# XCODE PLUGIN

*Dokumentacja wstępna*

*Stepnowski Marcin*

## 1. Cel projektu

Celem projektu jest stworzenie pluginu do programu XCode, będący IDE do języka Objective-C, który będzie refaktoryzował kod spełniając niżej wymienione wymagania funkcjonalne.

## 2. Wymagania funkcjonalne

### WF1. Dodawanie atrybutu @property na podstawie zaznaczonych zmiennych.

Kod źródłowy przed refaktoryzacją:

| Foo.h | Foo.m |
|---|---|
| `#import <Foundation/Foundation.h>`<br><br>`@interface Foo : NSObject{`<br>`  int _foold;`<br>`  NSString* _name;`<br>`  NSDate* _date;`<br>`  Foo* _another;`<br>`}`<br><br>`@end` | `#import "Foo.h"`<br><br>`@implementation Foo`<br><br>`@end` |

Po zaznaczeniu tekstu (zmiennych klasy) i wybraniu opcji faktoryzacji property:

| Foo.h | Foo.m |
|---|---|
| `#import <Foundation/Foundation.h>`<br><br>`@interface Foo : NSObject{`<br>`  int _foold;`<br>`  NSString* _name;`<br>`  NSDate* _date;`<br>`  Foo* _another;`<br>`}`<br><br>`@end` | `#import "Foo.h"`<br><br>`@implementation Foo`<br><br>`@end` |

Otrzymamy taki rezultat:

| Foo.h | Foo.m |
|---|---|
| `#import <Foundation/Foundation.h>`<br><br>`@interface Foo : NSObject{`<br>`  int _foold;`<br>`  NSString* _name;` | `#import "Foo.h"`<br><br>`@implementation Foo`<br>`@synthesize foold = _foold,`<br>`name = _name,` |

```
  NSDate* _date;
  Foo* _another;
}

@property (nonatomic) int fooId;
@property (nonatomic, copy) NSString* name;
@property (nonatomic, copy) NSDate* date;
@property (nonatomic, strong) Foo* another;

@end
```

```
date = _date,
another = _another;

@end
```

## WF2. Tworzenie definicji na podstawie nagłówków funkcji

Kod źródłowy przed refaktoryzacją:

Foo.m

```
#import <Foundation/Foundation.h>

@interface Foo : NSObject{
  int _fooId;
  NSString* _name;
  NSDate* _date;
  Foo* _another;
}

@property (nonatomic) int fooId;
@property (nonatomic, copy) NSString* name;
@property (nonatomic, copy) NSDate* date;
@property (nonatomic, strong) Foo* another;

-(id)initWithDictionary:(NSDictionary*)dictionary;
-(int)exampleIntReturnMethod;
-(bool)isEqual:(Foo*)other;

@end
```

Foo.m

```
#import "Foo.h"

@implementation Foo
@synthesize fooId = _fooId,
name = _name,
date = _date,
another = _another;

-(id)initWithDictionary:(NSDictionary *)dictionary{
  self = [super init];
  if(self){
#warning not implemented initWithDictionary:
  }
  return self;
}

-(bool)isEqual:(Foo *)other{
#warning not implemented isEqual:
  return false;
}

@end
```

Zaznaczamy tekst i wybieramy opcję dodawania nagłówków:

Foo.m

```
#import <Foundation/Foundation.h>

@interface Foo : NSObject{
  int _fooId;
  NSString* _name;
  NSDate* _date;
  Foo* _another;
}

@property (nonatomic) int fooId;
@property (nonatomic, copy) NSString* name;
@property (nonatomic, copy) NSDate* date;
@property (nonatomic, strong) Foo* another;

-(id)initWithDictionary:(NSDictionary*)dictionary;
-(int)exampleIntReturnMethod;
-(bool)isEqual:(Foo*)other;
```

Foo.m

```
#import "Foo.h"

@implementation Foo
@synthesize fooId = _fooId,
name = _name,
date = _date,
another = _another;

-(id)initWithDictionary:(NSDictionary *)dictionary{
  self = [super init];
  if(self){
#warning not implemented initWithDictionary:
  }
  return self;
}

-(bool)isEqual:(Foo *)other{
#warning not implemented isEqual:
```

| | |
|---|---|
| @end | return false;<br>}<br><br>@end |

Rezultat:

