



ВТУ "Св. Св. Кирил и Методий"
Факултет "Математика и Информатика"

**КУРСОВА РАБОТА
ПО ДИСЦИПЛИНАТА
Метрики за анализ на качеството на кода
На тема:
РЕФАКТОРИРАНЕ НА ЧЕСТО СРЕЩАНИ CODE
SMELLS**

Изготвил:
Пламенна Викторова Петрова
Специалност:
Софтуерно инженерство
Факултетен № 1909011132

Проверили:
доц. д-р Ивайло
Петров Дончев
Иван Добринов
Пеев

Велико Търново
2021

Съдържание

| | |
|------------------------------------------------|----|
| I. Problem : Gilded Rose Refactoring Kata..... | 4 |
| 1. Кратко описание на примера..... | 4 |
| 2. Първоначален код..... | 5 |
| 3. Причина за рефакторирането..... | 6 |
| 4. Код след рефакторирането..... | 8 |
| 5. Заключение..... | 17 |
| II. Problem : Checking Account..... | 17 |
| 1. Кратко описание на примера..... | 17 |
| 2. Първоначален код..... | 17 |
| 3. Причина за рефакторирането..... | 19 |
| 4. Код след рефакторирането..... | 20 |
| 5. Заключение..... | 21 |
| III. Problem : Animals Gone Wrong..... | 22 |
| 1. Кратко описание на примера..... | 22 |
| 2. Първоначален код..... | 22 |
| 3. Причина за рефакторирането..... | 24 |
| 4. Код след рефакторирането..... | 24 |
| 5. Заключение..... | 25 |
| IV. Problem : Savings Account..... | 26 |
| 1. Кратко описание на примера..... | 26 |
| 2. Първоначален код..... | 26 |

| | | |
|-----------------------------------|---------------------------------|----|
| 3. | Причина за рефакторирането..... | 27 |
| 4. | Код след рефакторирането..... | 28 |
| 5. | Заклучение..... | 29 |
| V. Problem : Phone Formatter..... | | 29 |
| 1. | Кратко описание на примера..... | 29 |
| 2. | Първоначален код..... | 30 |
| 3. | Причина за рефакторирането..... | 31 |
| 4. | Код след рефакторирането..... | 31 |
| 5. | Заклучение..... | 33 |

I. Problem : Gilded Rose Refactoring Kata

1. Кратко описание на примера

Проблемът представлява известна code kata, която може да се дефинира като упражнение в програмирането, целящо да помогне за подобряването на coding уменията чрез практика и повторение. Взема името си от тренировъчен подход в японските бойни изкуства. Конкретно за случая : симулирана е фентъзи странноприемница, предлагаща търговски услуги. Те включват :

- продажбата на item-и със sell-in стойност, обозначена от броя на дните, за които даденият item трябва да бъде продаден
- quality стойност, обозначаваща колко ценен е item-ът
- в края на работния ден системата понижава стойността на всеки item

Освен това са включени и допълнителните условия :

- Веднъж когато мине срокът за продажба в определен ден, quality-то деградира двойно по-бързо
- Quality-то на item-ите никога не е отрицателно
- "Aged Brie" нараства по Quality-и, колкото повече остарява
- Quality-то на даден item никога не е повече от 50
- "Sulfuras" като legendary item трябва да се продаде възможно най-бързо или неговото Quality драстично спада
- "Backstage passes" досъщ като Aged Brie нараства по Quality с приближаването на SellIn стойността му – Quality-то се увеличава с 2, когато са останали 10 или по-малко дни и с 3, когато дните са по-малко или равно на 5, но Quality-то спада на 0 след концерта
- "Conjured" item-ите деградират по Quality-и двойно по-бързо от нормалните item-и

Забележка : въпреки че по принцип item-ите не могат да имат Quality, надвишаващо 50, "Sulfuras" е легендарен item и се разглежда като изключение

Предоставен е код за рефакториране в UpdateQuality() метода, като целта е той да се оптимизира и да се запази същият резултат както при оригинала.

2. Първоначален код

```
16 1reference
17 public void UpdateQuality()
18 {
19     for (var i = 0; i < Items.Count; i++)
20     {
21         if (Items[i].Name != "Aged Brie" && Items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
22         {
23             if (Items[i].Quality > 0)
24             {
25                 if (Items[i].Name != "Sulfuras, Hand of Ragnaros")
26                 {
27                     Items[i].Quality = Items[i].Quality - 1;
28                 }
29             }
30         }
31         else
32         {
33             if (Items[i].Quality < 50)
34             {
35                 Items[i].Quality = Items[i].Quality + 1;
36             }
37             if (Items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
38             {
39                 if (Items[i].SellIn < 11)
40                 {
41                     if (Items[i].Quality < 50)
42                     {
43                         Items[i].Quality = Items[i].Quality + 1;
44                     }
45                 }
46                 if (Items[i].SellIn < 6)
47                 {
48                     if (Items[i].Quality < 50)
49                     {
50                         Items[i].Quality = Items[i].Quality + 1;
51                     }
52                 }
53             }
54         }
55     }
56 }
```

```

54     }
55 }
56
57 if (Items[i].Name != "Sulfuras, Hand of Ragnaros")
58 {
59     Items[i].SellIn = Items[i].SellIn - 1;
60 }
61
62 if (Items[i].SellIn < 0)
63 {
64     if (Items[i].Name != "Aged Brie")
65     {
66         if (Items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
67         {
68             if (Items[i].Quality > 0)
69             {
70                 if (Items[i].Name != "Sulfuras, Hand of Ragnaros")
71                 {
72                     Items[i].Quality = Items[i].Quality - 1;
73                 }
74             }
75         }
76         else
77         {
78             Items[i].Quality = Items[i].Quality - Items[i].Quality;
79         }
80     }
81     else
82     {
83         if (Items[i].Quality < 50)
84         {
85             Items[i].Quality = Items[i].Quality + 1;
86         }
87     }
88 }
89 }
90 }
91

