Nie tylko refaktoring

Mariusz Sieraczkiewicz
http://msieraczkiewicz.blogspot.com
http://www.bnsit.pl
http://www.lodz.jug.pl

Możesz swobodnie dystrybuować ten plik PDF, jeśli pozostawisz go w nietkniętym stanie. Możesz także cytować jego fragmenty, umieszczając w tekście odnośnik
http://msieraczkiewicz.blogspot.com

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

Porządki w kodzie czyli nie tylko o refaktoringu cz. 1

Początkowo moim zamysłem było stworzenie artykułu o refaktoringu. Jednak im bardziej zastanawiałem się nad tematem, tym bardziej utwierdzałem się w przekonaniu, iż nie będę pisał tylko i wyłącznie o refaktoringu. Chodzi o coś znacznie istotniejszego, o przelanie bardzo rozległej wiedzy, a w zasadzie doświadczenia związanego z tworzeniem kodu. Kodu, który nie tylko działa, nie tylko jest dobrze zaprojektowany, ale przede wszystkim doskonale się czyta. Kiedy osiągamy tę umiejętność, stajemu u progu profesjonalizmu. Programistycznego profesjonalizmu.

Zatem będzie to artykuł między innymi o refaktoringu, ale wzbogacony o zbiór przemyśleń, sugestii, czasami również wątpliwości, którą mają pobudzić Cię, Czytelniku, do refleksji, zweryfikowania swoich programistycznych poczynań. Wierzę, że spowodują cały proces zmian – wprowadzenia nowych, dobrych nawyków.

Przede wszystkim czytelność

Programowanie bardzo szybko ewoluuje. Pamiętam jeszcze dość dobrze czasy, kiedy rozpocząłem swoją przygodę z kodowaniem jakieś dziesięć lat temu. Programy pisało się wtedy całkiem inaczej. Ceniono pomysłowość, zwięzłość i enigmatyczność. Im kod był bardziej niezrozumiały, tym programista był lepszy.

Jednak z czasem systemy informatyczne stawały się coraz bardziej skomplikowane, wymagały coraz większej wiedzy i co najważniejsze, stały się produktem pracy zespołowej. Obecnie pojedynczy programista nie jest w stanie zdziałać zbyt wiele. Być

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.

może stworzy rozbudowany program desktopowy, natomiast nie będzie w stanie w wystarczająco skończonym czasie stworzyć rozproszonego systemu, opartego o architekturę trójwarstwową, zapewniającego odpowiedni poziom bezpieczeństwa, umożliwiającego zarządzanie prawami dostępu do wybranych części aplikacji, realizującym wielojęzyczność itp. itd. Takie systemy tworzy obecnie kilkunastu lub kilkudziesięciu programistów, w zależności od wielkości projektu, przez kilka lub kilkanaście miesięcy. Programista przestał być nierozumianym przez nikogo indywidualistą, a stał się graczem zespołowym, nastawionym na współpracę.

Co za tym idzie, sposób kodowania też musiał się zmienić. Wyłonił się podstawowy postulat dotyczący kodowania:

Przede wszystkim czytelność

Istnieją przynajmniej trzy podstawowe powody, które potwierdzają ważność tego stwierdzenia:

- wymagania się zmieniają,
- programowanie to umiejętność zespołowa,
- projekty są zbyt duże, aby pojedyncza osoba była w stanie ogarnąć całość.

Z tych właśnie powodu w ciągu ostatnich kilku lat bardzo mocno rozwijają się takie techniki jak refaktoring, pisanie testów oraz zwraca się ogromną uwagę na standard kodowania.

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

To właśnie Czytelność będzie głównym bohaterem tego artykułu. Będzie on zawierać sugestie i przemyślenia, które ułatwią realizację powyższego postulatu. Niektóre wskazówki będą stanowić moją subiektywną opinię, inne będą wyrażać mądrość doświadczeń społeczności programistycznej. Oczywiście należy pamiętać o pewnej zasadzie: "Jedyną niezmienną zasadą jest to, że nie ma niezmienych zasad". Uogólniając, należy stwierdzić, iż przedstawiane wnioski sprawdziły się w wielu sytuacjach, co nie znaczy, że są zasadne w 100% przypadkach. Dlatego należy uważnie się przyglądać pojawiającym się na co dzień problemom i odważnie stosować przytoczone wskazówki. Warto krytycznie spojrzeć na swoje nawyki lub ich brak i rozpocząć zmiany. Zatem do dzieła!

