sandwico

Coding standards: Real-time Fire Escape Route

by ERP

Mathilda Bresler
u16313382@tuks.co.za

Pieter Braak
u16313382@tuks.co.za

Kateryna Reva
u17035989@tuks.co.za

Jason Louw
u16313382@tuks.co.za

Xiao Jian Li
u16099860@tuks.co.za

12 June 2019

University Of Pretoria, Hatfield
Engineering, Built environment and Information Technology

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1 Purpose

The goal of these guidelines is to create uniform coding habits among the development team memebers so that reading, checking, and maintaining code written by different persons becomes easier. The intent of these standards is to define a natural style and consistency, yet leave to the authors of the software project source code, the freedom to practice their craft without unnecessary burden. General coding standards pertain to how the developer writes code. Adhering to these standards improve uniform style, clarity, flexibility, reliability and efficiency of our code.

2 Scope

This document describes the general software standards for code that is written for the *RTFE fire escape routes* project developed by Sandwico development team through the University of Pretoria for ERP a section of Epi Use. This includes standards for the programming languages *Java, C#, HTML, CSS, Javascript*.

3 Coding standards

3.1 Structured Programming

Structured (or modular) programming techniques are used. GO TO statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain

3.2 Indentation

A consistent use of indentation makes code more readable and errors easier to detect. 3 spaces is recommended per indent, but the exact number of blanks per indentation quantum may vary with the language. Statements that affect a block of code (i.e. more than one line of code) must be separated from the block in a way that clearly indicates the code it affects. Use vertical alignment of operators and/or variables whenever this makes the meaning of the code clearer.

Indentation should be used to:

- Emphasize the body of a control statement such as a loop or a select statement
- Emphasize the body of a conditional statement
- Emphasize a new scope block

Consistent indentation of 4 spaces is used throughout the system.
Examples:

```
public double distanceToGoal(Node n)
    {
        double dist = 0;
        Node temp = n;
        int position = nodes.indexOf(n);
        while(position < nodes.size() - 1)
        {
            temp = nodes.get(position);
            dist += temp.distanceToNode(nodes.get(++position));
        }
        return dist;
    }
```

```
     private JSONObject sendToRTFE(JSONObject req) throws Exception {
            JSONObject Response = new JSONObject();
            try{
                switch ( (String)req.get("type")){
                    case "build":{
```

```
                    Response.put("msg",build(req));
                    break;
                }
                case "buildingData":{
                    Response.put("msg",BuildingToUnityString(Response));
                }

            }
            Response.put("status", true);
        }catch(Exception e){
            if(verbose) {
                System.out.println("CRITICAL - UNITY FAIL");
                System.out.println(e.getMessage());
                System.out.println(e.getStackTrace().toString());
            }
            Response.put("msg","Exception: "+e.getMessage());
            Response.put("status", false);
        }
        return Response;
    }
```

3.3 Commenting

Inline comments explaining the functioning of the subroutine or key aspects of the algorithm shall be frequently used.

Comments should clearly demonstrate the function of the code, not only to other developers but also to you. Often when writing larger programs it can be easy to lose track of what certain functions do, and it is easier to read a well written comment that it is to trawl through lines of code trying to remember what the function of the code you have already written.

Comments should also not be too long. If your code has followed a coding standard, it should not be too difficult for developers to understand your code. Long comments are often unnecessary and make your code look messy.

Doxygen ”a tool for writing software reference documentation. The documentation is written within code, and is thus relatively easy to keep up to date.” must be used within the code. This is then used to generate Javadocs which documents the functionality of the code.
Example

```
   /**
     * getFormData function
     * @brief Converts the provided payload into a format used by the rest of the system
     *
     * @param payload
     * @return a JSONObject used by the rest of the system to execute the requests
     * @date 28/05/2019
     */
 else { // GET or HEAD method

    if (fileRequested.endsWith("/")) {
        fileRequested += DEFAULT_FILE;
        }
    File file = new File(WEB_ROOT, fileRequested);
    int fileLength = (int) file.length();
    String content = getContentType(fileRequested);
    if (method.equals("GET")) { // GET method so we return content
    byte[] fileData = readFileData(file, fileLength);
    // send HTTP Headers
    out.println("HTTP/1.1 200 OK");
    out.println("Date: " + new Date());
    out.println("Content-type: " + content);
    out.println("Content-length: " + fileLength);
    out.println(); // blank line between headers and content, very important !
    out.flush(); // flush character output stream buffer
    dataOut.write(fileData, 0, fileLength);
    dataOut.flush();
    }
    if (verbose) {
        System.out.println("File " + fileRequested + " of type " + content + " returned");
    }
}
```

3.4 Classes, Subroutines, Functions, and Methods

Methods, subroutines, and functions are kept reasonably sized. Software modules and methods should not contain an excessively large number of lines of code. They should be written to perform one specific task. If they become too long, then the task being performed should be broken down into subtasks which can be handled by new routines or functions.

3.5 Naming conventions

3.5.1 Variable names

Variables shall have mnemonic or meaningful names that convey to a casual observer, the intent of its use. Variables shall be initialized prior to its first use to eliminate chances of garbage values. Variables shall start without a capital letter, and be either written in camelCase:

```
int usersContainedInBuilding = 42;
```

or seperated by a _ character:

```
int users_contained_in_building = 42;
```

3.5.2 Classes

class names shall be capitalized and be a descriptive noun. Example:

```
public class Stairs extends Room {


}
```

3.5.3 Functions and Methods

Names of functions and methods shall have verbs in them describing it's function.

```
public static JSONObject handleRequest(JSONObject request)throws Exception
{

}
```

or seperated by a _ character:

```
 public int get_num_floors()
{

}
```

3.5.4 Files

Naming of files should allow for searching and navigation across files.