```

3. Причина за рефакторирането

Методът `UpdateQuality()` е типичен пример за Code Smell-a Long Method. Той съдържа прекалено редове код с дубликации – общо 76, което го прави трудно четим и разбираем. Създава се усещането, че в метода постоянно е натрупвана нова логика на посоки без комплексен поглед върху цялостната идея, което с течение на времето е довело до значителното разрастване на размерите му, а вложените `if else` конструкции само допринасят за постоянното увеличаване на цикломатичната му сложност, която е достигнала 19. Тя се проверява чрез вградения инструмент за измерване на метриките на кода за Visual Studio от Analyze -> Calculate Code Metrics менюто.

```

7 2 references
8 public class GildedRoseOriginal
9 {
10     private IList<Item> Items;
11
12     1 reference
13     public GildedRoseOriginal(IList<Item> Items)
14     {
15         this.Items = Items;
16     }
17 }

```

| Hierarchy | Maintainability Index | Cyclomatic Complexity | Depth of Inheritance | Class Coupling | Lines of Source code | Lines of Executable code |
|---------------------------------------|-----------------------|-----------------------|----------------------|----------------|----------------------|--------------------------|
| P01_GildedRoseRefactoringKata (Debug) | 68 | 32 | 1 | 5 | 194 | 39 |
| P01_GildedRoseRefactoringKata | 68 | 32 | 1 | 5 | 194 | 39 |
| GildedRoseOriginal | 54 | 20 | 1 | 2 | 85 | 26 |
| Items : IList<Item> | 100 | 0 | | 2 | 1 | 0 |
| GildedRoseOriginal(IList<Item>) | 96 | 1 | | 2 | 5 | 1 |
| UpdateQuality() : void | 45 | 19 | | 2 | 76 | 25 |
| Program | 58 | 3 | 1 | 5 | 73 | 9 |
| Item | 92 | 9 | 1 | 0 | 24 | 4 |

Според таблицата за цикломатична сложност, стойността за UpdateQuality() метода почти се доближава до графата за много сложен код, който трудно може да се тества и поддържа :

| Complexity Number | Meaning |
|-------------------|----------------------------------|
| 1-10 | Structured and well written code |
| | High Testability |
| | Cost and Effort is less |
| 10-20 | Complex Code |
| | Medium Testability |
| | Cost and effort is Medium |
| 20-40 | Very complex Code |
| | Low Testability |
| | Cost and Effort are high |
| >40 | Not at all testable |
| | Very high Cost and Effort |

Вижда се и, че Maintainability-и index-ът на UpdateQuality е 45, а редовете на Executable code-а са тройно по-малко в сравнение с тези на source code-а. Другата очевадна грешка е използването на Pascal Case naming конвенцията за IList<Item> field-а. Прието е в C# filed-овете, променливите и параметрите да са съобразени с Camel Case правилата за именуване.

Extension-ът за анализ на качеството на кода SonarLint пък закача redirect-и към няколко warning-а : за добавяне на readonly ключовата дума в при дефинирането на полето, за merge-ване на най-дълбоко нестнатите if statement-и с enclose-ващите им както и за подобряване на предпоследния expression, където всички участници носят едно и също наименование.

След кликване върху code issue-то до warning-а се отваря съответстващата част от документацията на SonarLint заедно с примери за несъвместим код и неговото решение за съвместимост. Например :

Collapsible "if" statements should be merged

Analyze your code

Code Smell Major clumsy

Merging collapsible if statements increases the code's readability.

Noncompliant Code Example

```
if (condition1)
{
    if (condition2)
    {
        // ...
    }
}
```

Compliant Solution

```
if (condition1 && condition2)
{
    // ...
}
```

Available in:
 | |

4. Код след рефакторирането

Постъпково описание на извършените по кода промени :

При изпълнение на подсказките от SonarLint кодът се редуцира на дължина от 76 реда на 64 за (source code) и 21 реда (executable code), обаче цикломатичната му сложност се запазва.

```
0 references
16 public void UpdateQuality()
17 {
18     for (var i = 0; i < items.Count; i++)
19     {
20         if (items[i].Name != "Aged Brie" && items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
21         {
22             if (items[i].Quality > 0 && items[i].Name != "Sulfuras, Hand of Ragnaros")
23             {
24                 items[i].Quality = items[i].Quality - 1;
25             }
26         }
27         else
28         {
29             if (items[i].Quality < 50)
30             {
31                 items[i].Quality = items[i].Quality + 1;
32             }
33             if (items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
34             {
35                 if (items[i].SellIn < 11 && items[i].Quality < 50)
36                 {
37                     items[i].Quality = items[i].Quality + 1;
38                 }
39                 if (items[i].SellIn < 6 && items[i].Quality < 50)
40                 {
41                     items[i].Quality = items[i].Quality + 1;
42                 }
43             }
44         }
45     }
46 }
47
48 if (items[i].Name != "Sulfuras, Hand of Ragnaros")
49 {
50     items[i].SellIn = items[i].SellIn - 1;
51 }
52
```

```

53     if (items[i].SellIn < 0)
54     {
55         if (items[i].Name != "Aged Brie")
56         {
57             if (items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
58             {
59                 if (items[i].Quality > 0 && items[i].Name != "Sulfuras, Hand of Ragnaros")
60                 {
61                     items[i].Quality = items[i].Quality - 1;
62                 }
63             }
64             else
65             {
66                 items[i].Quality = 0;
67             }
68         }
69         else
70         {
71             if (items[i].Quality < 50)
72             {
73                 items[i].Quality = items[i].Quality + 1;
74             }
75         }
76     }
77 }
78 }
79 }
80 }
81

```

Намаляваме дължината на метода чрез изнасяне на повтарящ се код в нови методи. Използва се рефакторинг техниката, позната като Extract Method.

