Лекция 9 (29. Ноември, 2011)

Links
http://codesion.com/
http://en.wikipedia.org/wiki/Javadoc
http://www.stack.nl/~dimitri/doxygen/

Първи час
Едно от нещата в съвременното програмиране, които не само не се осъвършенстват ами напротив – по-скоро отпадат – е писането на документация към кода. Навремето това е било нещо задължително, но когато програмираш чрез перфокарти или на асемблер някак си е нормално да документираш и какво точно прави кодът ти. Тъй като шансовете някой да мине на следващия ден и да нащрака още няколко дупки по перфокартата (перфолентата) ти е сравнително малък. Документирането на код, макар и полезно, е реализируемо предимно при завършени или почти непроменящи се проекти (или части от проекти). Когато се пише толкова динамичен софтуер, че всяко нещо бива променяно ежедневно, поддържането на документация е по-скоро пречка, отколкото нещо удобно. Затова и в днешно време писането на документация е (освен крайно досадна) работа с нисък приоритет, оставяна за накрая, незавършвана или давана на новите програмисти/стажанти. Тези от вас, които са работили поне малко знаят за какво говоря.

От друга страна, когато някой програмист напусне и някой друг заеме мястото му, е НАИСТИНА важно да има документация на направеното до сега, за да може новият кодер да навлезе по-бързо в работата, без да изхабява десетки часове в разглеждане и разгадаване на чужд код и логика. Тоест нужда от документация все пак има. Да не говорим, че ако разработвате софтуер за по-голяма компания, вие освен source code-а задължително им предавате и документация за него. Тоест можете да си позволите евентуаааално да няма такава ако сте най-високо в йерархията (тоест последната стъпка преди компилиране на изпълними файлове), но не и иначе.

Как да се справим с това, че тя от една страна ни трябва, а от друга ни спъва? Няма ли друг живот?
Има, разбира се! Програми за генериране на документация!
Какво е това животно и има ли то почва у нас? Представете си, че просто пишем код, без да се интересуваме за документацията му. Както казахме в началото на курса, избор на фиксирана структура на кода (naming convention, indentation, etc.) помага много за четимостта на кода както от други програмисти, така и от нас самите. Мислите си, че нямате проблем с код, който вие самите сте писали? Направете си експеримент – намерете сорс, който сте писали преди повече от година, и се опитайте да разберете какво прави стъпка по стъпка. Ще ви отнеме доста време. Ето и един от любимите ми коментари, срещани в код:

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```

(впрочем, можете да видите пълната класация в stackoverflow ето тук: http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered)
Писането на коментари из кода също помага доста за неговото разбиране (и тук разбирайте коментари, които описват ЛОГИКАТА на кода, а не синтаксиса – с него се очаква програмистите да са запознати). Ако пишем коментари пред всяка функция, описвайки какво прави тя, кодът ни би бил значително по-лесно разбираем за „външен" човек, тоест това би било нещо, което не отнема чак толкова време, а е бонус. Защо да не доразвием идеята нататък, като от въпросните коментари генерираме документация? Това е именно и идеята на програмите за автоматично генериране на документация от типа на Javadoc.

За да може една такава програма да функционира адекватно е нужно да са изпълнени няколко изисквания.

- Функциите/методите да са коментирани, да са описани какво връшат, както какви аргументи получават и какво връщат.
- Коментарът да е с някакъв фиксиран (специфичен) формат, за да може да бъде parse-нат от някакъв автомат.

Javadoc коментарите са нормални коментари от типа /*...*/, с тази разлика, че отварящия коментар е с двойна звездичка (тоест "/**"). Забележете, че от гледна точка на препроцесора (частта от компилатора, която parse-ва кода за да види синтактични грешки) не прави разлика между нормален и такъв коментар – за него втората звездичка е просто част от коментара. Оттам нататък в коментара се очаква да има (някои от):

- Описание на функцията, метода или класа, който следва
- Аргументи, които функцията или класът получават
- Какво връща фукнцията или мотодът ако не е void
- Кой е автор на въпросния код
- Дали (и ако да – какви) exception-и може да хвърли даденото парче код
- Версия на кода, или от коя версия на компилатора насам се поддържа

Ето един примерен такъв коментар:

```
/**
 * @author      Firstname Lastname <address @ example.com>
 * @version     2010.0331                   (E.g. ISO 8601 YYYY.MMDD)
 * @since       1.6                         (The Java version used)
 */
public class Test {
  // class body
}
```

В дадената таблица са описани възможните тагове, които можем да имаме в един javadoc коментар.

Tag & Parameter	Usage	Applies to	Since
@author *name*	Describes an author.	Class, Interface	
@version *version*	Provides version entry. Max one per Class or Interface.	Class, Interface	
@since *since-text*	Describes since when this functionality has existed.	Class, Interface, Field, Method	
@see *reference*	Provides a link to other element of documentation.	Class, Interface, Field, Method	
@param *name description*	Describes a method parameter.	Method	
@return *description*	Describes the return value.	Method	
@exception *classname description* **@throws** *classname description*	Describes an exception that may be thrown from this method.	Method	
@deprecated *description*	Describes an outdated method.	Method	
{**@inheritDoc**}	Copies the description from the overridden method.	Overriding Method	1.4.0
{**@link** *reference*}	Link to other symbol.	Class, Interface, Field, Method	
{**@value**}	Return the value of a static field.	Static Field	1.4.0

Ето и пример за коментар пред функция/метод, който проверява дали даден шахматен ход е валиден:

```
/**
 * Validates a chess move.
 * @param fromRow row from which a piece is being moved
 * @param fromCol column from which a piece is being moved
 * @param toRow row to which a piece is being moved
 * @param toCol column to which a piece is being moved
 * @return true if the move is valid, otherwise false
 */
boolean isValidMove(int fromRow, int fromCol, int toRow, int toCol)
{
   ...
}

/**
 * Moves a chess piece.
 *
 * @see java.math.RoundingMode
 */
boolean doMove(int fromRow, int fromCol, int toRow, int toCol)
{
   ...
}
```

Ползването на автоматично-генерирана документация улеснява и ускорява ЗНАЧИТЕЛНО не само създаването на документацията, а също така и нейната поддръжка. Представете си, че искаме да променим даден написан код, към който има документация. След модификацията на source-а е нужно да отворим документацията, да намерим къде точно е описана въпросната функция, и да променим документацията. След това трябва да commit-нем не само променения код, а също така и променената документация. От друга страна, ако тя се генерира автоматично, можем да променим само функцията и коментара над нея – постигайки абсолютно същия ефект.