Poprzez przykład do celu

Analiza kodu mniej doświadczonych programistów, często doprowadzała mnie do zaskakujących spostrzeżeń, umożliwiających znalezienie źródła problemów młodych (ale również i tych doświadczonych) adeptów sztuki programowania. Dlatego artykuł ten oparty będzie o przykład nie najlepiej napisanej klasy, która będzie analizowana i stopniowo udoskonalana.

Celem, postawionym przed autorami poniższego kodu, było zaimplementowanie klasy pochodnej klasy java.util.BitSet (wektora bitowego) wzbogaconej o:

- możliwość konkatenacji,
- właściwość narzuconej długości wektora (pole length),
- specyficznego mnożenia dwóch wektorów bitowych polegającego na zwróceniu wartości 0, jeśli jedynki w obu wektorach bitowych powtarzają się na parzystej ilości miejsc,

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.

oraz wartości 1, jeśli jedynki pokrywają się na nieparzystej ilości miejsc,

- operację zamiany wektora na ciąg znakowy (w określonym z góry formacie),
- operację zamiany wektora w ciąg bajtów.

Pragnę zaznaczyć, iż treść przykładu nie ma tu większego znaczenia. Przytoczony kod służy tylko jako ilustracja często występujących niedoskonałości programistycznych. Ponadto, ponieważ nieodłączną częścią refaktoringu są testy, sprawdzające testowany kod, jako dodatek do artykułu została zamieszczona klasa testowa do analizowanej klasy.

Oto zaproponowana implementacja nowej wersji wektora bitowego:

```
import java.util.* ;

public class ExtendedBitSet extends BitSet {

    int length ;

    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) {

        this( str.length(), str );
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        int strLength = str.length();
        for(int i = 0; i < strLength; ++i) {
            if(str.charAt( strLength - 1 - i) == '1') set(i) ;
        }
    }
    public static ExtendedBitSet merge(ExtendedBitSet a, ExtendedBitSet b) {
        StringBuffer str = new StringBuffer(a.convertToBitString() + b.convertToBitString()) ;
        return new ExtendedBitSet(a.length + b.length, str.toString()) ;
    }
    public static int boolMultiply(ExtendedBitSet a, ExtendedBitSet b) {
        int sum = 0 ;
        int len ;
        if(a.length < b.length) len = a.length ;
        else len = b.length ;
        for(int i = 0; i < len; i++) {
            if (a.get(i) && b.get(i)) sum++ ;
        }
        return sum % 2 ;
    }
    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 ;
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
                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 );
    }
}
```

W pierwszej kolejności spójrzmy na klasę całościowo. Jedna z pierwszych rzeczy, która rzuca się w oczy to fakt, że metody konkatenacji i mnożenia wektorów są statyczne. Jest to sprzeczne z bardzo ważną zasadą:

> *Twórz spójne interfejsy i klasy*

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

Jeśli przyjrzymy się klasie bazowej BitSet, łatwo zauważymy, iż żadna publiczna metoda nie jest statyczna. Dostępne są m. in. niestatyczne metody or(Bitset), xor(Bitset), których celem jest modyfikacja obiektu na rzecz którego są one wywolywane (operacja na this), a nie udostępnienie metody zewnętrznej (statycznej), która tworzy nowy obiekt, będący efektem implementowanej operacji. Zatem obydwie metody (merge i boolMultiply) swoją postacią wprowadzają rozdźwięk w strukturze nowej klasy, prowadząc do niespójnego interfejsu klasy ExtendedBitSet. W tym przypadku utrzymanie spójności poprzez zamianę metod statycznych na metody niestatyczne, uprości używanie klasy ExtendedBitSet, gdyż będzie się z niej korzystać tak samo jak z klasy BitSet.