Откриват се две дубликации на :

```

if (items[i].Quality > 0 && items[i].Name != "Sulfuras, Hand of Ragnaros")
{
    items[i].Quality = items[i].Quality - 1;
}

```

Те се заместват чрез извикването на метода `DecrementQualityForNormalItems(int itemIndex)`, който приема за параметър `index`-а на съответния `item` :

2 references

```
private void DecrementQualityForNormalItems(int itemIndex)
{
    if (items[itemIndex].Quality > 0 && items[itemIndex].Name != "Sulfuras, Hand of Ragnaros")
    {
        items[itemIndex].Quality = items[itemIndex].Quality - 1;
    }
}
```

След заместването :

0 references

```
24 public void UpdateQuality()
25 {
26     for (var i = 0; i < items.Count; i++)
27     {
28         if (items[i].Name != "Aged Brie" && items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
29         {
30             DecrementQualityForNormalItems(i);
31         }
32         else
33         {
34             if (items[i].Quality < 50)
35             {
36                 items[i].Quality = items[i].Quality + 1;
37
38                 if (items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
39                 {
40                     if (items[i].SellIn < 11 && items[i].Quality < 50)
41                     {
42                         items[i].Quality = items[i].Quality + 1;
43                     }
44
45                     if (items[i].SellIn < 6 && items[i].Quality < 50)
46                     {
47                         items[i].Quality = items[i].Quality + 1;
48                     }
49                 }
50             }
51         }
52
53         if (items[i].Name != "Sulfuras, Hand of Ragnaros")
54         {
55             items[i].SellIn = items[i].SellIn - 1;
56         }
57     }
}
```

```

57
58     if (items[i].SellIn < 0)
59     {
60         if (items[i].Name != "Aged Brie")
61         {
62             if (items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
63             {
64                 DecrementQualityForNormalItems(i);
65             }
66         }
67         else
68         {
69             items[i].Quality = 0;
70         }
71     }
72     else
73     {
74         if (items[i].Quality < 50)
75         {
76             items[i].Quality = items[i].Quality + 1;
77         }
78     }
79 }
80 }
81 }
82 }
83

```

Следващата стъпка е да пренесе друг припокриващ се код в метод за инкрементиране на Quality-то. Засечен е патърнът :

```

if (items[i].Quality < 50)
{
    items[i].Quality = items[i].Quality + 1;
}

```

Част от него бива преместен във външния му if като допълнителен condition, но за целта на метода се връща първоначалният му вид :

```

3 references
private void IncrementQuality(int itemIndex)
{
    if (items[itemIndex].Quality < 50)
    {
        items[itemIndex].Quality = items[itemIndex].Quality + 1;
    }
}

```

Оптимизираният UpdateQuality() метод :

```

0 references
32 public void UpdateQuality()
33 {
34     for (var i = 0; i < items.Count; i++)
35     {
36         if (items[i].Name != "Aged Brie" && items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
37         {
38             DecrementQualityForNormalItems(i);
39         }
40         else
41         {
42             if (items[i].Quality < 50)
43             {
44                 items[i].Quality = items[i].Quality + 1;
45
46                 if (items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
47                 {
48                     if (items[i].SellIn < 11)
49                     {
50                         IncrementQuality(i);
51                     }
52
53                     if (items[i].SellIn < 6)
54                     {
55                         IncrementQuality(i);
56                     }
57                 }
58             }
59         }
60
61         if (items[i].Name != "Sulfuras, Hand of Ragnaros")
62         {
63             items[i].SellIn = items[i].SellIn - 1;
64         }

```

```

65
66         if (items[i].SellIn < 0)
67         {
68             if (items[i].Name != "Aged Brie")
69             {
70                 if (items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
71                 {
72                     DecrementQualityForNormalItems(i);
73                 }
74                 else
75                 {
76                     items[i].Quality = 0;
77                 }
78             }
79             else
80             {
81                 IncrementQuality(i);
82             }
83         }
84     }
85 }
86
87 }
88

```

На 42 и 43 ред се вижда, че снипетите участват в IncrementQuality(int itemIndex) метода. Можем да се лишим от външния if, защото сме сигурни, че това няма да доведе до нежелани последствия заради същата разписана логика в изнесения метод. Получаваме :

```
36     if (items[i].Name != "Aged Brie" && items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
37     {
38         DecrementQualityForNormalItems(i);
39     }
40     else
41     {
42         IncrementQuality(i);
43
44         if (items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
45         {
46             if (items[i].SellIn < 11)
47             {
48                 IncrementQuality(i);
49             }
50
51             if (items[i].SellIn < 6)
52             {
53                 IncrementQuality(i);
54             }
55         }
56     }
```

Преглеждаме кода за блокове, които смислово вървят заедно и ги extract-ме в нови методи :

```
1 reference
32     private void UpdateQualityForItemsThatAgeWell(int itemIndex)
33     {
34         IncrementQuality(itemIndex);
35
36         if (items[itemIndex].Name == "Backstage passes to a TAFKAL80ETC concert")
37         {
38             if (items[itemIndex].SellIn < 11)
39             {
40                 IncrementQuality(itemIndex);
41             }
42
43             if (items[itemIndex].SellIn < 6)
44             {
45                 IncrementQuality(itemIndex);
46             }
47         }
48     }
```

```
0 references
50 public void UpdateQuality()
51 {
52     for (var i = 0; i < items.Count; i++)
53     {
54         if (items[i].Name != "Aged Brie" && items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
55         {
56             DecrementQualityForNormalItems(i);
57         }
58         else
59         {
60             UpdateQualityForItemsThatAgeWell(i);
61         }
62
63         if (items[i].Name != "Sulfuras, Hand of Ragnaros")
64         {
65             items[i].SellIn = items[i].SellIn - 1;
66         }
67
68         if (items[i].SellIn < 0)
69         {
70             if (items[i].Name != "Aged Brie")
71             {
72                 if (items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
73                 {
74                     DecrementQualityForNormalItems(i);
75                 }
76                 else
77                 {
78                     items[i].Quality = 0;
79                 }
80             }
81             else
82             {
83                 IncrementQuality(i);
84             }
85         }
86     }
87 }
```

```
1 reference
50 private void UpdateQualityForExpiredItems(int itemIndex)
51 {
52     if (items[itemIndex].Name != "Aged Brie")
53     {
54         if (items[itemIndex].Name != "Backstage passes to a TAFKAL80ETC concert")
55         {
56             DecrementQualityForNormalItems(itemIndex);
57         }
58         else
59         {
60             items[itemIndex].Quality = 0;
61         }
62     }
63     else
64     {
65         IncrementQuality(itemIndex);
66     }
67 }
68 }
```

```
0 references
69 public void UpdateQuality()
70 {
71     for (var i = 0; i < items.Count; i++)
72     {
73         if (items[i].Name != "Aged Brie" && items[i].Name != "Backstage passes to a TAFKAL80ETC concert")
74         {
75             DecrementQualityForNormalItems(i);
76         }
77         else
78         {
79             UpdateQualityForItemsThatAgeWell(i);
80         }
81
82         if (items[i].Name != "Sulfuras, Hand of Ragnaros")
83         {
84             items[i].SellIn = items[i].SellIn - 1;
85         }
86
87         if (items[i].SellIn < 0)
88         {
89             UpdateQualityForExpiredItems(i);
90         }
91     }
92 }
93 }
94 }
95 }
```

