KONVENCIE PÍSANIA KÓDU V PROJEKTE JIM

a k nemu prislúchajúcim projektom

Juraj Šimek 2014

Tento dokument opisuje, aké konvencie je nutné dodržiavať pri dopĺňaní kódu agenta a s ním súvisiacich projektov. Agent je programovaný v jazyku *Java* a preto je vhodné dodržiavať konvencie uvedené v dokumente *Java Code Conventions* [1]. Tieto konvencie sú stručne uvedené nižšie s prípadnými drobnými zmenami, preto aj ten, kto ich pozná, by si mal tento dokument prečítať. Ak má autor dobrý dôvod na zmenu programátorského štýlu, môže túto zmenu vo svojom kóde uskutočniť, pričom ku tomuto bloku kódu uvedie komentár, prečo zmenu vykonal.

KONVENCIE PÍSANIA KÓDU V JAZYKU JAVA

NÁZVY

Názvy *tried*, *rozhraní* a *výčtových typov* začínajú veľkým písmenom. Je dôležité vyberať stručné ale výstižné názvy tried. Pri viacslovných pomenovaniach sa používa camel-case.

Príklady:
```
class ThisIsMyClass { }
interface MyInterface { }
```

Názvy *premenných* a *metód* začínajú malým písmenom. Pri viacslovných pomenovaniach sa znova použije camel-case. Znova je dôležité dbať na to, aby bol názov zrozumiteľný, ale pritom stručný.

Príklady:
```
void myNewMethod() { ... }
int someVariable = 40;
```

Názvy konštánt sa uvádzajú veľkým písmom. Slová sa oddeľujú podtržníkom (_).

Príklady:
```
public final static int THIS_IS_CONSTANT = 30;
```

Názvy balíkov sa začínajú malým písmenom.
```
sk.fiit.jim.agent
```

FORMÁTOVANIE KÓDU

Každý kód vo vnútri akéhokoľvek bloku musí byť odsadený **tabulátorom**:

```
{
	int x = 4;
}
```

Otváracia zátvorka označujúca blok kódu sa nachádza hneď **za** príkazom, ku ktorému prislúcha:

```
if (x == 3) {
      // code
}
```

Telo každého *cyklu* a podmieneného príkazu je uvedené v zátvorkách **{** a **}** **aj v prípade, že predstavuje len jeden príkaz!**

```
if (x == 3) {
      return false;
} else {
      return true;
}
```

Toto je nesmierne dôležité kvôli predchádzaniu chybám pri rozširovaní kódu v budúcnosti.

Každý *príkaz* sa nachádza na **osobitnom riadku**:

```
numberA++;
numberB++;
```

Každú *premennú* je vhodné definovať na novom riadku.

```
int myVariableA = 4;
int anotherVariable;
int thisVariableIsAlsoNeeded;
```

Dĺžka riadku by nemala presiahnuť 80 znakov. Ak je príkaz príliš dlhý rozdelí sa, pričom sa rozdelená časť vhodne odsadí a operátory sa nachádzajú na novom riadku. Bližšie informácie v kapitole Wrapping Lines dokumentu [1]. V tomto prípade treba pamätať hlavne na to, aby bolo pri pohľade na kód jasné, čo patrí spolu, a čo je už iný príkaz.

Medzi definíciami metód v triede uvedieme jednu medzeru:

```
public String toString() {
      // code
}

/**
 * Comment
 */
public int getThisNumber() {
      // code
}
```

SERIALIZÁCIA

Ak trieda požaduje implementovanie rozhrania `Serializable` a v rámci toho definíciu `serialVersionUID`, **neuvádza sa** `@SupressWarning("serialization")`, ale do vnútra triedy sa uvedie definícia, napríklad `static final long serialVersionUID = 42L;` Číslo `serialVersionUID` môže byť samozrejme ľubovoľné. Viac o jeho význame a o serializácii objektov je v [3] a [4].

KOMENTÁRE

Jednoriadkové komentáre sa píšu výlučne pomocou `//` a neuzatvárajú sa medzi **/*** a ***/**

```
// single line comment
```

Ak má komentár viac riadkov, napíše sa medzi `/*` a `*/`, pričom je vhodné uviesť na každom riadku znak *. Medzi znakom * a prvým písmenom komentára je jedna medzera.

```
/*
 * This is
 * Multiline
 * Comment
 */
```

DOKUMENTAČNÉ KOMENTÁRE

Pred každou triedou a metódou, ktorá je verejná sa uvedie dokumentačný komentár medzi `/**` a `*/` generujúci JavaDoc. Viac informácií o písaní JavaDocu je v [2]. Nepoužívajú sa žiadne vymyslené tagy. JavaDoc pripúšťa tieto tagy:

```
@author
@version
@param
@return
@exception
@see
@since
@serial
@deprecated
```

Význam jednotlivých tagov je uvedený v [2].