| Foo.h | Foo.m |
|---|---|
| #import <Foundation/Foundation.h><br><br>@interface Foo : NSObject{<br>int _fooId;<br>NSString* _name;<br>NSDate* _date;<br>Foo* _another;<br>}<br><br>@property (nonatomic) int fooId;<br>@property (nonatomic, copy) NSString* name;<br>@property (nonatomic, copy) NSDate* date;<br>@property (nonatomic, strong) Foo* another;<br><br>-(id)initWithDictionary:(NSDictionary*)dictionary;<br>-(int)exampleIntReturnMethod;<br>-(bool)isEqual:(Foo*)other;<br><br>@end | #import "Foo.h"<br><br>@implementation Foo<br>@synthesize fooId = _fooId,<br>name = _name,<br>date = _date,<br>another = _another;<br><br>-(id)initWithDictionary:(NSDictionary *)dictionary{<br>self = [super init];<br>if(self){<br>#warning not implemented initWithDictionary:<br>}<br>return self;<br>}<br><br>-(int)exampleIntReturnMethod{<br>#warning not implemented exampleIntReturnMethod<br>return 0;<br>}<br><br>-(bool)isEqual:(Foo *)other{<br>#warning not implemented isEqual:<br>return false;<br>}<br><br>@end |

Jak widzimy refaktoryzacja, nie polega tylko na generowaniu kodu, ale umieszczeniu go w pliku .m w odpowiednim miejscu, o ile jest to możliwe, tak aby definicje funkcji, były w takiej samej kolejności jak nagłówki funkcji. Generacja tych definicja, jest możliwa poprzez zaznaczenie wszystkich nagłówków funkcji i wybraniu opcji dodawania definicji w pliku .m.

## 3. Podział na moduły i komunikacja

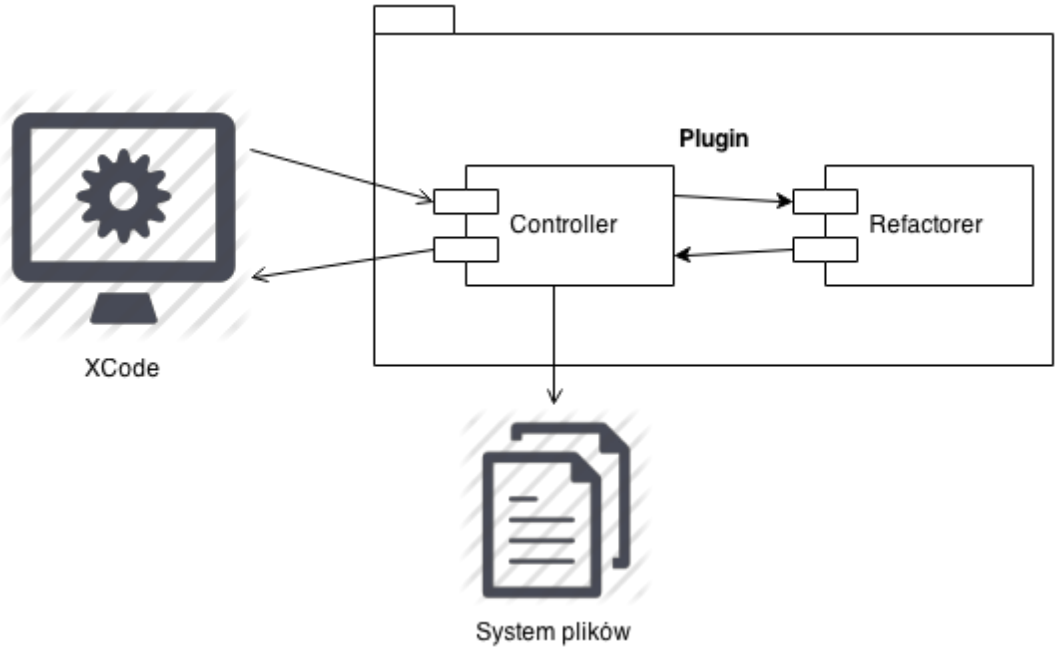
Aplikacja, będzie podzielona na dwa moduły:

1. Controller - moduł odpowiadający na komunikację z XCodem, napisany w języku Objective-C. Na starcie programu XCode, nasz moduł będzie rejestrował się jako słuchacz zdarzeń, które będą mu niezbędne do komunikacji z użytkownikiem. Ten moduł na podstawie zachowania użytkownika będzie wysyłał odpowiednio przetworzone dane do modułu refaktoryzującego
2. Refactorer - główny moduł programu, odpowiadający za faktyczną refaktoryzację kodu, napisany w języku C++/Objective-C. Moduł ten będzie otrzymywał takie dane na wejście jak:
    a. część nagłówkową kodu jako ciąg znaków
    b. tekst do refaktoryzacji znajdujący się w części nagłówkowej jako ciąg znaków
    c. część implementacyjną kodu jako ciąg znaków

    Natomiast na wyjście:

a. zrefaktoryzowaną część nagłówkową kodu jako ciąg znaków
b. zrefaktoryzowaną część implementacyjną kodu jako ciąg znaków

W tym module odbywać się będzie też parsowanie przed refaktoryzacja, która odbywać się będzie znak po znak.




