Porządki w kodzie czyli nie tylko o refaktoringu cz. 2

Zatem zrobiliśmy pierwszy krok w kierunku uporządkowania pierwotnego kodu.

Zróbmy zatem kolejny. Przyjrzyjmy się bliżej następującemu fragmentowi:

```
public class ExtendedBitSet extends BitSet {

    int length ;
```

Pole klasy length ma widoczność pakietową. Prawdopodobnie nie to było zamierzeniem autora. Być może zapomniał w sposób jawny napisać odpowiedni modyfikator dostępu. W każdym razie warto znać jedną z podstawowych konsekwencji tego rozwiązania: pole to będzie dostępne dla wszystkich klas w pakiecie, co ogranicza jego enkapsulacje informacji. Przypadek ten można uogólnić do stwierdzenia:

> *Nadawaj najbardziej restrykcyjny z możliwych modyfikatorów dostępu*

W tym przypadku byłby to oczywiście modyfikator prywatny:

```
    private int length;
```

W przypadku gdy tworzona klasa będzie klasą bazową innych klas oraz będzie potrzeba udostępnienia tego pola, należy użyć modyfikatora dostępu protected:

```
protected int length;
```

Dobrą praktyką jest jednak wybieranie modyfikatora private, aż do momentu, gdy nie zaistnieje potrzeba użycia tego pola w klasie podrzędnej (czyli do momentu wyprowadzania nowych klas). Jest to analogia do typowej dla administratorów sieciowych strategii bezpieczeństwa: domyślnie blokuj.

Poza powyższymi uwagami, chciałbym zaproponować jeszcze jedno udoskonalenie. Jestem zwolennikiem jawnego inicjalizowania zmiennych, gdyż jest to bezpośrednie ujawnienie intencji autora. Jeśli tworzę pole obiektowe, to jawnie przypisuję mu wartość początkową na null, kiedy tworzę liczbę całkowitą inicjuję ją zerem. W ten sposób oszczędzam potencjalnemu czytelnikowi mojego kodu domyślania się, czy rzeczywiście chodziło mi o wartość domyślną, czy być może nie dokońca znając szczegóły języka, przyjąłem błędne założenie. W przypadku wyszukiwania przyczyn błędów może to mieć duże znaczenie.

Zatem podsumowując:

Jawnie inicjuj pola i zmienne

W efekcie uzyskamy takie rozwiązanie:

```
private int length = 0;
```

W powyżej przytoczonym wierszu ujawnił się jeszcze jeden nawyk związany ze sposobem programowania

Jak najwięcej przestrzeni dla Twoich oczu

Wzrok lubi przestrzeń. Nie lubi zbitych zdań, ogromnych ilości wyrażeń na niewielkiej przestrzeni. Zróbcie prosty eksperyment. Weźcie niewielki fragment swojego kodu i dodajcie kilka spacji (pomiędzy operatorami, pomiędzy wierszami) i porównajcie, który zapis jest czytelniejszy.

Oto przykład:

```
public byte[] toByteArray() {
    int bytesNumber ;
    if(length % 8 == 0) bytesNumber = length / 8 ;
    else bytesNumber = length / 8 + 1 ;
    byte[] arr = new byte[bytesNumber] ;
    for(int j = bytesNumber - 1, k = 0; j >= 0 ; j--, k++) {
        for(int i = j * 8 ; i < (j + 1) * 8; i++){
            if(i == length) break ;
            if(get(i)) arr[k] += (byte)Math.pow(2, i % 8) ;
        }
    }
```

```
        return arr ;
    }
```

oraz wersja przestrzenna:

```
    public byte[] toByteArray() {
        int bytesNumber = 0;
        if ( length % 8 == 0 ) {
            bytesNumber = length / 8;
        } else {
            bytesNumber = length / 8 + 1;
        }
        byte [] arr = new byte[ bytesNumber ];
        for( int j = bytesNumber - 1, k = 0; j >= 0 ; j--, k++ ) {
            for( int i = j * 8 ; i < ( j + 1 ) * 8; i++ ) {
                if ( i == length ) {
                    break;
                }
                if ( get( i ) ) {
                    arr[ k ] += (byte) Math.pow( 2, i % 8 );
                }
            }
        }
        return arr ;
    }
```

