Programowanie obiektowe

Wykład 14: Dokumentacja kodu

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.3
4 marca 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Dokumentacja kodu

- Aktualnie oprogramowanie jest najczęściej tworzone przez zespoły programistyczne.
- Z tego powodu staje się szczególnie ważne, aby nasz kod był zrozumiały.
- Oprócz pisania czytelnego kodu możemy tworzyć jego dokumentację.
- Java oferuje bardzo wygodne narzędzie do dokumentacji kodu `javadoc`.

Komentarze dokumentacyjne

- Java pozwala na częściową integrację dokumentacji z kodem.
- Dokumentacja jest tworzona w plikach z kodem, a następnie przetwarzana do postaci HTML.
- Treść dokumentacji znajduje się w specjalnych znacznikach `/** dokumentacja */`.
  - Należy uważać, aby nie pomylić dokumentacji z komentarzem `/* komentarz */`.
- W komentarzach można umieszczać wybrane znaczniki HTML i specjalne adnotacje.
- Pierwsze zdanie dokumentacji służy jest wykorzystywane jako streszczenie.
- Położenie znaczników w kodzie określa jaki fragment kodu jest dokumentowany.

Możliwości dokumentacji

- Dokumentować można następujące elementy kodu:
  - pakiety,
  - klasy i interfejsy,
  - metody i konstruktory,
  - pola.
- Ponieważ dokumentacja jest przeznaczona dla zewnętrznych użytkowników naszego kodu to nie ma sensu komentować prywatnych elementów pakietów i klas.
- Bardziej kontrowersyjna jest kwestia czy należy dokumentować *każdy* publiczny element [Martin, 2010].

Używanie znaczników HTML

- Możemy tworzyć dokumentację korzystając HTML, dzięki czemu będzie ładnie wyglądać w przeglądarce.

Krótkie stresczenie działania metody.

1. Możemy generować fikuśny opis w **HTML**,
2. należy jednak uważać, aby opis *nie przestał być czytelny* w środowisku programistycznym.
3. Przy okazji, metoda `generateHTML` nic nie robi.

Rysunek 1: Wygenerowana dokumentacja z użyciem HTML

- Należy jednak pamiętać, że dokumentacja powinna być także czytelna w plikach z kodem.

```
/**
* Krótkie streszczenie działania metody.
* <OL>
* <LI>Możemy generować fikuśny opis w <strong>HTML</strong>,</LI>
* <LI>należy jednak uważać, aby opis <em>nie przestał być
     czytelny</em> w środowisku programistycznym.</LI>
* <LI>Przy okazji, metoda <code>generateHTML</code> nic nie
     robi.</LI>
* </OL>
*/
```

Dokumentacja jako hypertext

- Tworzona dokumentacja ma charakter Hypertekstu.
- Odpowiednie adnotacje tworzą odwołania do innych części dokumentacji

@see Odnośnik do dokumentacji innych klas.

- `@see classname`
- `@see fully-qualified-classname`
- `@see fully-qualified-classname#method-name`

@link Odnośnik umieszczony wewnątrz tekstu.

- `@link package.class#member label`

@docRoot Odnośnik do początku dokumentacji.

@inheritDoc Odnośnik do dokumentacji klasy nadrzędnej.

Komentarze do metod

- Komentując metody możemy używać specjalnych adnotacji.

  @param Nazwa i opis parametru metody.
  - `@param name description`

  @return Opis zwracanej wartości.

  @throws Informacja o wyjątkach rzucanych przez funkcję.
  - `@throws full_nazwa description`

  @deprecated Informuje, że metoda nie powinna być używana.

Przykład komentowania metod

```
/**
     * Wyliczenie mody. Wskazuje najczęściej występujący element na liście
         timeSeries
     * @param timeSeries lista elementów.
     * @param <T> dowolny typ.
     * @return wyliczona moda typu T.
     * @throws NullArrayList wyrzucany dla pustego argumentu timeSeries.
     */
    public static <T> T calculateMode(ArrayList<T> timeSeries) throws
         NullArrayList
```

calculateMode

```
public static <T> T calculateMode(java.util.ArrayList<T> timeSeries)
                           throws pl.edu.pw.mini.mluckner.op.lecture14.NullArrayList
```

Wyliczenie mody. Wskazuje najczęściej występujący element na liście timeSeries

Type Parameters:

`T` - dowolny typ.

Parameters:

`timeSeries` - lista elementów.

Returns:

wyliczona moda typu T.

Throws:

`pl.edu.pw.mini.mluckner.op.lecture14.NullArrayList` - wyrzucany dla pustego argumentu timeSeries.

Rysunek 2: Wygenerowana dokumentacja

Inne znaczniki dotyczące komentarzy

@version wersja kodu,

@author autor kodu,

@since wersja od której wprowadzono dany element

Adnotacje

- Adnotacje nie są wykorzystywane tylko do tworzenia dokumentacji.
- Są uniwersalnym mechanizmem, który nie wpływa bezpośrednio na program, a może być interpretowany przez przez narzędzia i biblioteki, które wpłyną na działanie programu.
- Mogą być odczytane z plików źródłowych, plików klas lub podczas wykonywania programu.

Adnotacje informujące

@Override Informacja o nadpisaniu elementu. przydatne, jeżeli chcemy się upewnić, że nadpisujemy istniejącą metodę

@Deprecated Informuje, że metoda nie powinna być używana.

@SupressWarnings Ogranicza wyświetlanie ostrzeżeń.

```
@SupressWarnings("deprecation","serial")
```

Adnotacje kontrolujące

@Readonly Kontroluje czy argumenty nie zostały zmienione

```
public String getText(@Readonly Paper
sheetOfPaper)
```

@NotNull Kontroluje czy zwracamy nie pustą wartość

```
@NotNull String text =
sheetOfPaper.getText();
```

Bibliografia

[Martin, 2010] Martin, R. C. (2010).
Czysty kod. Podręcznik dobrego programisty.
Helion.