XCODE PLUGIN

Dokumentacja koncowa

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>`
 	
`@interface Foo : NSObject{`
`  int _foold;`
`  NSString* _name;`
`  NSDate* _date;`
`  Foo* _another;`
`}`
 	
`@end`
	`#import "Foo.h"`
	
	`@implementation Foo`
	
	`@end`

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

Foo.h	Foo.m

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

Otrzymamy taki rezultat:

Foo.h	Foo.m

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

@end
```

Rezultat:

Foo.h

```
#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;
}

-(int)exampleIntReturnMethod{
#warning not implemented exampleIntReturnMethod
  return 0;
}

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

@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

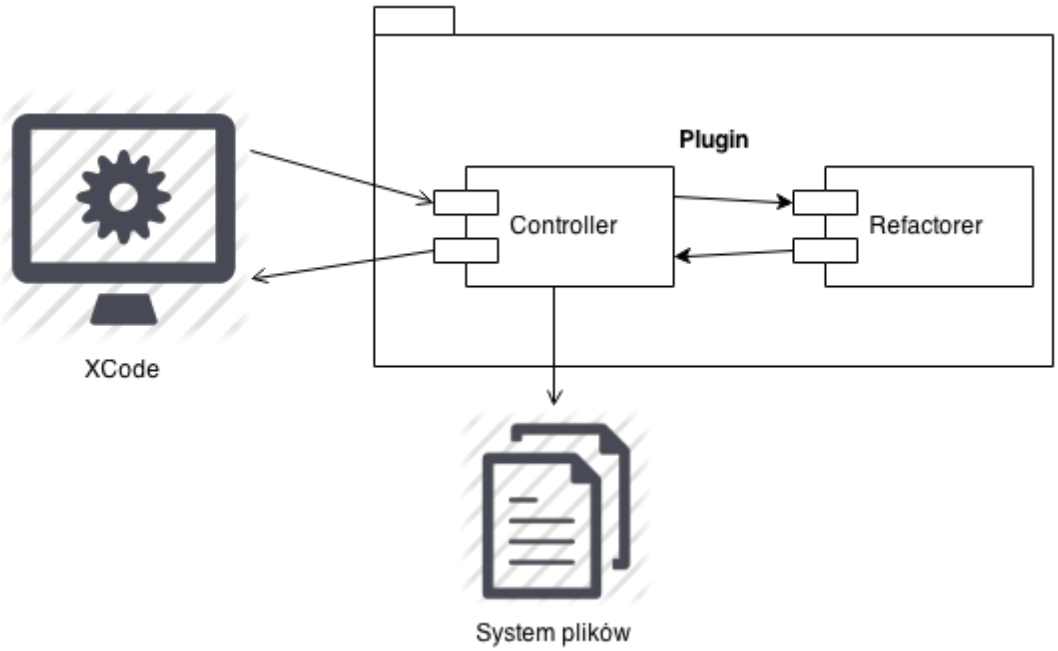
Aplikacja, będzie podzielona na dwa moduły:

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++. Moduł ten będzie otrzymywał takie dane na wejście jak:
    a. część nagłówkową kodu jako ciąg znaków
    b. część implementacyjną kodu jako ciąg znaków
    c. przedział w jakim znajduje się zaznaczony kod

    Natomiast na wyjście:
        a. zrefaktoryzowaną część nagłówkową kodu jako ciąg znaków