Modyfikacja ta zajęła mi około dwóch minut. Jednak umiejętność tworzenia kodu zgodnego ze stylem kodowania, która jest moim

nawykiem, umożliwia mi pisanie takiego kodu bez żadnego dodatkowego nakładu czasu. Za to jaka przyjemność z czytania! A to dopiero początek. Kiedy mówimy o stylu kodowania przychodzi mi do głowy jeszcze jedna reguła:

> *Używaj konsekwentnie przyjętego standardu kodowania*

Obecnie nie wyobrażam sobie tworzenia kodu, który nie podlega z góry ustalonym zasadom. Jeśli chodzi o pracę w zespole, jest to wręcz warunek konieczny pracy grupowej. A mamy ogromne wsparcie, gdyż istnieje wiele gotowych do wykorzystania standardów, np. *Code Conventions for the Java Programming Language* (http://java.sun.com/docs/codeconv/), oraz narzędzi, które pomogą go, szczególnie w początkowym okresie, sumiennie przestrzegać (*Checkstyle* http://checkstyle.sourceforge.net/).

Poniżej znajduje się przykład omawianej klasy sformatowany wg standardu opartego na standardzie zaproponowanym przez Suna.

```
public class ExtendedBitSet extends BitSet {

    private int length = 0;

    public ExtendedBitSet( int size, String str ) {

        super( size );

        length = size;

        int strLength = str.length();
```

```
        for( int i = 0; i < strLength; ++i ) {
            if ( str.charAt( strLength - 1 - i ) == '1' ) {
                set( i );
            }
        }
    }
    public ExtendedBitSet( String str ) {
        super( str.length() );
        int strLength = str.length();
        length = strLength;
        for( int i = 0; i < strLength; ++i ) {
            if( str.charAt( strLength - 1 - i ) == '1' ) {
                set( i );
            }
        }
    }
    public void merge( ExtendedBitSet extendedBitSet ) {
        for ( int i = extendedBitSet.nextSetBit( 0 ); i >= 0;
                        i = extendedBitSet.nextSetBit( i + 1 ) ) {
            this.set( this.length + i );
        }
        this.length = this.length + extendedBitSet.length;
    }
    public int boolMultiply( ExtendedBitSet extendedBitSet ) {
        int sum = 0;
        int len = 0;
        if( this.length < extendedBitSet.length ) {
            len = this.length;
```

```
        } else {
            len = extendedBitSet.length;
        }
        for( int i = 0; i < len; i++ ) {
            if ( this.get(i) && extendedBitSet.get(i) ) {
                sum++;
            }
        }
        return sum % 2 ;
    }

    public byte[] toByteArray() {
        int bytesNumber = 0;
        if ( length % 8 == 0 ) {
            bytesNumber = length / 8;
        } else {
            bytesNumber = length / 8 + 1;
        }

        byte [] arr = new byte[ bytesNumber ];
        for( int j = bytesNumber - 1, k = 0; j >= 0 ; j--, k++ ) {
            for( int i = j * 8 ; i < ( j + 1 ) * 8; i++ ) {
                if ( i == length ) {
                    break;
                }
                if ( get( i ) ) {
                    arr[ k ] += (byte) Math.pow( 2, i % 8 );
                }
```

```
            }
        }
        return arr ;
    }

    public String convertToBitString( int size ) {
        char [] resultArray = new char[ size ];
        for ( int i = 0; i < size; ++i ) {
            resultArray[ i ] = '0';
        }
        for ( int i = this.nextSetBit( 0 ); i >= 0; i = this.nextSetBit( i + 1 ) ) {
            resultArray[ size - 1 - i ] = '1';
        }
        return new String( resultArray );
    }

    public String convertToBitString() {
        return convertToBitString( this.length );
    }
}
```