Istnieje jeszcze jedna zasada, którą warto przytoczyć analizując metody merge i boolMultiply:

Unikaj statycznych elementów w programowaniu

Elementy statyczne to pozostałość po programowaniu proceduralnym, gdyż statyczność oznacza globalność. A przecież jedną z konsekwencji programowania obiektowego jest zamykanie implementowanych funkcjonalności w autonomicznych i możliwie jak najbardziej niezależnych obiektach. Dlatego elementów statyczne używaj tylko wtedy, kiedy nie ma innego wyjścia lub kiedy informacja lub operacja ma rzeczywiście charakter globalny. Zatem używaj pól statycznych jako stałych, szczególnie stałych globalnych, zaś metod statycznych używaj dla operacji globalnych. Przykładem użycia metod i pól statycznych jest wzorzec Singletonu, jednak „wzorcowość” tego wzorca bywa kwestionowana

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

(http://c2.com/cgi/wiki?SingletonsAreEvil). Ponadto należy pamiętać, że metody statyczne nie są polimorficzne, co oznacza, że nie możemy dostarczyć ich alternatywnych implementacji oraz że nie możemy ich zastępować za pomocą mocków. Zatem ich użycie powoduje usztywnienie kodu oraz utrudnia testowanie.

Zmieńmy zatem nieco przytoczony kod, zgodnie z pierwszymi dwoma regułami:

```
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 ;
    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 ;
}
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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;
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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:

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
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) ;

        }

    }
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        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

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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();
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        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;
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        } 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 );
                }
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
            }
        }
        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

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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.

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

Porządki w kodzie czyli nie tylko o refaktoringu cz. 3

Przyjrzyjmy się teraz metodzie boolMultipy oraz zmiennej lokalnej o nazwie sum. Zmienna ta jest deklarowana na samym początku metody, uzywana jest dużo później. Jest to nawyk, który pozostał jeszcze po językach proceduralnych (takich jak wczesne C przy PL/SQL), gdzie wymaga się zadeklarowania wszystkich zmiennych używanych w metodzie na samym początku. Na szczęście większość współczesnych języków programowania (w szczególności języków obiektów) nie ma tego ograniczenia. Zamiast tego podejścia sugeruję realizację zasady leniwej deklaracji zmiennych, którą można wyrazić za pomocą zdania

Deklaruj zmienne najpóźniej jak to tylko możliwe

Warto to robić z bardzo prostego powodu – łatwiej będzie czytać kod, jeśli w zasięgu wzroku będziemy mieli operacje wykonywane na danej zmiennej. Przy bardziej złożonych metodach umieszczenie deklaracji na samym początku, może spowodować, że analizując dalszą część metody nie będziemy w stanie zorientować się czy zmienna była do tej pory przetwarzana czy nie i co wpłynęło na jej wartośc.

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

Załóżmy, że przewinęliśmy ekran tak, że widzimy następujący kod:

```
        for( int i = 0; i < len; i++ ) {

            if ( this.get(i) && extendedBitSet.get(i) ) {

                sum++;

            }

        }

        return sum % 2 ;
```

Rodzi się we mnie natychmiast pytanie – a co to za suma? Czy działo się z nią coś wcześniej? Czy może jej wartość została pobrana z zewnątrz?

W przypadku jednej zmiennej jeszcze to nie jest duży problem, ale wyobraźmy sobie sytuację, kiedy takich zmiennych jest pięć. Śledzenie logiki takiej metody będzie bardzo trudne.

Bardzo prosta operacja przeniesienia deklaracji nieco niżej spowoduje, że kod będzie dużo bardziej czytelny:

```
    public int boolMultiply( ExtendedBitSet extendedBitSet ) {

        int len = 0;

        if( this.fixedLength < extendedBitSet.fixedLength ) {

            len = this.fixedLength;

        } else {

            len = extendedBitSet.fixedLength;

        }
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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

Nieco bardziej złożona sytuacja jest w metodzie toByteArray. Jest ona niesamowicie trudna w analizie. Mi zajęło około 20 minut zrozumienie zasady jej działania. Teraz sobie wyobraźmy projekt złożony z tysiąca klas, z których każda zawiera tego typu metody. Zapanowanie nad takim kodem będzie graniczyło z cudem.
Spójrzmy na kod tej metody:

```
    public byte[] toByteArray() {
        int bytesNumber = 0;

        if ( fixedLength % 8 == 0 ) {
            bytesNumber = fixedLength / 8;
        } else {
            bytesNumber = fixedLength / 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++ ) {
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
            if ( i == fixedLength ) {
                break;
            }
            if ( get( i ) ) {
                arr[ k ] += (byte) Math.pow( 2, i % 8 );
            }
        }
    }
    return arr;
}
```

Metoda toByteArray zwraca bajtową reprezentacje ciągów bitowych – czyli ciąg bitowy o długości 14 bitów można zaprezentować za pomocą 2 bajtów, zaś ciąg bitowy o długości 23 bity można zaprezentować za pomocą 3 bajtów.