W ramach projektu zostanie zrealizowany moduł refaktoryzujący.

## 4. Idea działania programu

Program po zainstalowaniu jako plugin do programu XCode, będzie uruchamiał się z każdą instancją tego programu. Użytkownik podczas pisaniu kodu w owym programie, będzie mógł w oknie edycji zaznaczyć tekst i wywołać odpowiednią funkcję refaktoryzującą poprzez wywołanie skrótu klawiszowego, z paska menu bądź z menu kontekstowego( **opcjonalnie** ). Zanim nastąpi refaktoryzacja, następuje zapisanie edytowanego pliku(.h) na dysk, lokalizacja pliku z implementacją (.m) i zapisanie go jeśli jest otwarty w którejś z zakładek programu. Następnie wczytanie pliku z implementacją i przekazanie go razem z nagłówkami do refaktoryzacji. Po przeprowadzeniu refaktoryzacji, zmiany zostają zapisane bezpośrednio na dysk.

## 5. Gramatyka

Gramatyka zapisana w notacji EBNF

class_component = class_interface | class_implementation ;
class_interface = “@interface ”, ciag_znaków, “:”, ciąg_znaków, [protocols], [variables_declarations], [properties_and_methods], “@end” ;

protocols = “<”, lista_ciągów, “>” ;

variables_declarations = “{“ { variable_declaration } “}”;
variable_declaration = variable,  “ “, {“*}, ciąg_znaków, “;” ;
variable = ciąg_znaków, { “*” } ;
properties_and_methods = {property_declaration | method_declaration }
property_declaration = “@property”, [property_atributes], variable_declaration ;
property_atributes = “(“, lista_ciągów, “)” ;
method_declaration = method_header, “;” ;
method_header = method_type, type_in_method, method_header_part* ;
method_type = “-” | “+” ;
type_in_method = “(“, variable, “)” ;
method_header_part = ciąg_znaków, “:”, type_in_method, ciąg_znaków ;

class_implementation = “@implementation”, [synthesize], [methods], “@end”
synthesize = “@synthesize”, synthesize_declaration, { “,”, synthesize_declaration }, “;” ;
synthesize_declaration = ciąg_znaków | ciąg_znaków “=” ciąg_znaków ;
methods = { method_implementation } ;
method_implementation = method_header, “{“, method_body, “}” ;
method_body = { ( “{“, method_body, “}” ) |  ? dowolny ciąg znaków bez {} ? } ;

litera = {“a” | .. | “z” | “A” | .. | “Z”} ;
ciag_znaków = { litera | “_” | cyfra } ;
lista_ciągów = ε | ciąg_znaków ,  {“,” ciąg _znaków }
cyfra = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

Komentarze:

Komentarz = '/*'...bez zagnieżdżeń...'*/' |  //....newline
Komentarze mogą pojawiać się między dowolnym tokenem

## 6. Projekt implementacji

Do dokumentacji załączono plik diagram.pdf w którym znajdują się diagramy procedur wykorzystywanych przez interpretator.

Struktury danych:

Tekst do parsowania, będzie trzymany w klasie implementującej dane metody:

```
char getChar();
char getNextSourceChar();
unsigned int getPos();
void setPos(unsigned int);
operator ++ (przesuwający pozycje bufora o jeden)
```

Oprócz czytania z bufora, trzeba będzie zapewnić też pisanie:

```
        void write(string);
```

Opis procedur:
---
char getChar();
Funkcja zwracający znak na aktualnej pozycji w buforze
---
Funkcja zwracający obecny znak, chyba że znak ten jest częścią komentarza (lub jest białym znakiem zależnie od parametru wejściowego), to wtedy następny znak pomijając komentarz ( i/lub białe znaki), przestawia pozycje wskaźnika.

```
char getSourceChar(bool skipBlanks = true)
        ch = bieżący znak
        if (skipBlanks)
                while ch jest białym znakiem
                        ch = następny znak
        if ch == /
                nextCh = następny znak
                if nextCh == *
                        przesuwaj wskaźnik bieżącego znaku aż napotkasz */
                        return getNextSourceChar
                if nextCh == /
                        przesuwaj wskaźnik bieżącego znaku aż napotkasz \n
                        return getNextSourceChar

return ch;
```

---
void write(string);
Wstawia dany ciag znaków do bufora na aktualnej pozycji
---