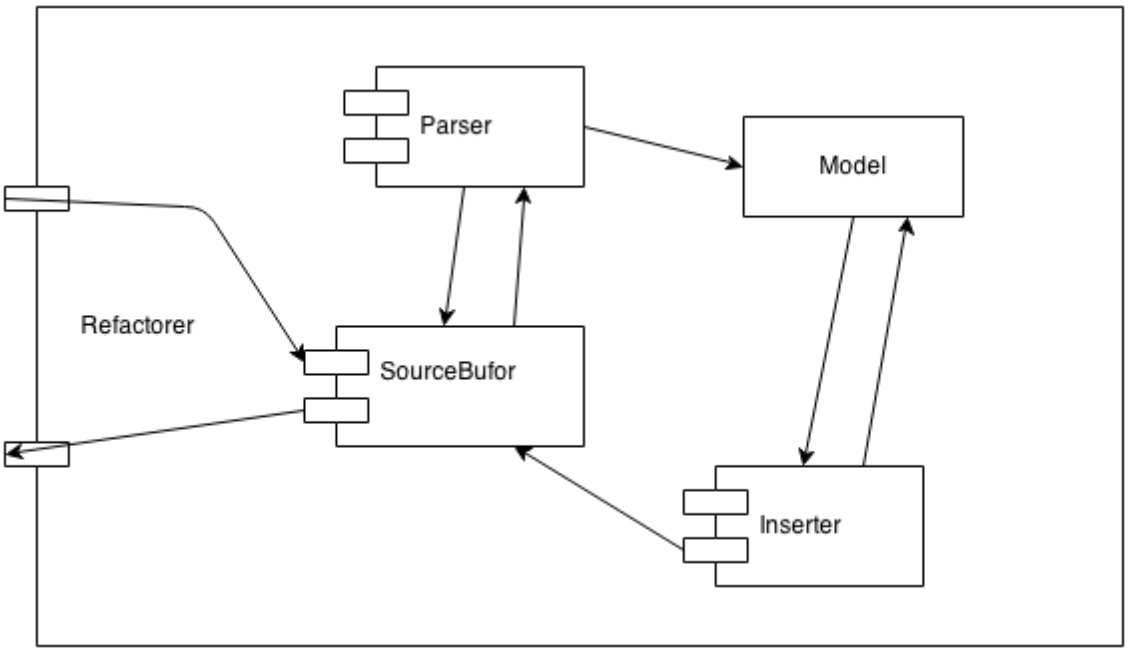
b. zrefaktoryzowaną część implementacyjną kodu jako ciąg znaków

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



W ramach projektu zostanie zrealizowany moduł refaktoryzujący.

Moduł refaktoryzujący:



Zrealizowany w postaci biblioteki, składającej się na poszczególne moduły:

- SourceBufor - moduł odpowiedzialny za czytanie kodu źródłowego, a także pisanie
- Parser - moduł tłumaczący kod źródłowy, który czyta zbBufora i tworzy na jego podstawie model

- Model - przetłumaczony kod na obiekty łatwe do czytania i modyfikacji
- Inserter - odpowiada za tworzenie kodu źródłowego na podstawie Modelu i taki kod zapisuje do bufora
- Refactorer - obiekt z którego korzystać będzie końcowy użytkownik biblioteki, zrealizowany jako funktor, poprzez przeciążenie operatora wywołania podawać będziemy wejście jako parametry funkcji w postaci referencji , a wyjście otrzymamy poprzez modyfikację parametrów wejściowych:

Refactorer
`virtual bool operator () (std::string &interface, std::string &implementation, unsigned startPos, unsigned endPos) = 0;`

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

7. Testy

Do testów została wykorzystana biblioteka Qt w wersji 5.0, dlatego też wymagana jest ona do uruchomienia ich:

- Testy jednostkowe:
  - Stworzone zostały dwa projekty testowe pokrywające większość funkcjonalności modułu, który testują:
    - BuforTest - projekt testujący bufor

Rezultat uruchomionego testu, aby zobrazować przypadki testowe:

```
********* Start testing of BuforTest *********
Config: Using QtTest library 5.2.0, Qt 5.2.0
PASS : BuforTest::initTestCase()
PASS : BuforTest::inkrementacja(Simply string)
PASS : BuforTest::inkrementacja(String with spaces)
PASS : BuforTest::inkrementacja(String with spaces and enter)
PASS : BuforTest::inkrementacja(String with //)
PASS : BuforTest::inkrementacja(String with /* */)
PASS : BuforTest::inkrementacja(String with and spaces //)
PASS : BuforTest::inkrementacja(String with /* */ and spaces)
PASS : BuforTest::inkrementacjaException(0)
PASS : BuforTest::inkrementacjaException(1)
PASS : BuforTest::inkrementacjaException(2)
PASS : BuforTest::inkrementacjaException(5)
```

