Bohužel, naše **zboží postupně ztrácí kvalitu** s blížícím se koncem trvanlivosti. Vytvořili jsme systém pro aktualizaci stavu skladových zásob (funkce `update_quality()`). Její původní autor už ale naši společnost opustil. Vaším úkolem je **aktualizovat stávající systém**, aby dokázal pracovat s novým typem zboží.

Náš systém: Položky na skladě

```
class Item:

    def __init__(self, name, days_left, quality):
        self.name = name
        self.days_left = days_left
        self.quality = quality
```

- Všechny položky mají vlastnost `days_left`, která udává počet dní zbývajících do konce trvanlivosti.
- Všechny položky mají vlastnost `quality`, která vyjadřuje kvalitu zboží.
- Na konci každého dne náš systém aktualizuje (obvykle sníží) obě hodnoty u každé skladové položky.

Zatím jednoduché, že? Ale teď to teprve začne být zajímavé...

Náš systém: Další vlastnosti

- Jakmile zboží dosáhne svého data trvanlivosti (`days_left==0`), `quality` se snižuje 2x tak rychle.
- Kvalita zboží není nikdy záporná.
- "Aged Brie" (zrající sýr) ale naopak s přibývajícím časem získává na kvalitě.
- Kvalita zboží nemůže být vyšší než 50. Až na následující výjimku:
- "Sulfuras" je legendární předmět, který nikdy neztrácí kvalitu. Jeho `quality` je vždy 80 a nikdy se nemění.
- "Backstage passes" (vstupenky do zákulisí), podobně jako "Aged Brie", s časem získává na kvalitě; `quality` se zvyšuje o 2, když do koncertu zbývá 10 nebo méně dní, a o 3, když do koncertu zbývá 5 nebo méně dní. Po koncertu ale kvalita tohoto zboží klesne na 0.

Váš úkol: zapracovat nový typ zboží

Nedávno jsme podepsali smlouvu s dodavatelem použitého zboží. Jeho zpracování vyžaduje aktualizaci našeho systému:

- "Conjured" (použité) zboží ztrácí kvalitu 2x rychleji než normální zboží.

Můžete udělat jakékoli změny ve funkci `update_quality()` a přidat jakýkoli nový kód za předpokladu, že vše bude fungovat stejně jako dosud. Ale neměňte třídu `Item`, protože ta je použita i v jiných částech našeho systému.

Aktuální implementace

```
def update_quality(items):
    for item in items:
        if item.name != 'Aged Brie' and item.name != 'Backstage passes':
            if item.quality > 0:
                if item.name != "Sulfuras":
                    item.quality = item.quality - 1
        else:
            if item.quality < 50:
                item.quality = item.quality + 1
                if item.name == 'Backstage passes':
                    if item.days_left < 11:
                        if item.quality < 50:
                            item.quality = item.quality + 1
                    if item.days_left < 6:
                        if item.quality < 50:
                            item.quality = item.quality + 1
        if item.name != 'Sulfuras':
            item.days_left = item.days_left - 1
        if item.days_left < 0:
            if item.name != 'Aged Brie':
                if item.name != 'Backstage passes':
                    if item.quality > 0:
                        if item.name != 'Sulfuras':
                            item.quality = item.quality - 1
                else:
                    item.quality = item.quality - item.quality
            else:
                if item.quality < 50:
                    item.quality = item.quality + 1
```

Dojmy z kódu

- Používá kód složité konstrukce či pokročilé vlastnosti Pythonu?
- Přesto - je čistý? Dokážete říct, co se v něm děje?
- Čím tento kód "smrdí"? Co se vám na něm nelíbí?
- Řekli byste, že funguje správně? Jak to ověřit?
- Víte, jak do kódu přidat novou vlastnost?
- Co musíme udělat, abychom novou vlastnost dokázali do kódu přidat?