И последно – хардкоднатите стрингове се replace-ват с константи :

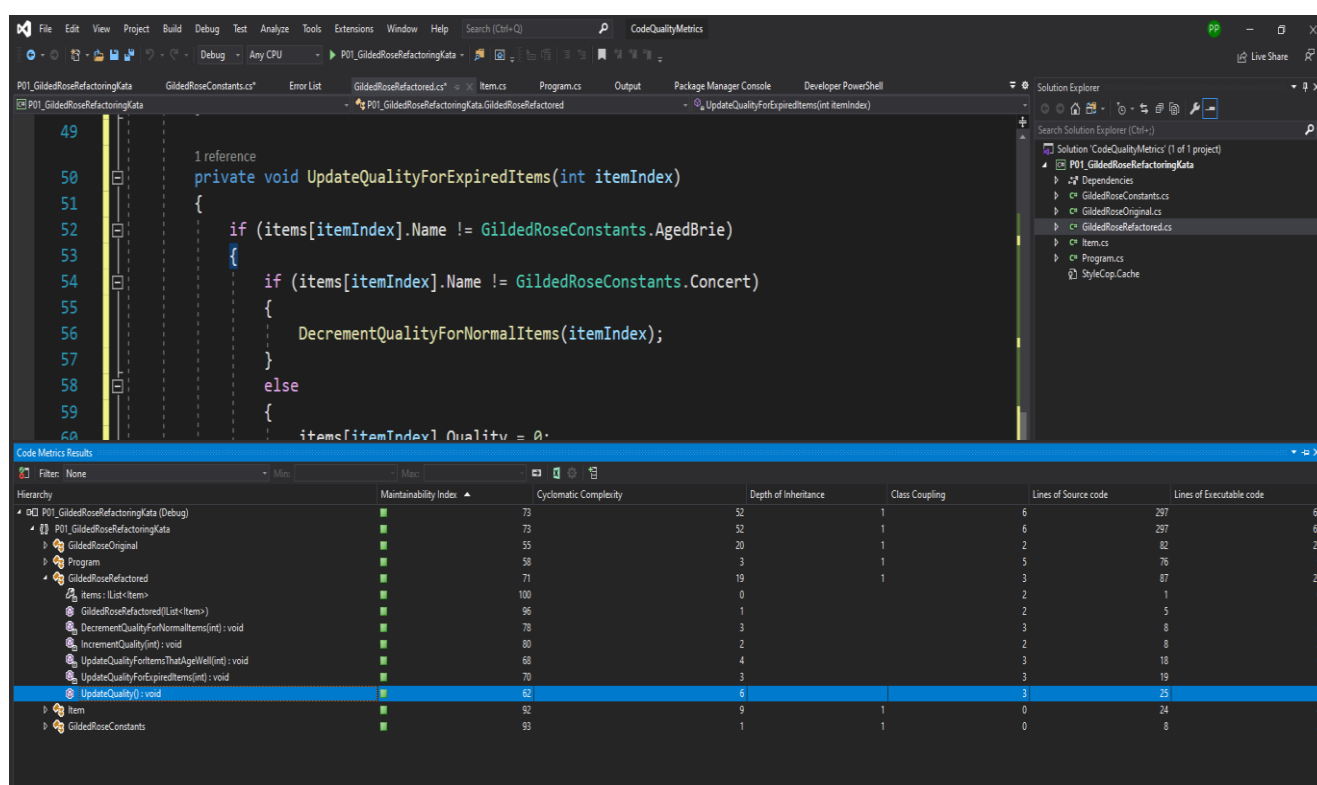
```
0 references
public void UpdateQuality()
{
    for (var i = 0; i < items.Count; i++)
    {
        if (items[i].Name != GildedRoseConstants.AgedBrie && items[i].Name != GildedRoseConstants.Concert)
        {
            DecrementQualityForNormalItems(i);
        }
        else
        {
            UpdateQualityForItemsThatAgeWell(i);
        }

        if (items[i].Name != GildedRoseConstants.Sulfuras)
        {
            items[i].SellIn = items[i].SellIn - 1;
        }

        if (items[i].SellIn < 0)
        {
            UpdateQualityForExpiredItems(i);
        }
    }
}
```


5. Заключение

След направените промени методът става много по-къс и лесен за разбиране. С помощта на Extract Method техниката голяма част логиката от UpdateQuality() се преорганизира в отделени методи, които се извикват на необходимите места. По този начин се наблюдава драстично спадане на цикломатичната сложност – от 19 на 6 пункта, което е 3 пъти по-малко от преди. Редовете на source и executable кода също намаляват тройно. Maintainability Index-ът е леко повишен, но за сметка на това всички останали Code Metrics показатели са подобрени.



II. Problem : Checking Account

1. Кратко описание на примера

Разписан е CheckingAccount клас, съдържащ набор от пропърти, които го характеризират : AccountNumber, CustomerName, Email, Address, ZipCode, City, State, Country, SocialSecurityNumber, ActiveDate както и метод за проверка за последните четири цифри на SSN-а.

2. Първоначален код

```
7 1 reference
8 public class CheckingAccountOriginal
9 {
10     1 reference
11     public string AccountNumber { get; set; }
12
13     1 reference
14     public string CustomerName { get; set; }
15
16     1 reference
17     public string Email { get; private set; }
18
19     1 reference
20     public string Address { get; set; }
21
22     1 reference
23     public int ZipCode { get; set; }
24
25     1 reference
26     public string City { get; set; }
27
28     1 reference
29     public string State { get; set; }
30
31     1 reference
32     public string Country { get; set; }
33
34     6 references
35     public string SocialSecurityNumber { get; set; }
36
37     1 reference
38     public DateTime ActiveDate { get; set; }
39
40     0 references
41     public string GetSSNLast4Digits()
42     {
43         int index = SocialSecurityNumber.LastIndexOf("-", StringComparison.Ordinal);
44         return index > 0 && index < SocialSecurityNumber.Length
45             ? SocialSecurityNumber.Substring(index + 1, SocialSecurityNumber.Length)
46             : SocialSecurityNumber;
47     }
48 }
```