PASS : BuforTest::dekrementacjaException()
PASS : BuforTest::moveByException(Pusty)
PASS : BuforTest::moveByException(Nie pusty)
PASS : BuforTest::setPos(Simply string)
PASS : BuforTest::setPos(String with spaces)
PASS : BuforTest::setPos(String with spaces and enter)
PASS : BuforTest::setPos(String with //)
PASS : BuforTest::setPos(String with /* */)
PASS : BuforTest::setPos(String with and spaces //)
PASS : BuforTest::setPos(String with /* */ and spaces)
PASS : BuforTest::getSourceChar(Simply string)
PASS : BuforTest::getSourceChar(String with spaces)
PASS : BuforTest::getSourceChar(String with spaces and enter)
PASS : BuforTest::getSourceChar(String with //)
PASS : BuforTest::getSourceChar(String with /* */)
PASS : BuforTest::getSourceChar(String with and spaces //)
PASS : BuforTest::getSourceChar(String with /* */ and spaces)
PASS : BuforTest::isEnd()
PASS : BuforTest::isEnd(1)
PASS : BuforTest::isEnd(12)
PASS : BuforTest::cleanupTestCase()
Totals: 33 passed, 0 failed, 0 skipped
********* Finished testing of BuforTest *********

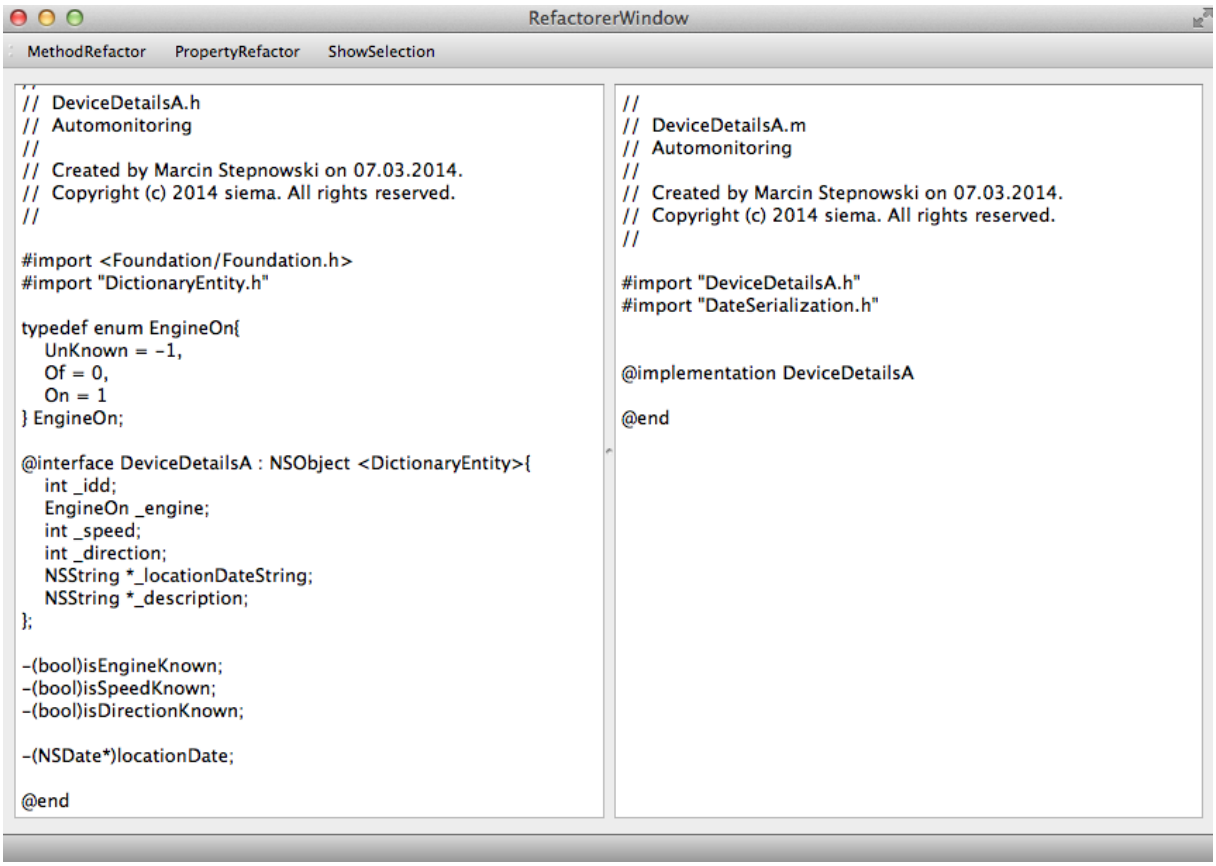
- ParserTest - projekt testujący moduł parsujący

