CONSULTANT TRACKER
CODING STANDARDS

EDITED BY

SIBEKEZELO MAMBA 16095414

JOHAN DE WAAL 16155140

STEPHEN MUNRO 16024479

HULISANI MUDIMELI 16073364

NGONIDZASHE MUJURU 16285256

TATENDA MAFUNGA 16094965

University of Pretoria

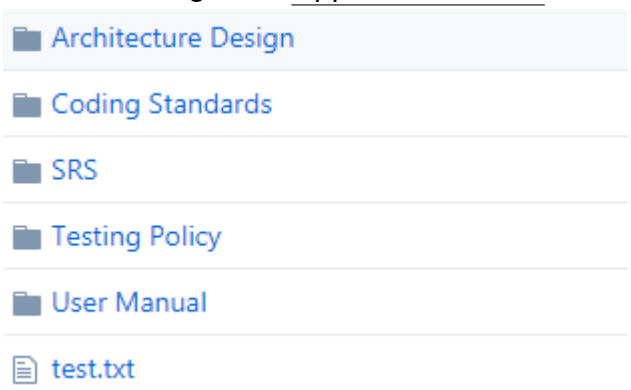
2018

Contents

1 Github Repository

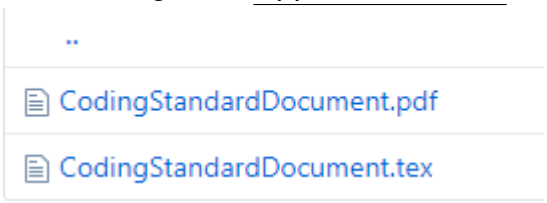
1.1 Folders

Folder naming uses UpperCamelCase



1.2 Files

File naming uses UpperCamelCase

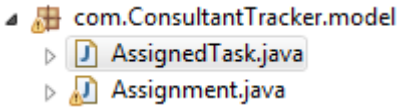


2 Overview

The following coding conventions pertain to how code is written and structured for the Consultant Tracking project. The standards are developed to be followed regardless of programming language used.

3 File Names

The preferred file naming style is UpperCamelCase



4 Formatting

4.1 Line Length

- The maximum number of characters in a line should be 80.

- Control structure conditions may exceed 80 characters, if they are simple to read and understand.

- Statements containing queries that are clear may exceed 80 character:

Example:

```
Connection connection = (Connection) getServletContext().getAttribute("DBConnection");
```

- Lines containing method/class names or variable declarations may exceed 80 characters.

Example:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servlet
    // TODO Auto-generated method stub
    doGet(request, response);
}
```

4.2 Line Wrapping

Conditions should not be wrapped in multiple lines, it is recommended to split the conditions.

Do not use multiple statements on a single line if the length is going to

exceed 80 characters.
Introduce line break after the opening and closing braces.

If a Line is too long because it contain a long expression, wrap using the following guides:
1. Divide the expression into smaller subexpressions.
2. Introduce a line break after each comma in the expression, and align each expression with the first character of the expression preceding the comma.
3. Introduce a line break after the operator with the lowest precedence, until the line does not exceed the 80 character limit.

4.3 Spacing

One blank line should be used in the following cases:
•between methods
•Between the local variables in a method and its first statement.

4.4 Use Of Parentheses

Use parentheses on expressions containing mixed operators, do not assume other programmers understand precedence as well you as you do.

4.5 Commenting

Each class or function should contain at least one sentence that provide a description or summary of the entity.
Block comment may be used at the beginning of each file.

line comments must be followed by a newline, for easy readability.
Trailing comments must be shifted far enough to separate them from statements.
Commenting your code is great but avoid commenting obvious code.
Comments should not be long and overcomplicated.

5 Naming

5.1 Variable declaration

Variable names are nouns and should provide a clear meaning of what the object is.

5.2 Classes, Methods and Interface Declarations

Class names are written in UpperCamelCase.
Start Setter methods by set, followed by Variable to set.
Start Getter methods by get, followed by variable to get

```
public String getConsultant_Name() {
    return this.consultant_Name;
}

public void setConsultant_Name(String consultant_Name) {
    this.consultant_Name = consultant_Name;
}
```

Class names uses UpperCamelCase

```
public class Consultant
```

6 Statements

6.1 Simple Statements

Each line should contain at most one statement.

6.2 Compound Statements

Use Braces on all control statements inside the compound statement to reduce errors of forgetting to close braces.
Enclosed statements should be indented one more level than the compound statement for readability.

6.3 Return Statements

If the return value is an expression use parentheses to provide clear meaning and precedence.

6.4 If, If else Statements

If statements should always use , avoid the error-prone form:
if(condition) statement //AVOID.
if(condition) { statement } //USE.

6.5 For Statements

for statement should have the following form:

```
for (initialization; condition; update) {
    statements;
}
```

6.6 While loop

A while statement should have the following form:

while (condition) {

    statements;

}

6.7 Do-While Loop

A do-while statement should have the following form:

do {

    statements;

} while (condition);

6.8 try-catch Statements

A try-catch statement should be in the following form, the finally block can be omitted. try {

    statements;

} catch (ExceptionClass e) {

    statements;

} finally { statements;

}

7 Standards

7.1 Software Verification and Validation

7.2 Inspection

The issue of code inspection was addressed by using Eclipse IDE. As the backend is done in Java, Eclipse was chosen as it is the most popular java IDE (White,2014) and it automatically checks common errors. The documentation was written by different members of the team and inspected by our Documentation Leader, Tatenda Mafunga.

7.3 Walkthrough

Manually executes the product using simple test data. Usually the developer who produces the product leads the team to perform the walkthrough. The team checks the product step by step while asking questions. A product walkthrough was carried out to ensure that the any inconsistencies would be ironed out. Stephen Munro, Backend team, led the walkthrough of the backend and the remainder of the team asked questions to clarify any misunderstandings and suggest improvements. Johan van De Waal led the walkthrough of the front-end, which was carried out in a similar fashion.

7.4 Pair-Programming

The integration of the front-end and the backend was carried out applying pair programming principles to ensure that the implementation was of a high quality. The code was done on one laptop with a member from the backend team and the front-end team taking the roles of driver and navigator. The members switched roles as often as they found necessary.

In carrying out the software verification and validation, the following agile principles were followed: 1. Good is good enough 2. Keep it simple and easy to do 3. Value working software over comprehensive documentation

7.5 References

1. White, O. 2014. IDEs vs. Build Tools: How Eclipse, IntelliJ IDEA & NetBeans users work with Maven, Ant, SBT & Gradle. Online Available at: https://zeroturnaround.com/rebellabs/ides-vs-build-tools-how-eclipse-intellij-id #disqus_thread Accessed 05/05/18