Конструкторът на класа :

```
public CheckingAccountOriginal(string accountNumber, string customerName,
string email, string address, int zipCode, string city, string state, string country,
string socialSecuriyNumber, DateTime activeDate)
{
    AccountNumber = accountNumber;
    CustomerName = customerName;
    Email = email;
    Address = address;
    ZipCode = zipCode;
    City = city;
    State = state;
    Country = country;
    SocialSecurityNumber = socialSecuriyNumber;
    ActiveDate = activeDate;
}
```

3. Причина за рефакторирането

В примера се откриват Bloaters Code Smell-овете Primitive Obsession и Long Parameter List. Първият изпъква, когато дивелъпърите опитват да използват примитиви за дефинируеми прости домейн модели. Стига се до обесивно поставяне на примитиви без мярка. Примитивите представляват реалните градивни блокове на класовете, като example-и за примитивни типове данни са : int, bool, short, double, char, float, string и т.н. Чести сценарии, които стават причина за появата на Primitive Obsession, могат да бъдат :

- употребата на екзесивни примитиви за дадена дефиниция на клас и неговия scope
- употребата на константи
- дубликация на примитиви и др.

Конкретно за класа CheckingAccount. Първоначално са добавени пропъртите за най-важната информация за акаунта – номер, име и имейл на клиента, дата за активност. В последствие броят на пропъртите нараства с още пет, като новите пропъртите са описателни за адреса и локацията на клиента. Това е така, защото не е правилно адресът да бъде репрезентиран само от един стринг, тъй като той е много по-комплексен. И най-накрая са прибавени едно пропърти и метод за извличане на сензитивна за клиента информация, адресираща Social Security. В един момент се оказва, че в класа са възникнали редица проблеми, между които са :

- Примитивите, отнасящи се до адреса, нарушават SOLID Single Responsibility принципа, според който един клас трябва да има single responsibility, encapsulate-нат за него
- Ако Address related пропъртите са необходими на други места в приложението, ще се получи дубликация на код.
- Изнасянето или форматирането на SocialSecurityNumber логиката се различава от поведението на класа. За да се извика GetLast4Digits() метода, трябва да се инстанцира CheckingAccount класа, което е ненужно.
- SocialSecurityNumber в качеството си на сензитивна информация трябва да бъде криптиран
- Може да се обособи и примитивна зависимост за клиента – име и имейл

За втория Code Smell Long Parameter List показателен е списъкът от параметри в конструктора на класа. Те наброяват 10, което е твърде много за един конструктор.

4. Код след рефакторирането

Отделните смислово обединени пропърти се изнасят в собствени класове. После те се bind-ват в CheckingAccount класа, а съобразно Introduce Parameter Object техниката в конструктора някои от параметрите се заместват с обект.

```
7 | 4 references
8 | public class Customer
9 | {
10 |     0 references
11 |     public string Name { get; set; }
12 |
13 |     0 references
14 |     public string Email { get; private set; }
15 | }
16 |
17 | 3 references
18 | public class Address
19 | {
20 |     0 references
21 |     public int ZipCode { get; set; }
22 |
23 |     0 references
24 |     public string City { get; set; }
25 |
26 |     0 references
27 |     public string State { get; set; }
28 |
29 |     0 references
30 |     public string Country { get; set; }
31 | }
32 |
33 | 2 references
34 | public class SocialSecurity
35 | {
36 |     5 references
37 |     public string SocialSecurityNumber { get; set; }
38 |
39 |     0 references
40 |     public string GetSSNLast4Digits()
41 |     {
42 |         int index = SocialSecurityNumber.LastIndexOf("-", StringComparison.Ordinal);
43 |         return index > 0 && index < SocialSecurityNumber.Length
44 |             ? SocialSecurityNumber.Substring(index + 1, SocialSecurityNumber.Length)
45 |             : SocialSecurityNumber;
46 |     }
47 | }
```

```

1 reference
public class CheckingAccountRefactored
{
    1 reference
    public string AccountNumber { get; set; }

    1 reference
    public Customer Customer { get; set; }

    1 reference
    public Address Address { get; set; }

    1 reference
    public DateTime ActiveDate { get; set; }

    1 reference
    public SocialSecurity SocialSecurity { get; set; }

    0 references
    public CheckingAccountRefactored(string accountNumber, Customer customer, Address address, DateTime activeDate, SocialSecurity socialSecurity)
    {
        AccountNumber = accountNumber;
        Customer = customer;
        Address = address;
        ActiveDate = activeDate;
        SocialSecurity = socialSecurity;
    }
}

```

5. Заключение

След приложените промени се постига decouple-ване на CheckingAccount класа, като SOLID принципът Single Responsibility е спазен. Въведени са нови класове Customer, Address и SocialSecurity, които използвани като Object пропъртите могат да намерят употреба и в други класове освен споменатия. По този начин се избягва дубликация на код. GetSSNLast4Digits() методът се енкапсулира за SocialSecurity класа и бива обвързан с обекти от този клас. Броят на параметрите в конструктора спада на половина заради Introduce Parameter Object способа. Въпреки че пет параметъра стоят на границата за допустимост, е възможно за в бъдеще да се осъществи и constructor overloading с прилежащи field-ове с цел да се избере подходящ конструктор според случая без наслагване на излишни параметри, а за останалите пропъртите да бъдат записани стойности допълнително, при което конструкторът с пет параметъра ще бъде най-разширеният.

Например :

```

42
50 1 reference
    public CheckingAccountRefactored(string accountNumber, DateTime activeDate)
51  {
52      AccountNumber = accountNumber;
53      ActiveDate = activeDate;
54  }
55
56 1 reference
    public CheckingAccountRefactored(string accountNumber, DateTime activeDate, Customer customer)
57      : this(accountNumber, activeDate)
58  {
59      Customer = customer;
60  }
61
62 1 reference
    public CheckingAccountRefactored(string accountNumber, DateTime activeDate, Customer customer, Address address)
63      : this(accountNumber, activeDate, customer)
64  {
65      Address = address;
66  }
67
68 0 references
    public CheckingAccountRefactored(string accountNumber, Customer customer, Address address, DateTime activeDate, SocialSecurity socialSecurity)
69      : this(accountNumber, activeDate, customer, address)
70  {
71      SocialSecurity = socialSecurity;
72  }
73 }
74
75

```

III. Problem : Animals Gone Wrong

1. Кратко описание на примера

Представени са класовете `Animal`, играещ ролята на parent class и наследниците му `Cat` и `Fish`. `Animal` съдържа едно пропърти `Breed`, което се подава на конструктора му както и един `protected virtual` метод `Walk()`, който извежда съобщение на конзолата. `Cat` класът се доизгражда като се добавя пропърти `Name` и се `override`-ва методът. `Fish` пък наследява всички базови функционалности и хвърля `NotImplementedException` в `Walk()`.