Rezultat uruchomionego testu, aby zobrazować przypadki testowe:

********* Start testing of ParserTest *********
Config: Using QtTest library 5.2.0, Qt 5.2.0
PASS : ParserTest::initTestCase()
PASS : ParserTest::parseString(string)
PASS : ParserTest::parseString(string:qs)
PASS : ParserTest::parseString(_12string:qs)
PASS : ParserTest::parseString(_12string#qs)
PASS : ParserTest::parseString(_12string qs)
PASS : ParserTest::parseString(string qs)
PASS : ParserTest::parseString(/* kom */ string qs)
PASS : ParserTest::parseStringList(string, _string, STR1NG)
PASS : ParserTest::parseStringList(string , _string/**** ssss */ ,STR1NG:)
PASS : ParserTest::parseVariableNotPointer(int)
PASS : ParserTest::parseVariableDeclaration(NSData*** data;)
PASS : ParserTest::parseVariableDeclaration(NSData ***data;)
PASS : ParserTest::parseVariableDeclaration(NSData* * * data;)
PASS : ParserTest::parseVariableDeclaration(NSData** * data;)
PASS : ParserTest::parseVariableDeclaration(NSData *** data;)
PASS : ParserTest::parsePositioning(Bez komentarzy i bialych znakow)
PASS : ParserTest::parsePositioning(Biale znaki)
PASS : ParserTest::parsePositioning(Komentarze)
PASS : ParserTest::parsePropertyDeclarationException())
PASS : ParserTest::parsePropertyDeclarationException(;)
PASS : ParserTest::parsePropertyDeclaration(Z atrybutami)
PASS : ParserTest::parsePropertyDeclaration(Bez atrybutami)

PASS : ParserTest::parseMethodHeaderPart()
PASS : ParserTest::parseMethodHeader()
PASS : ParserTest::parseMethodHeaderDeclaration()
PASS : ParserTest::parseMethodDefinition(Bez zagniezdzen)
PASS : ParserTest::parseMethodDefinition(Z zagniedzeniami)
PASS : ParserTest::parseSynthesizedVariable(Samo property)
PASS : ParserTest::parseSynthesizedVariable(property = variable)
PASS : ParserTest::parseSynthesizeBlock(puste)
PASS : ParserTest::parseSynthesizeBlock(1)
PASS : ParserTest::parseSynthesizeBlock(3)
PASS : ParserTest::parseClassInterface(Pusta)
PASS : ParserTest::parseClassInterface(Protocols)
PASS : ParserTest::parseClassInterface({})
PASS : ParserTest::parseClassInterface(property)
PASS : ParserTest::parseClassInterface(method)
PASS : ParserTest::parseClassInterface(method i property)
PASS : ParserTest::parseClassImplementation(Pusta)
PASS : ParserTest::parseClassImplementation(Synthesize)
PASS : ParserTest::parseClassImplementation(Synthesize i method)
PASS : ParserTest::parseClassImplementation(method)
PASS : ParserTest::parseFile(InterFace)
PASS : ParserTest::parseFile(Implementation)
PASS : ParserTest::cleanupTestCase()
Totals: 46 passed, 0 failed, 0 skipped
********* Finished testing of ParserTest *********