Algrotym można opisać w następujących krokach:

- określ liczbę bajtów,
- dla każdej ósemki bitów (lub kilku bitów w przypadku niepełnych bajtów) wykonaj następującą operację:
  - sprawdź wartość każdego bitu
  - jeśli bit ma wartość 1, dodaj odpowiednią potęgę dwójki (wynikającą z pozycji bitu w bajcie) do wyniku danego bajtu.

Dlaczegóżby nie wyrazić tego algorytmu w formie programistycznej? Często o tym się zapomina. Jako programiści

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

próbujemy w zwięzły sposób wyrazić nasze pomysły, co zazwyczaj prowadzi do bardzo nieczytelnych rozwiązań, których nie tylko inni ale i my sami nie jesteśmy w stanie zrozumieć w krótkim czasie. Ideałem jest dążenie do tego, aby jedno spojrzenie wystarczyło do odszyfrowania intencji twórcy. Nie potrzebujemy zagłębiać się w szczegóły realizacji algortmu, ważne byśmy wychwycili jego główną myśl. Spójrzmy na inną implementację metody toByteArray:

```
public byte[] toByteArray() {
    int bytesCount = computeBytesCount();
    byte [] byteArray = new byte[ bytesCount ];
    for ( int i = 0; i < bytesCount; ++i ) {
        int byteNumber = bytesCount - i - 1;
        byteArray[ byteNumber ] = computeByteValue( i );
    }
    return byteArray;
}
```

Najważniejsza zmiana, polega na bezpośrednim wyrażeniu algorytmu na ogólnym poziomie. Dzięki czemu jesteśmy w stanie w krótkim czasie zrozumieć intencję twórcy. Pomocnicza metoda computeBytesCount znajduje ilość bajtów nowej reprezentacji. W pętli dla każdego bajtu wykonywana jest operacja obliczania wartości bajtu i zapamiętywania wyniku. Czyż taki zapis nie jest dużo prostszy? Co z tego, że nie widać wszystkich elementów realizacji algorytmu. Bardziej dociekliwi zawsze mogą zajrzeć do metod computeBytesCount i computeByteValue.

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

Mogą one wyglądać następująco:

```
private byte computeByteValue( int byteNumber ) {
    int firstBitPosition = byteNumber * 8;
    int lastBitPosition = ( byteNumber + 1 ) * 8 - 1;

    byte byteValue = 0;
    for ( int i = this.nextSetBit( firstBitPosition );
        i >= firstBitPosition &&  i <= lastBitPosition;
                i = this.nextSetBit( i + 1 ) ) {
        int currentBitPosition = i - firstBitPosition;
        if ( get( i ) == true ) {
            byteValue += (byte) Math.pow( 2, currentBitPosition % 8 );
        }
    }
    return byteValue;
}

private int computeBytesCount() {
    int bytesCount = 0;
    if ( fixedLength % 8 == 0 ) {
        bytesCount = fixedLength / 8;
    } else {
        bytesCount = fixedLength / 8 + 1;
    }
    return bytesCount;
}
```

Warto zauważyć, że kod realizujący zadanie zbytnio się nie uprościł,

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

ale dużo łatwiej jest go teraz przeanalizować i zrozumieć. Teoria refaktoringu nazywa tego typu działania wyłuskiwaniem metody (ang. extract method). Prawdę mówiąc tutaj poszliśmy nieco dalej, gdyż zmodyfikowaliśmy nieco implementację algorytmu, tak aby stała się bardziej czytelna. Zwróćmy uwagę na wiersze:

```
int firstBitPosition = byteNumber * 8;

int lastBitPosition = ( byteNumber + 1 ) * 8 - 1;
```

Tak naprawdę są one wyodrębnieniem bardzo enigmatycznych wyrażeń z wiersza:

```
for( int i = j * 8 ; i < ( j + 1 ) * 8; i++ ) {
```