2. Първоначален код

```

7 0 references
8 public class AnimalsGoneWrongOriginal
9 {
10     3 references
11     public class Animal
12     {
13         1 reference
14         public string Breed { get; set; }
15
16         2 references
17         public Animal(string breed)
18         {
19             Breed = breed;
20         }
21
22         2 references
23         protected virtual void Walk()
24         {
25             Console.WriteLine("An animal is walking");
26         }
27     }
28 }

```

```

24 1 reference
25 public class Cat : Animal
26 {
27     2 references
28     public string Name { get; set; }
29
30     0 references
31     public Cat(string breed, string name)
32     : base(breed)
33     {
34         Name = name;
35     }
36
37     2 references
38     protected override void Walk()
39     {
40         Console.WriteLine($"The cat {Name} is walking");
41     }
42 }
43
44 1 reference
45 public class Fish : Animal
46 {
47     0 references
48     public Fish(string breed)
49     : base (breed)
50     {
51     }
52
53     2 references
54     protected override void Walk()
55     {
56         throw new NotImplementedException();
57     }
58 }
59 }

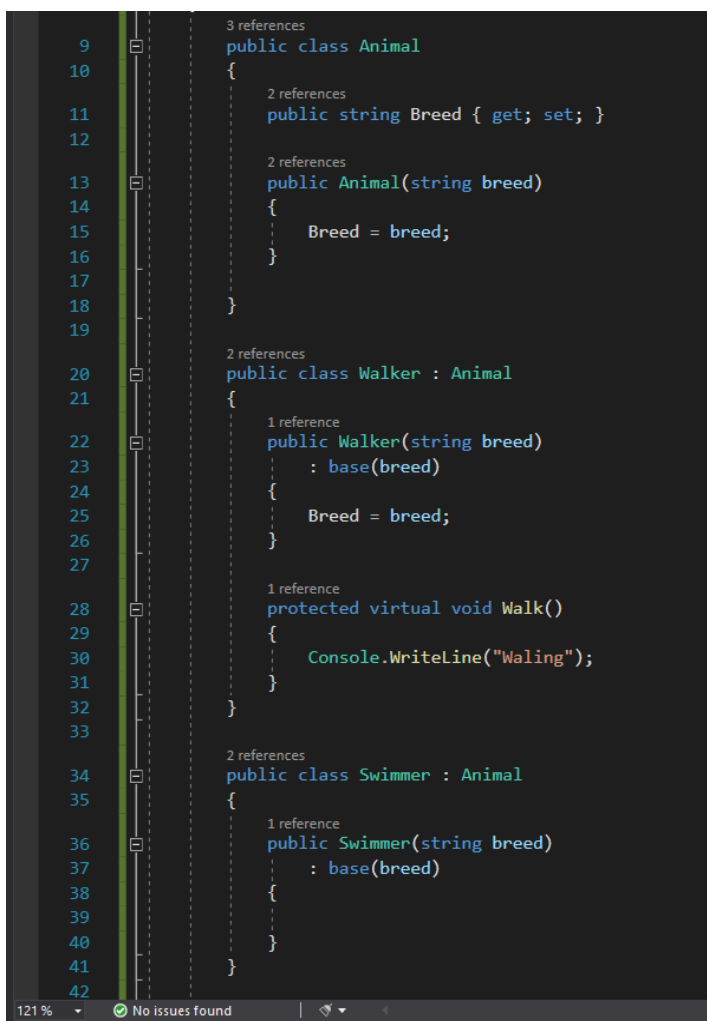
```

3. Причина за рефакторирането

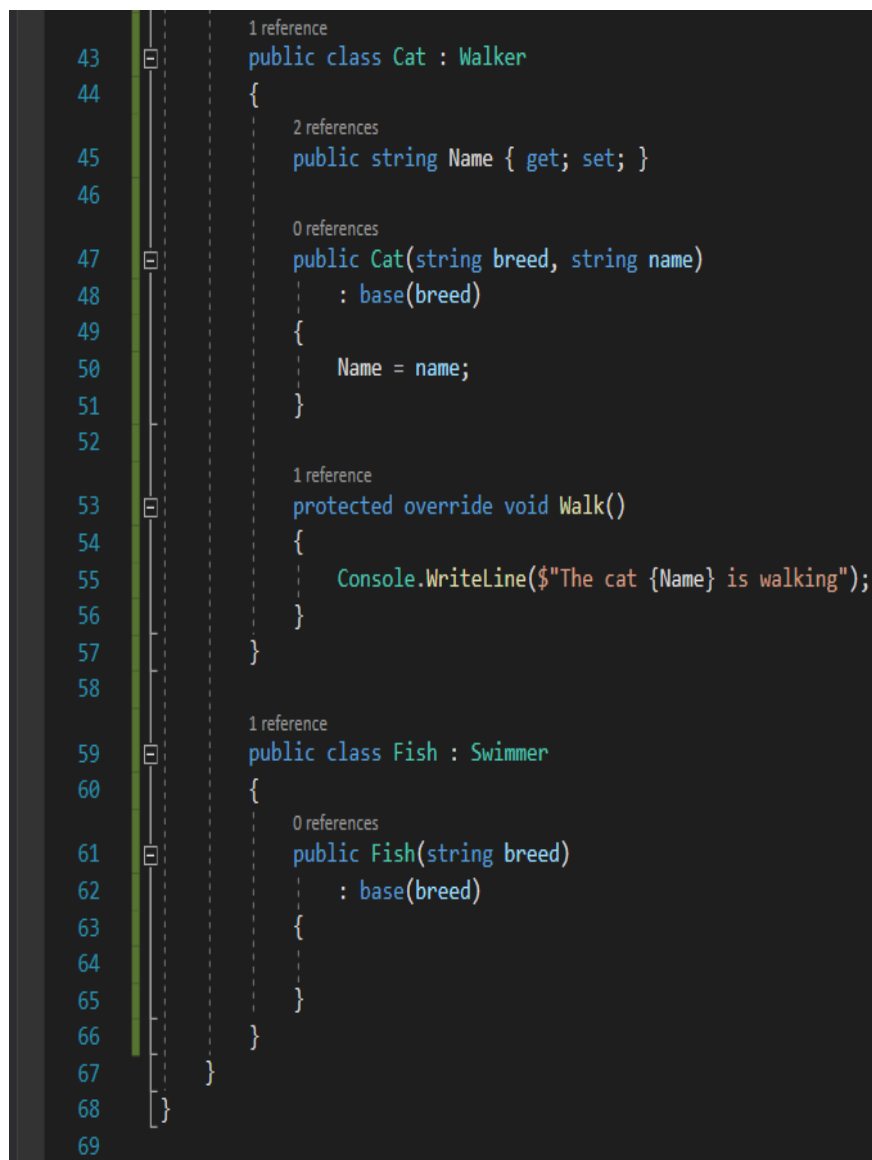
Чисто смислово погледнато една риба не може да ходи, тя плува във водата. Затова във метода Walk() се throw-ва exception. От гледна точка на рефакторинга този проблем е проявление на Code Smell-а Refuse Bequest, същността на който се състои в наследяването на базов клас, не използващ нито един от наследените си фийлдове, пропъртите или методи, като последните могат да бъдат оставени с празни body-та или с NotImplementedException-и. Наследяването трябва да се прилага, когато даден клас иска да използва код от родителския му клас. Ако логиката на класовете започне да се отклонява за всеки от тях и наследниците останат с неимплементирани функционалности, йерархията престава да бъде кохезивна и се разбива.