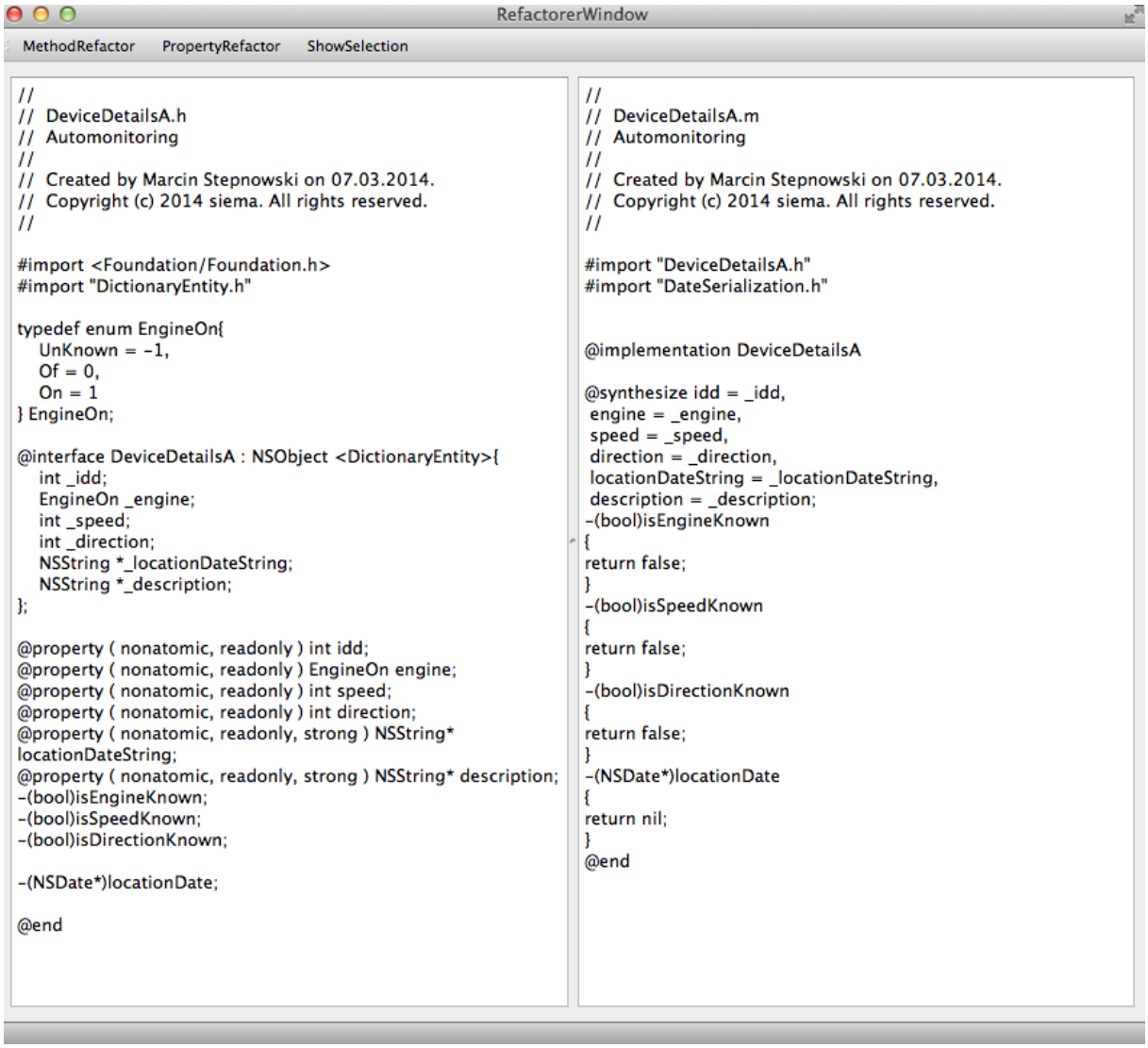
- GUITest - Program okienkowy, “przypominający” IDE w którym pisząc kod mamy do dyspozycji obie funkcje refaktoryzujące



Po zaznaczeniu tekstu możemy wybrać jedną z funkcji z nad obszaru tekstowego:

- MethodRefactor - refaktoryzacja metod
- PropertyRefactor - refaktoryzacja atrybutów
- ShowSelection - opcja pomocna w testowaniu biblioteki, pokazuje pozycje zaznaczenia obu kodów źródłowych w pasku statusu

Po zaznaczeniu odpowiednio metod, zmiennych i uruchomieniu refaktoryzacji nasze okno wygląda tak:



W przypadku gdyby refaktoryzacja się nie powiodło, czy też nie była potrzebna, otrzymamy odpowiednią informacje, w przypadku niepowiedzenia informacją będzie treść wyjątku

rzuconego przez praser bądź bufor.