Každý *autor* uvedie do dokumentačného komentára kódu, ktorému prislúcha svoje meno v nasledovnom tvare:

`@author meno_autora (nazov_timu - rok)`
`meno_autora` – autorovo celé meno
`nazov_timu` – meno tímu, ak v tíme nepracuje, uvedie len rok
`rok` – rok v ktorom je kód upravovaný

Príklad:
`@author Juraj Simek (Infinity – 2014)`

Pred každou *public* metódou treba do komentára uviesť pomocou `@param` význam jednotlivých parametrov, pomocou `@exception` opísať kedy vyhadzuje akú výnimku a potmocou `@return` napísať, čo vracia:

```
/**
 * Comment to the meaning of the method.
 *
 * @param someParam description
 * @return number of static pages
 * @exception Exception if there is no file with selected name
 *
 */
 public int doSomethingElse(Object someParam) throws Exception{
      // implementation
 }
```

Dokumentačné komentáre musia byť stručné, ale predovšetkým zrozumiteľné, aby sa v nich vyznali aj tí, čo daný kód neprogramovali.

VÝNIMKY

Ak zachytávame výnimku, je potrebné ju zalogovať. Nikdy možné výnimky neignorujeme pomocou catch(Exception e) {} ! Vhodné je buď výnimku zachytiť a lognúť a potom ošetriť výnimočný stav, alebo ju znova vyhodiť a ošetriť prípadný výnimočný stav v inej časti kódu, ak to v tejto časti nie je možné. Nasledovný kód by sa preto nemal vyskytovať často:

```
catch(NoSuchMethodException e) {
      LOG.error("Situation description", e);
      throw e;
}
```

Niekedy je však vhodné zalogovať, že výnimka niekde nastala a vyhodiť ju. Preto je vyššie uvedený kód povolený v situáciách, keď je to opodstatnené.

LOGOVANIE

V metóde, kde sa loguje treba získať logger pomocou `JLog.getLogger()`. Ideálne je vytvoriť členskú premennú pre celú triedu, ktorá bude logger obsahovať:

```
private final static Logger LOG = JLog.getLogger();
```

Loguje sa potom pomocou štandardného API triedy Logger [5]. Pre úplnosť informácií v krátkosti uvádzame to, ako sa loguje pomocou tohto rozhrania.

Štandardne logger vytvoríme ako privátnu statickú konštantu v triede, ktorá ho používa. Vďaka tomu budeme môcť zachytávať, v ktorej triede vznikla udalosť logovania. Loggeru dáme identifikátor, ktorý najčastejšie predstavuje meno danej triedy:

```
private final static Logger LOG =
Logger.getLogger(LoggingExamples.class.getName());
```

Logovanie v Jimovi však použije predkonfigurovaný globálny logger pomocou `JLog.getLogger()` tak, ako to bolo opísané vyššie.

Keď sme získali Logger, v akejkoľvek metóde danej triedy potom možno tento logger použiť na logovanie rôznych udalostí. Java API predpisuje nasledovné najčastejšie používané úrovne logovania `Level.FINEST`, `Level.FINER`, `Level.FINE`, `Level.INFO`, `Level.WARNING`, `Level.ERROR a Level.SEVERE`, kde `FINEST – FINE` sú najnižšei úrovne logovania a logujú sa nimi najmenej podstatné informácie, `INFOm` logujeme len nejaké informácie, ktoré chceme vložiť do logu, `WARNING` slúži na logovanie varovaní a `ERROR` so `SEVERE` na logovanie chýb, kde `SEVERE` je závažnejšia chyba ako `ERROR`. Loguje sa pomocou metódy log:

```
LOG.log(Level.INFO, "Information");
```

Pre základné typy logovania definované Java API loggerom možno použiť aj logovanie pomocou predpísaných metód:

```
LOG.info("Information");
```

Logger JLog používaný v projekte Jim navyše, oproti úrovniam v triede `Level` sú definované tieto typy logov:

```
LogType.INIT
LogType.AGENT_MODEL
LogType.WORLD_MODEL
LogType.INCOMING_MESSAGE
LogType.OUTCOMING_MESSAGE
LogType.LOW_SKILL
LogType.HIGH_SKILL
LogType.GUI
```

Tieto typy umožňujú logovať informácie pre špeciálne udalosti. Pomocou týchto typov logujeme

len informácie do úrovne `Level.INFO` štandardného Java API loggera (tj. `Level.FINEST`, `Level.FINER`, `Level.FINE`, `Level.INFO`), pričom ak môžeme použiť jeden z týchto rozšírených typov, použijeme ho. Ak chceme logovať nejakú udalosť, ktorú `LogType` nepredpisuje, buď ju do `LogType` pridáme, alebo použijeme `Level.INFO` (prípadne nejaký iný level nižší ako `Level.INFO`). Nový typ logu možno pridať do triedy `LogType` pridaním nasledovného riadku v zozname definícií konštánt:

```
public static LogType NEW_TYPE_NAME = new
LogType("NEW_TYPE_NAME", BASE_LEVEL_NUMBER + NEXT_NUMBER);
```

NEW_TYPE_NAME predstavuje názov nového logovacieho typu a píše sa veľkými písmenami, nakoľko ide o konštantu. *NEXT_NUMBER* udáva poradové číslo nového levelu a zapíše sa ako číslo o jednotku väčšie od najväčšieho čísla, ktoré používa nejaký typ logu definovaný v triede `LogType`.