4. Код след рефакторирането

Решението е да се изнесат отделни супер класове за Cat и Fish :



```
9 | public class Animal
10 | {
11 |     public string Breed { get; set; }
12 |
13 |     public Animal(string breed)
14 |     {
15 |         Breed = breed;
16 |     }
17 |
18 | }
19 |
20 | public class Walker : Animal
21 | {
22 |     public Walker(string breed)
23 |         : base(breed)
24 |     {
25 |         Breed = breed;
26 |     }
27 |
28 |     protected virtual void Walk()
29 |     {
30 |         Console.WriteLine("Waling");
31 |     }
32 |
33 | }
34 | public class Swimmer : Animal
35 | {
36 |     public Swimmer(string breed)
37 |         : base(breed)
38 |     {
39 |     }
40 | }
41 |
42 |
```

5. Заключение

Базирайки се на похвата Extract Superclass създаваме два нови класа Walker и Swimmer, при което премахваме Walk() метода от Animal класа и го поставяме в Walker. По-нататък Cat наследява Walker, а Fish – Swimmer. Следователно Code Smell-ът Refused Bequest, принадлежащ към групата Object-Orientation Abusers, е избегнат, кодовата организация и яснота са подобрени, а inheritance йерархията е логически обособена.

IV. Problem : Savings Account

1. Кратко описание на примера

В класа `SavingsAccount` са включени `private` полетата `accountNumber` и `balance`, които са включени като параметри в конструктора му. Разписани са `public void` методите `Withdraw(double amountToWithdraw)`, `Transfer(double amountToTransfer, SavingsAccount recepientAccount)`, `ProcessFee(double fee)` както и `private` метода `NotifyAccountHolder(string warningMessage)`. Хардкодването на стрингове е предотвратено чрез употребата на константи.

2. Първоначален код

```
7 public class SavingsAccountOriginal
8 {
9     private string accountNumber;
10
11     private double balance;
12
13     public SavingsAccountOriginal(string accountNumber, double balance)
14     {
15         this.accountNumber = accountNumber;
16         this.balance = balance;
17     }
18
19     private void NotifyAccountHolder(string warningMessage)
20     {
21         Console.WriteLine(warningMessage);
22     }
23
24     public void Withdraw(double amountToWithdraw)
25     {
26         if (balance < SavingsAccountConstants.MinimumBalance)
27         {
28             NotifyAccountHolder(SavingsAccountConstants.WithdrawalMinimumBalanceWarning);
29         }
30         else
31         {
32             balance -= amountToWithdraw;
33             Console.WriteLine($"Balance after withdrawal of {amountToWithdraw} : {balance}");
34         }
35     }
36 }
```



3. Причина за рефакторирането

На пръв поглед класът `SavingsAccount` не изглежда толкова хаотичен, но проблемът тук се изразява в разпространението на подобен код и по-точно проверката за минимално равнене на сметката.

```
if (balance < SavingsAccountConstants.MinimumBalance)
```

```
{
```

```
    // ...
```

```
}
```

Въпросната проверка се изпълнява на три места. Ако сметнем за нужно да добавим нови промени по кода, трябва да ги приложим и на трите, което е и предпоставката за откриването на `Code Smell-a Shotgun Surgery` от категорията `Change Preventers`. Характерно за него е извършването на едни и същи или сходни корекции на много различни места в `codebase-a`.

Въпреки че при SavingsAccount не целият код е идентичен за всеки разгледан случай, същинската му логика е. Ето защо класът има нужда от рефакторинг.