Další postup

1. Doplnit testy.
   - Zjistit, zda kód funguje podle očekávání.
   - Zajistit nápravu, pokud nefunguje.
   - Ochránit stávající funkčnost.
2. Provést refactoring.
   - Převést kód do čistšího tvaru.
3. Přidat novou funkcionalitu.

Unittest

In [1]:
```
from display import display
display('test_gildedrose_v0.py')
```

```
import unittest
from gildedrose import *

class GildedRoseTest(unittest.TestCase):
    pass

if __name__=='__main__':
    unittest.main()
```

In [2]:
```
%run test_gildedrose_v0.py
```

```
----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
```

Jednotlivé testy

Postupně přidáváme jednotlivé testy, např.:

In [3]:
```
display('test_gildedrose_v1.py', 4, 10)
```

```
class GildedRoseTest(unittest.TestCase):

    def test_name_does_not_change(self):
        item = Item('foo', 0, 0)
        update_quality([item])
        self.assertEqual('foo', item.name)
```

... a v okamžiku, kdy máme pokryté všechny specifikace, vidíme, že žádný test neselhává:

In [4]:
```
%run test_gildedrose_v1.py
```

```
..............
----------------------------------------------------------------------
Ran 14 tests in 0.031s

OK
```

Ačkoli se to zdá skoro neuvěřitelné, původní nečitelná implementace funguje správně.

Refactoring

- Nyní **máme testy** a budeme provádět změny v kódu.
- Testy budeme spouštět tak často, jak to jen jde.
  - Provedeme-li změnu, kvůli které nějaký test selže, víme, že máme něco špatně.
- **Nebojím se změny dělat, protože na chybu snadno přijdu!**

LIVE DEMO

Po mnoha iteracích jsme se mohli dostat např. do následujícího stavu:

In [5]:
```
display('gildedrose_v4.py')
```

```
BACKSTAGE_TRIPLE_LIMIT = 5
BACKSTAGE_DOUBLE_LIMIT = 10

class Item:

    def __init__(self, name, days_left, quality):
        self.name = name
        self.days_left = days_left
        self.quality = quality

def update_quality(items):
    for item in items:
        if item.name == 'Aged Brie':
            update_brie(item)
        elif item.name == 'Backstage passes':
            update_backstage(item)
        elif item.name == 'Sulfuras':
            update_sulfuras(item)
        else:
            update_normal(item)

def update_brie(item):
    item.days_left -= 1
    increase_quality(item)
    if item.days_left < 0:
        increase_quality(item)

def update_backstage(item):
    item.days_left -= 1
    if item.days_left < 0:
        item.quality = 0
        return
    increase_quality(item)
    if item.days_left <= BACKSTAGE_DOUBLE_LIMIT:
        increase_quality(item)
    if item.days_left <= BACKSTAGE_TRIPLE_LIMIT:
        increase_quality(item)

def update_sulfuras(item):
    pass

def update_normal(item):
    item.days_left -= 1
    decrease_quality(item)
    if item.days_left < 0:
        decrease_quality(item)

def decrease_quality(item):
    if item.quality > 0:
        item.quality -= 1

def increase_quality(item):
    if item.quality < 50:
        item.quality += 1
```

Závěr

- Kód stále není ideální, ale je již poměrně čitelný.
- Oddělené funkce, které zpracovávaji jednotlivé druhy zboží.
- Mnoho krátkých jednoúčelových funkcí místo jedné dlouhé, která dělala vše.
- Je zřejmé, kde a jak kód upravit, aby byl schopen pracovat s novým typem zboží!

Testy jsou důležité!

- Bez nich bychom netušili, zda původní implementace funguje.
- Bez nich bychom možná nechápali zcela správně specifikace.
- Bez nich bychom se neodvážili refactoring dělat!

Konfigurace notebooku

Následující ignorujte.

In [6]:
```
%%HTML
<style>
.CodeMirror { min-width: 100% !important; }
</style>
```