Czyż intencja w takim przypadku nie staje się oczywista? Moje wieloletnie doświadczenia doprowadziły mnie do wniosku:

Jawnie nazywaj złożone elementy kodu

zamiast

```
j * 8
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

napisz

```
int firstBitPosition = byteNumber * 8;
```

Zasadę te rozszerzam do instrukcji warunkowych. Często w kodzie możemy spotkać wyrażenia, za którymi kryje się pewna logika. Warto bezpośrednio wyrazić naszą intencję. Czytelność niesamowicie wzrasta.

Zamiast

```
if ( index >= 0 )
```

napisz

```
boolean isIndexInRange = ( index >= 0 );

if ( isIndexInRange )
```

Kod zaczyna się czytać jak książkę! Przecież programowanie jest dla ludzi. Ułatwiajmy zatem sobie życie.

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

A oto najważniejsza zasada, będąca kwintesencją powyższych rozważań:

Pisz kod w taki sposób, aby czytało się go jak powieść. Używaj jednoznacznych i jednocześnie prostych nazw. Realizowane operacje dziel na logiczne części i każdą implementuj w osobnych metodach.

Myślę, że jak na jeden raz, wystarczy. Wystarczy, żeby zaostrzyć apetyty i wzbudzić ochotę na więcej. Żeby oprowadzić nieco po ogrodzie refaktoringu i jego przyległościach. Zapewniam, że to niesamowite miejsce i daje niesamowicie wiele radości i satysfakcji.

Poniżej zamieszczam ostateczną wersję kodu, który przechodzi załączone testy (oczywiście ze zmianą uwzględniającą niestatyczność metod merge i boolMultiply).

Końcowa postać przykładowej klasy (warto ją porównać z postacią początkową):

```
public class ExtendedBitSet extends BitSet {

    private int fixedLength = 0;

    public ExtendedBitSet( int size, String str ) {

        super( size );

        fixedLength = size;

        initializeBitSet( str );

    }
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
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 );

        }

    }

}


public void merge( ExtendedBitSet extendedBitSet ) {

    for ( int i = extendedBitSet.nextSetBit( 0 ); i >= 0;

                    i = extendedBitSet.nextSetBit( i + 1 ) ) {

        this.set( this.fixedLength + i );

    }

    this.fixedLength = this.fixedLength + extendedBitSet.fixedLength;

}


public int boolMultiply( ExtendedBitSet extendedBitSet ) {

    int len = 0;

    if( this.fixedLength < extendedBitSet.fixedLength ) {

        len = this.fixedLength;

    } else {

        len = extendedBitSet.fixedLength;
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        }

        int sum = 0;

        for( int i = 0; i < len; i++ ) {

            if ( this.get( i ) && extendedBitSet.get( i ) ) {

                sum++;

            }

        }

        return sum % 2 ;

    }


    public byte[] toByteArray() {

        int bytesCount = computeBytesCount();

        byte [] byteArray = new byte[ bytesCount ];


        for ( int i = 0; i < bytesCount; ++i ) {

            int byteNumber = bytesCount - i - 1;

            byteArray[ byteNumber ] = computeByteValue( i );

        }

        return byteArray;

    }


    private byte computeByteValue( int byteNumber ) {

        int firstBitPosition = byteNumber * 8;

        int lastBitPosition = ( byteNumber + 1 ) * 8 - 1;


        byte byteValue = 0;

        for ( int i = this.nextSetBit( firstBitPosition );

            i >= firstBitPosition &&  i <= lastBitPosition;
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com   http: //www.bnsit.pl   http://www.lodz.jug.pl

```
                i = this.nextSetBit( i + 1 ) ) {
            int currentBitPosition = i - firstBitPosition;
            if ( get( i ) == true ) {
                byteValue += (byte) Math.pow( 2, currentBitPosition % 8 );
            }
        }
        return byteValue;
    }



    private int computeBytesCount() {
        int bytesCount = 0;
        if ( fixedLength % 8 == 0 ) {
            bytesCount = fixedLength / 8;
        } else {
            bytesCount = fixedLength / 8 + 1;
        }
        return bytesCount;
    }

    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';
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        }

        return new String( resultArray );

    }


    public String convertToBitString() {

        return convertToBitString( this.fixedLength );

    }

}
```