4. Код след рефакторирането

Проверката за minimum balance се изнася в private bool метода IsBalanceUnderMinimum(). Методът от своя страна се извиква в съответните if-ове на public методите по-долу.

```
7  public class SavingsAccountRefactored
8  {
9      private string accountNumber;
10
11     private double balance;
12
13     0 references
14     public SavingsAccountRefactored(string accountNumber, double balance)
15     {
16         this.accountNumber = accountNumber;
17         this.balance = balance;
18     }
19
20     3 references
21     private void NotifyAccountHolder(string warningMessage)
22     {
23         Console.WriteLine(warningMessage);
24     }
25
26     3 references
27     private bool IsBalanceUnderMinimum()
28     {
29         return balance < SavingsAccountConstants.MinimumBalance;
30     }
31
32     0 references
33     public void Withdraw(double amountToWithdraw)
34     {
35         if (IsBalanceUnderMinimum())
36         {
37             NotifyAccountHolder(SavingsAccountConstants.WithdrawalMinimumBalanceWarning);
38         }
39         else
40         {
41             balance -= amountToWithdraw;
42             Console.WriteLine($"Balance after withdrawal of {amountToWithdraw} : {balance}");
43         }
44     }
45 }
```

```
42 | 0 references  
43 | public void Transfer(double amountToTransfer, SavingsAccountRefactored recipientAccount)  
44 | {  
45 |     if (IsBalanceUnderMinimum())  
46 |     {  
47 |         NotifyAccountHolder(SavingsAccountConstants.TransferMinimumBalanceWarning);  
48 |     }  
49 |     else  
50 |     {  
51 |         balance -= amountToTransfer;  
52 |         recipientAccount.balance += amountToTransfer;  
53 |         Console.WriteLine($"The sender's balance after the transfer of {amountToTransfer} is {balance}  
54 |             and the balance of the addressee account is {recipientAccount.balance}");  
55 |     }  
56 | }  
57 | 0 references  
58 | public void ProcessFee(double fee)  
59 | {  
60 |     balance -= fee;  
61 |     Console.WriteLine($"Balance after processing the {fee} fee : {balance} ");  
62 |     if (IsBalanceUnderMinimum())  
63 |     {  
64 |         NotifyAccountHolder(SavingsAccountConstants.MinimumBalanceWarning);  
65 |     }  
66 | }  
67 | }  
68 | }  
69 | }
```

5. Заключение

Основната логика на проверката е extract-ната в собствен метод. След като тя е събрана в метода `IsBalanceUnderMinimum()`, ако се наложи нещо да се коригира по отношение на калкулацията на `minimum balance`, то ще стане на едно място. С разрешаването на Shotgun Surgery освен премахването на дубликациите се постига по-добра организация на кода и по-лесна поддръжка на класа.

V. Problem : Phone Formatter

1. Кратко описание на примера

Създадени са два класа `Phone` и `Customer`. Първият има стрингово поле за неформатиран телефонен номер, който се подава на конструктора му.

Декларирани са методи за извеждане на символи от телефонния номер като код на страната, префикс и т.н. Приложен е методът Substring за field-a unformattedNumber. Вторият пък си служи с поле от тип Phone и има деклариран метод за конкатениране на телефонния номер на база извикване на методи от Phone през field-a.

2. Първоначален код

```
9  public class Phone
10 {
11     private string unformattedNumber;
12
13     1 reference
14     public Phone(string unformattedNumber)
15     {
16         this.unformattedNumber = unformattedNumber;
17     }
18
19     1 reference
20     public string GetAreaCode()
21     {
22         string areaCode = unformattedNumber.Substring(0, 4);
23         return areaCode;
24     }
25
26     1 reference
27     public string GetPrefix()
28     {
29         string prefix = unformattedNumber.Substring(4, 2);
30         return prefix;
31     }
32
33     1 reference
34     public string GetMiddleDigits()
35     {
36         string middleDigits = unformattedNumber.Substring(6, 3);
37         return middleDigits;
38     }
39
40     1 reference
41     public string GetLastDigits()
42     {
43         string lastDigits = unformattedNumber.Substring(9, 4);
44         return lastDigits;
45     }
46 }
```

```

3 references
public class Customer
{
    private Phone phone;

    1 reference
    public Customer(Phone phone)
    {
        this.phone = phone;
    }

    1 reference
    public void GetPhoneNumber()
    {
        string formattedNumber = $"{phone.GetAreaCode()} {phone.GetPrefix()} {phone.GetMiddleDigits()} {phone.GetLastDigits()}";
        Console.WriteLine(formattedNumber);
    }
}

```

3. Причина за рефакторирането

Обектите от тип `Customer` достъпват данните за обектите от тип `Phone` повече отколкото вътрешните си данни. Тези симптоми пораждат Code Smell-а от категорията Couplers Feature Envy. `Customer` обектите се пресягат към методите, идващи от `field`-а `phone`, за да форматират телефонния номер и в този ред на мисли отговорностите на класовете не са разпределени правилно.

4. Код след рефакторирането

Посредством похвата `Move Method` логиката от `GetPhoneNumber()` от оригинала се пренася от `Customer` класа в `Phone` и новият метод се кръщава `FormatPhoneNumber()`. Той връща като резултат конкатениран стринг за телефонния номер, а в `Customer` във вече преобразувания `GetPhoneNumber()` се `execute`-ва `FormatPhoneNumber()`, като стрингът се принтира на конзолата.

```

9      3 references
10     public class Phone
11     {
12         private string unformattedNumber;
13
14         0 references
15         public Phone(string unformattedNumber)
16         {
17             this.unformattedNumber = unformattedNumber;
18         }
19
20         1 reference
21         public string GetAreaCode()
22         {
23             string areaCode = unformattedNumber.Substring(0, 4);
24             return areaCode;
25         }
26
27         1 reference
28         public string GetPrefix()
29         {
30             string prefix = unformattedNumber.Substring(4, 2);
31             return prefix;
32         }
33
34         1 reference
35         public string GetMiddleDigits()
36         {
37             string middleDigits = unformattedNumber.Substring(6, 3);
38             return middleDigits;
39         }
40
41         1 reference
42         public string GetLastDigits()
43         {
44             string lastDigits = unformattedNumber.Substring(9, 4);
45             return lastDigits;
46         }
47
48         1 reference
49         public string FormatPhoneNumber()
50         {
51             string formattedNumber = $"{GetAreaCode()} {GetPrefix()} {GetMiddleDigits()} {GetLastDigits()}";
52             return formattedNumber;
53         }
54     }

```

```

1 reference
public class Customer
{
    private Phone phone;

    0 references
    public Customer(Phone phone)
    {
        this.phone = phone;
    }

    0 references
    public void GetPhoneNumber()
    {
        Console.WriteLine(phone.FormatPhoneNumber());
    }
}

```


5. Заключение

С изнасянето на кода, който извършва phone formatting в класа Phone Customer престава да взимства от feature-ите на Phone и разчита на него да изпълни необходимото. Оттук следва, че кодовата организация става по-завършена, понеже методите за обработка на телефонните данни са разположени един до друг във Phone класа.

Линк към github repository-то на курсовата работа : [plamenna-petrova/CodeQualityMetrics \(github.com\)](https://github.com/plamenna-petrova/CodeQualityMetrics)