Ak by nastala v nejakej triede chyba, alebo potrebujeme zalogovať varovanie, použijeme na to Level.ERROR, Level.SEVERE alebo Level.WARNING (alebo LogType.ERROR, LogType.SEVERE, LogType.WARNING, keďže trieda LogType dedí od triedy Level). Napríklad v prípade použitia `LogType.LOW_SKILL` môžeme zrozumiteľne logovať rozhodnutia na úrovni low skillov. Do triedy `LogType` možno pridať aj vlastné typy logov, ak je to potrebné, pričom treba sledovať logiku pridávania týchto typov v triede `LogType`. Význam jednotlivých typov je nasledovný:

- *INIT*: Inicializačné správy napríklad vo funkcii main(), nadviazanie spojenia so serverom, načítanie XML súborov a všetky akcie, ktoré konfigurujú agenta skôr, ako začne odosielať nejaké dáta na sever.
- *AGENT_MODEL*: Logy týkajúce sa triedy AgentModel.
- *WORLD_MODEL*:
- *INCOMING_MESSAGE*: Logovanie správy prichádzajúcej zo servera alebo logovanie informácií z funkcií, ktoré takúto správu spracúvajú.
- *OUTCOMING_MESSAGE*: Logovanie správy odosielanej na server alebo logovanie informácií z funkcií, ktoré takúto správu spracúvajú a odosielajú.
- *LOW_SKILL*: Informačné logy týkajúce sa low skillov.
- *HIGH_SKILL*: Informačné logy týkajúce sa high skillov.
- *GUI*: Informačné logy týkajúce sa grafického rozhrania robota.

Logger sa konfiguruje vo funkcii `main()`. Pred jeho prvým spustením treba zavolať metódu `JLog.setup()`, ktorá prijíma 2 boolean hodnoty. Prvá určuje, či sa bude logovať na konzolu, druhá, či sa bude logovať do súboru. Súbor sa nachádza v koreňovom adresári projektu a má názov *jim_logs.html*. Logy sú uložené v prehľadnej *HTML* tabuľke.

Bez ďalších nastavení sa logger obalený triedou `JLog` správa ako štandardný `Logger` z Java API. Ak chceme logovať aj dodatočné typy logov menované vyššie, musíme v main() tieto typy pridať pomocou metódy `addLogType()`:

```
JLog.addLogType(LogType.INCOMING_MESSAGE);
```

Ak chceme pridať všetky typy, ktoré definuje LogType, zavoláme metódu `addAllLogTypes()` triedy JLog. Potolm možno logovať tieto nové typy pomocou metódy log triedy Logger:

```
LOG.log(LogType.INCOMING_MESSAGE, "asdfasdf");
```

Logy by mali byť zrozumiteľné. Každý zápis do logu má mať opodstatnenie a má byť zmysluplný a formulovaný tak, aby sa dalo pochopiť, čo sa udialo a prečo. Logovať treba len pomocou tu opísaného API. Na logovanie sa nepoužívajú vlastné metódy a **hlavne nie System.out.println().** Nie je vhodné definovať ďalšie Logger-y ale treba používať ten, ktorý je nakonfigurovaný pomocou JLog.

Vhodné je obmedziť sa len na používanie metód `log(), finest(), finer(), fine(), config(), info(), warning(), severe()`.

ĎALŠIE KONVENCIE

Metódy, ktoré vracajú kolekcie alebo polia nesmú vracať null. Namiesto null je potrebné vrátiť prázdnu kolekciu alebo prázdne pole.

Vyhnite sa skratkám. Všeobecne známe skratky sú v poriadku.

Ak chcete niečo vytvoriť, najprv sa pozrite do dokumentácie, či na to neexistuje trieda a až vtedy ak neexistuje ju vytvorte. Neprogramujte veci ako hešovacie tabuľky, ktoré má Java v knižniciach.

Používajte výhody generického programovania pri práci s kolekciami. Vždy uvádzajte do zátvoriek < a >, aké typy sa v danej kolekcii budú používať.

ZDROJE

[1] Java Code Conventions: http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

[2] How to Write Doc Comments for the Javadoc Tool: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

[3] Implementing Serializable: http://www.javapractices.com/topic/TopicAction.do?Id=45

[4] Java Serialization Example Tutorial, Serializable, serialVersionUID: http://www.journaldev.com/2452/java-serialization-example-tutorial-serializable-serialversionuid

[5] Java Logging: http://tutorials.jenkov.com/java-logging/index.html