Dodatek. Test przykładowej klasy.

```
public class ExtendedBitSetTest extends TestCase {

    public void testConstructorSizeAndString() throws Exception {

        assertEquals( "{0, 2}", new ExtendedBitSet( 10, "101" ).toString() );

        assertEquals( "0000000101", new ExtendedBitSet( 10, "101" ).convertToBitString() );

        assertEquals( "0000001011", new ExtendedBitSet( 10, "1011" ).convertToBitString() );

        assertEquals( "0010001011", new ExtendedBitSet( 10, "10001011" ).convertToBitString() );

        assertEquals( "{0, 1, 3, 7}", new ExtendedBitSet( 10, "10001011" ).toString() );

        assertEquals( "{0}", new ExtendedBitSet( 10, "001" ).toString() );

        assertEquals( "0000000001", new ExtendedBitSet( 10, "001" ).convertToBitString() );

        assertEquals( "00000000001", new ExtendedBitSet( 10, "001" ).convertToBitString( 11 ) );

    }

    public void testMerge() throws Exception {

        ExtendedBitSet extendedBitSet1 = new ExtendedBitSet( 10, "1" );

        ExtendedBitSet extendedBitSet2 = new ExtendedBitSet( 10, "1" );

        assertEquals( "0000000001", extendedBitSet1.convertToBitString() );

        assertEquals( "0000000001", extendedBitSet2.convertToBitString() );
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        ExtendedBitSet mergedBitSet = ExtendedBitSet.merge( extendedBitSet1, extendedBitSet2 );

        String mergedString = mergedBitSet.convertToBitString();


        assertEquals( "00000000010000000001", mergedString );

        assertEquals( "{0, 10}", mergedBitSet.toString() );

        assertTrue( mergedBitSet.get( 0 ) == true );

    }


    public void testBoolMultiple() throws Exception {

        ExtendedBitSet extendedBitSet1 = new ExtendedBitSet( 3, "1" );

        ExtendedBitSet extendedBitSet2 = new ExtendedBitSet( 10, "1" );

        assertEquals( 1, ExtendedBitSet.boolMultiply( extendedBitSet1, extendedBitSet2 ) );


        extendedBitSet1 = new ExtendedBitSet( 1000, "1" );

        extendedBitSet2 = new ExtendedBitSet( 2, "1" );


        assertEquals( 1, ExtendedBitSet.boolMultiply( extendedBitSet1, extendedBitSet2 ) );


        extendedBitSet1 = new ExtendedBitSet( 10, "1" );

        extendedBitSet2 = new ExtendedBitSet( 10, "1" );


        assertEquals( 1, ExtendedBitSet.boolMultiply( extendedBitSet1, extendedBitSet2 ) );


        extendedBitSet1 = new ExtendedBitSet( 10, "1" );

        extendedBitSet2 = new ExtendedBitSet( 10, "10" );


        assertEquals( 0, ExtendedBitSet.boolMultiply( extendedBitSet1, extendedBitSet2 ) );
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

```
        extendedBitSet1 = new ExtendedBitSet( 10, "10" );

        extendedBitSet2 = new ExtendedBitSet( 10, "10" );


        assertEquals( 1, ExtendedBitSet.boolMultiply( extendedBitSet1, extendedBitSet2 ) );


        extendedBitSet1 = new ExtendedBitSet( 10, "110" );

        extendedBitSet2 = new ExtendedBitSet( 10, "110" );


        assertEquals( 0, ExtendedBitSet.boolMultiply( extendedBitSet1, extendedBitSet2 ) );

    }


    public void testToByteArray() throws Exception {


        ExtendedBitSet extendedBitSet = new ExtendedBitSet( "100000110" );

        byte[] toByteArray = extendedBitSet.toByteArray();


        assertEquals( 1, toByteArray[ 0 ] );

        assertEquals( 6, toByteArray[ 1 ] );


        extendedBitSet = new ExtendedBitSet( "10111111111" );


        toByteArray = extendedBitSet.toByteArray();


        assertEquals( 5, toByteArray[ 0 ] );

        assertEquals( -1, toByteArray[ 1 ] );

    }

}
```

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.
http://msieraczkiewicz.blogspot.com http: //www.bnsit.pl http://www.lodz.jug.pl

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/

Copyright © 2008 Mariusz Sieraczkiewicz. Pewne prawa zastrzeżone.