3.6 Use of Braces

In some languages, braces are used to delimit the bodies of conditional statements, control constructs, and blocks of scope. Our development team shall use either of the following bracing styles consistently: 1)

```
catch (Exception e){
//                System.out
}
```

2)

```
catch (Exception e)
{
//                System.out
}
```

Braces shall be used even when there is only one statement in the control block to improve readability and promote consistency.

3.7 Program Statements

Statements shall be limited to be one per line, and nested statements shall be avoided wherever possible.

3.8 Meaningful Error Messages

Error handling is very important in the system. Making error messages meaningful makes it easier to identify what errors have occurred due to which circumstances. These messages should also be displayed in ways that make it easy for review.

When possible, they should indicate what the problem is, where the problem occurred, and when the problem occurred. A useful Java exception handling feature is the option to show a stack trace, which shows the sequence of method calls which led up to the exception

3.9 Design patterns

Design patters are used throughout the system to improve code maintainability and modularization. This allows for addition of features without changing the entire structure of the system.

3.10 Variable declarations

Each variable definition will be limited to one per line, and each variable will be proceeded by a type.

4 File structure

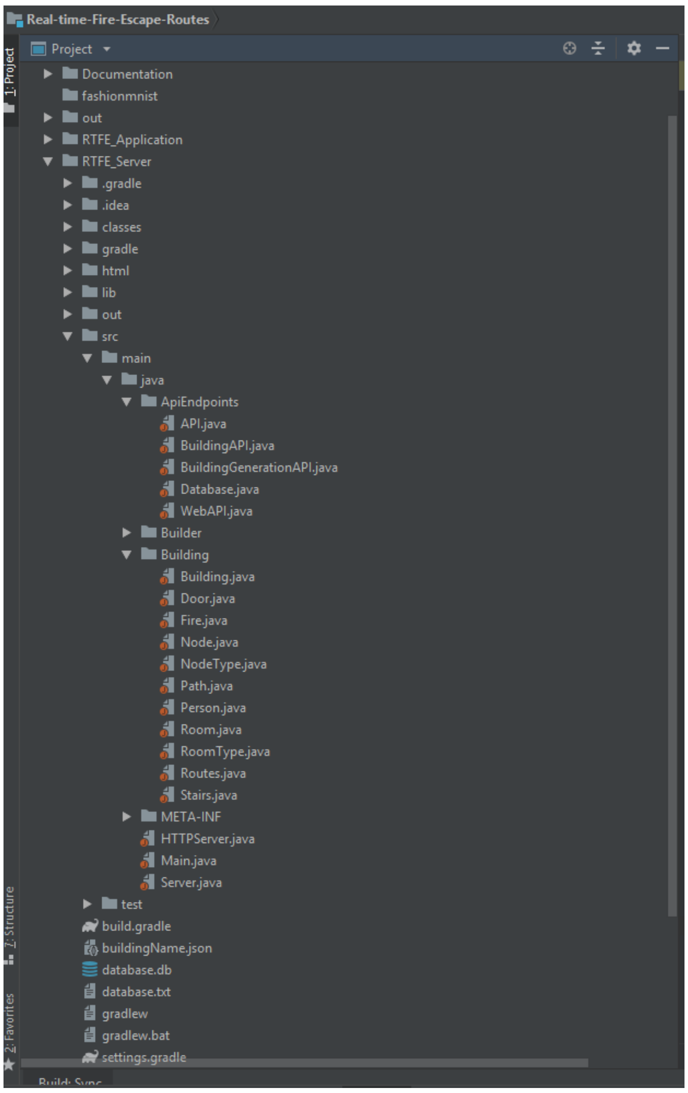
The files are stored in a hierarichal manner. This means that Items organized in folders and subfolders. The source files will be sorted into directories with regards to functional dependency. These directories will in turn will be placed in directories accordance with the subsystems they belong to.
Source files are given meaningful names, and all linked subroutines have similar functions that are logically linked and structured.
This is used since:

- Familiar & widely used
- Good at representing the structure of information
- Similar items are stored together
- Subfolders can function as task lists

Guidelines when constructing or modifying the file structure:

- Avoid overlapping categories
- Don't let your folders get too big
- Don't let your structure get too deep
- Data duplication is not permitted

Example:



```
▶ Documentation
  fashionmnist
▶ out
▶ RTFE_Application
▼ RTFE_Server
    ▶ .gradle
    ▶ .idea
    ▶ classes
    ▶ gradle
    ▶ html
    ▶ lib
    ▶ out
    ▼ src
        ▼ main
            ▼ java
                ▼ ApiEndpoints
                      API.java
                      BuildingAPI.java
                      BuildingGenerationAPI.java
                      Database.java
                      WebAPI.java
                ▶ Builder
                ▼ Building
                      Building.java
                      Door.java
                      Fire.java
                      Node.java
                      NodeType.java
                      Path.java
                      Person.java
                      Room.java
                      RoomType.java
                      Routes.java
                      Stairs.java
                ▶ META-INF
                  HTTPServer.java
                  Main.java
                  Server.java
        ▶ test
      build.gradle
      buildingName.json
      database.db
      database.txt
      gradlew
      gradlew.bat
      settings.gradle
```