Wróćmy do naszego pola length. Tak naprawdę posiada ono jeszcze jeden mankament – jego nazwa jest dokładnie taka sama jak nazwa metody z klasy bazowej. Jest to sytuacja niekorzystna z dwóch powodów:

- dwa różne byty nie powinny mieć tej samej nazwy (metoda i pole), gdyż może to prowadzić do pomyłek,
- nazwa zmiennej nie odzwierciedla sensu właściwości. Pole to

przechowuje wartość, która określa ustaloną długość wektora bitowego. Dużo większy sens miałaby na przykład nazwa fixedLength.

Zmieńmy zatem nazwe pola length na fixedLength.

Podsumowując powyższe rozważania:

Nie używaj jednej nazwy do różnych celów

oraz

Nadawaj polom, metodom i klasom nazwy, które jednoznacznie odzwierciedlają ich znaczenie

Analizując dalej przykład, spójrzmy na oba konstruktory, wyraźnie zauważymy pewną właściwość - jest tam mnóstwo powtarzającego się kodu. W ten sposób docieramy do zasady będącej esencją refaktoringu:

Eliminuj wszelkie powtórzenia

Powtórzenie to zło, które towarzyszy programistom na każdym kroku. Kuszące kopiuj-wklej, zazwyczaj ostatecznie prowadzi do

kilkunastominutowych lub co gorsza wielogodzinnych poszukiwań błędów, wynikających z rozsynchronizowania się podobnych fragmentów kodu. Powtórzenia na dłuższą metę są nie do utrzymania, stąd ich eliminowanie jest podstawowym celem wszelkich refaktoringów. Przykładowy kod możemy zmienić do następującej postaci:

```
public ExtendedBitSet( int size, String str ) {
    super( size );
    fixedLength = size;
    initializeBitSet( str );
}
public ExtendedBitSet( String str ) {
    this( str.length(), str );
    initializeBitSet( str );
}

private void initializeBitSet( String str ) {
    int strLength = str.length();
    for( int i = 0; i < strLength; ++i ) {
        if ( str.charAt( strLength - 1 - i ) == '1' ) {
            set( i );
        }
    }
}
```

Kod nam się powoli porządkuje i wygląda coraz lepiej. Wprowadziliśmy zmiany związane z wyglądem (standard kodowania i przestrzeń), wyeliminowaliśmy kilka powtórzeń i

niejednoznaczności. Oczywiście zawsze należy wyważyć stopień refaktorowania lub upiększania kodu, tak aby nie stać się ofiarą perfekcjonizmu. Warto wesprzeć się pomocą innych programistów, najlepiej takich, którzy sami posługują się pewnymi zasadami oraz posiadają duże doświadczenie, i poprosić o opinię. Z pewnością wiele można się będzie dowiedzieć na temat swojego programowania.

O mnie

Mariusz Sieraczkiewicz

Trener, konsultant, menedżer projektów IT, coach. Założyciel zespołu programistów Equilibrium. Współinicjator JUGa Łódź. Autor artykułów o inżynierii oprogramowania. Współwłaściciel firmy szkoleniowej BNS IT. Szczęśliwy mąż :-)
http://www.linkedin.com/pub/2/a24/812

O BNS IT

BNS IT jest firmą szkoleniowo-doradczą zajmującą się **rozwijaniem i doskonaleniem zespołów programistycznych**.

Nasze usługi skierowane są do firm zatrudniających programistów. Pracujemy dla trzech grup klientów:

- firm zajmujących się wytwarzaniem oprogramowania
- firm, które posiadają zespoły programistyczne, lecz działają na innych rynkach niż IT
- firm integrujących systemy informatyczne

http://www.bnsit.pl/