Parsowany kod przechowywany będzie w strukturach odpowiadającym poszczególnym części kodu. Np. sparsowany tekst: `@property (nonatomic, copy) NSDate* date;` przechowywany będzie w strukturze property o polach atrybuty i deklaracja, przy czym oprócz suchych faktów interesujących nas z punktu widzenia kompilatora, przechowywana będzie też pozycja w pliku, w którym rozpoczyna się opis danej struktury i pozycja w którym kończy się opis

Algorytm parsowania pliku .h:

```
// obiekt przechowujacy tekst do parsowania
 Bufor bufor;
List componentList;

void szukajComponentu()
{
```

```
     while(bufor.getSourceChar() != “@”)
          bufor++;

Component component =  parsujComponent()
 if(component != NULL)
componentList.append(component);

     if (!bufor.isEnd())
          szukajComponentu();
}


Component parsujComponent(){
   unsigned int poz = bufor.getPos();
//przesuwaj wskaźnik bufora i sprawdzaj czy wychodzi interface lub implementation
 jeśli w buforze jest interface
    return parsujInterface(poz)
jeśli w buforze jest implementation
    return parsujImplementation(poz)
 w innym przypadku
    return NULL;
}


Interface parsujInterface(poz){
          Interface interface;
          interface.setStartPoz( poz );
          string className = parsujString();
          interface.setClassName(className);
          if(  !(bufor.getSourceChar() = “:”) )
                    zakoncz działanie i poinformuj o złych danych wejsciowych
          string motherClass = parsujString();
          list protocols = parsujProtocols();
          list variables = parsujVariables();

          int declarationsStart = bufor.getPos();
          list declarations = parsujDeclarations()

          istotne zmienne (variables, declarationsStart, declarations) zapisz w polach interface

          return interface;
}

list parsujVariables(){
          list variables;
          if(bufor.getSourceChar() == “{“)
```

```
                while(bufor.getSourceChar() != “}){
                        Variable variable = parsujVariable();
                        variables.append(variable);
                }
        return variables;
}

Variable parsujVariable(){
        string type = parsujString();
        sprawdz ile nastepnych znaków jest “*” az do napotkania spacji
        sprawdz ile nastepnych znaków jest “*” po napotkaniu spacji
        dodaj odpowiednia ilosc gwiazdek do typu
        string name = parsujString();
        jesli getChar nie jest “;” poinformuj ze jest blad

        return Variable(type, name);
}

list parsujDeclarations(){
        char ch = bufor.getSourceChar();

        list declarations;
        jesli ch == “@”
                Property property = parsujProperty();
                declarations.append(property);
        jesli ch == “-” / “+”
                MethodDeclaration method = parsujMethodDeclaration();
                declarations.append(method);
        w innych przypadkach zakoncz prace i poinformuj o bledzie

        return declarations
}

Property parsujProperty(){
        jesli nastepne znaki nie ukladaja sie w ciag “property” zglos
        blad i przerwij prace

        Property property;
        property.setStartPoz(bufor.getPos());
        list atrybuty = parsujPropertyAttributes();
        Variable variable = parsujVariable();
        property.setVariable(variable);
        property.setEndPos(bufor.getPos());
}

MethodDeclaration parsujMethodDeclaration(){
```

```
        MethodDeclaration declaration;
        declaration.setStartPos(bufor.getPos())
        declaration.setHeader( parsujHeader );
        if(bufor.getSourceChar() != “;”)
                zglos blad

        bufor++;
        declaration.setEndPos(bufor.getPos());
}

MethodHeader parsujMethodHeader(){
        char ch = bufor.getChar();
        bool staticMethod;
        if(ch == “+”)
                staticMethod = true;
        else
                staticMethod = false;

        bufor++;

        string returnType = parsujTypeInMethod();

        list methodParts = parsujMethodParts;

        if(methodParts is empty)
                zglos blad! powinna byc przynajmniej jedna czesc

        MethodHeader method;
        zapisz wszystkie zmiennej do pol obiektu method

        return method;
}

list parsujMethodParts{
        list parts

        while getSourceChar != “;” || != “{
                parts.append( parsujMethodPart());

        return parts;
}

MethodPart parsujMethodPart(){
        string name = parsujString();
        if bufor.getSourceChar != “:”
                zglos blad
```

```
        bufor++;

        string type = parsujTypeInMethod();
        string variableName = parsujString();

        MethodPart part(name, type, variableName);
        return part;
}

string parsujTypeInMethod{
        if(bufor.getSourceChar() != “(“)
                zglos blad!

        string pRet = parsujString;

        if(bufor.getSourceChar() != “)“)
                zglos blad!

        return pRet;
}

string parsujString(){
        string ciag;
        ciag.append(bufor.getSourceChar())
        bufor++;
        while(bufor.getChar nie jest znakiem specjalnym ){
                ciag.append(bufor.getChar());
                bufor++;
        }
        return ciag;
}
```