Transport4You1
Coding Policy

Version 1.00

Revision History

Date	Version	Description	Author
2010-09-26	1.00	First Version	Dino Bartošak

Doc. No.:

Table of Contents

1. Introduction

1.1 Purpose of this document

The purpose of this document is to provide a coding guide for our project team so we can all use same coding conventions. It is very important for the team to use unique coding conventions for many reasons. For example, all of team members can read code from each other and understand what's going on. Or if a new member comes into team he could adapt very quickly and start working efficient. Good thing in our project is that we will use only one language (java) so there is no need for multiplying the same content but for different language. Each team member should read this document in detail so when he starts programming he could use described conventions. Also each member should have this document opened when he is coding so he could adapt easily into these conventions.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction,* describes contents of this guide and main purpose of this document.
- Section 2, *Programming Guide*, some useful advices for good programming.
- Section 3, *Java Coding Standard*, java coding conventions

1.3 Intended Audience

The intended audience is:

- All team members of Transport4You1 project.
- Supervisor of the Transport4You1 project.

1.4 Scope

The scope of this document is adjusted to this project. In this document are only coding conventions that we will use in our project. It includes standards for java programming language such as names (classes, variables, methods, interfaces), indentations, package names, language of programming (English). There are examples that covers all of them.

1.5 References

[1] Java Coding Standard: http://java.sun.com/docs/codeconv/CodeConventions.pdf
[2] Eight Secrets of Successful Programmers: http://www.merrioncomputing.com/Programming/7Secrets.htm
[3] Java Programming Style Guidelines: http://geosoft.no/development/javastyle.html

2. Programming Guide (eight secrets of successful programming)

These eight secrets are basic rules of good programming [2].

2.1 Code for human consumption

This means that we should be coding for clarity first, over efficiency and speed second. Computers work with low-level binary code that is not intuitive for humans and that is the main reason why high languages were developed - to help the programmer.

2.2 Comment often and comment well

The purpose of the commenting is to tell you (and any team member or future developer) what the program is intended to do. Good comment is such that tells you what particular class/method is doing not the way how it is doing.
Good comment: Calculate the ticket cost for passenger*; Bad comment:* cost += getTicketCost(passenger)

2.3 Layout code to increase legibility

Code indentation is very important for readability of code. Any code branch (an IF…THEN...ELSE construction) and any code repetition (a WHILE...END WHILE construction) should be indented so that it is easy to see where they start and end and what their purpose is.

2.4 Expect the unexpected and deal with it

Before you open a file, make sure that the file is present. Before you set focus to a control, make sure that the control is visible and enabled. Before doing something with an object, make sure that object exists. Try to work out what conditions could cause your code to fail and test for them before they cause the program to fall over, and do not rely on error handling to.

2.5 Name your variables to aid readability

There are a number of ways of variable/class naming. The key is to be consistent and to be as informative as possible. If you name a variable ticketCost, you give the programmer extra information as to what that variable is expected to contain (cost of particular ticket, not just any cost).

2.6 Keep your functions and subroutines simple

A function or subroutine should ideally only do one thing. One of the greatest sources of misunderstanding is a subroutine that does a number of different operations. This should be split into separate functions for each different thing it is doing so that these in turn are easy to reuse, and the scope of a code change is easy to understand. Good way of looking in that is that each function should not be longer of your screen size (this is in general of course).

2.7 Scope functions and variables appropriately

Functions and variables that are only used in one module should not be visible outside that module. Variables that are only used in a function or subroutine should not be visible outside that function or subroutine. This prevents any use of a variable or function outside of where it makes sense to use. This also reduces debugging time.

2.8 Never stop listening and learning

Keep an eye out for new opinions and methodologies and to evaluate them in an impartial and rational way.

3. Java Coding Standard

3.1 Naming conventions

Each name should be as much as simplified so that can be easy understandable to programmer.

3.1.1 Package names

Names representing packages should be in all lower case.

Transport4Yout1 convention:

```
dsd.transport4you.*
```

3.1.2 Class names

Names representing types must be nouns and written in mixed case starting with upper case.

Transport4Yout1 convention:

```
class Passenger{
}

class RouteSystem{
}
```

3.1.3 Interface names

Names representing interfaces are prefixed with capital “I”.

Transport4Yout1 convention:

```
Interface IPassenger   {
}

interface IRouteSystem{
}
```

3.1.4 Variable names

Variable names must be in mixed case starting with lower case. This makes variables easy to distinguish from types, and effectively resolves potential naming collision as in the declaration `Passenger passenger;`
So called “descriptive camel case”.

Transport4Yout1 convention:

```
passenger, ticketCost, myRoutes etc...
```

3.1.5 Constant names

Names representing constants (final variables) must be all uppercase using underscore to separate words. In general, the use of such constants should be minimized. In many cases implementing the value as a method is a better choice:

```
int getMaxIterations() // NOT: MAX_ITERATIONS = 25
{
  return 25;
}
```

Transport4Yout1 convention:

```
MAX_ROUTE, TICKET_VALUE
```

3.1.6 Method names

Names representing methods must be verbs and written in mixed case starting with lower case, „descriptive camel case".

Transport4Yout1 convention:

```
getPassenger(), computeTotalCost()
```

3.1.7 Abbreviation and acronym names

Abbreviations and acronyms should not be uppercase when used as name.

Transport4Yout1 convention:

```
exportHtmlSource(); // NOT: exportHTMLSource();
openDvdPlayer();    // NOT: openDVDPlayer();
```

3.1.8 Language of programming

English is the preferred language for international development.

Transport4Yout1 convention:

All names should be written in English.

3.1.9 Variable names regarding their scope

Variables with a large scope should have long names, variables with a small scope can have short names. Scratch variables used for temporary storage or indices are best kept short. A programmer reading such variables should be able to assume that its value is not used outside a few lines of code. Common scratch variables for integers are *i*, *j*, *k*, *m*, *n.*

Transport4Yout1 convention:

Scratch variables for integers: *i*, *j*, *k*, *m*, *n*..

3.1.10 Implicit name of the object

The name of the object is implicit, and should be avoided in a method name

Transport4Yout1 convention:

```
route.getLength();   // NOT: route.getRouteLength();
```

3.1.11 Terms get/set

The terms *get/set* must be used where an attribute is accessed directly.

Transport4Yout1 convention:

```
passenger.getName();
passenger.setName(name);

matrix.getElement(2, 4);
matrix.setElement(2, 4, value);
```

3.1.12 Using "is" prefix for Boolean variables and methods

Using the *is* prefix solves a common problem of choosing bad boolean names like `status` or `flag`. `isStatus` or `isFlag` simply doesn't fit, and the programmer is forced to chose more meaningful names.

Setter methods for boolean variables must have `set` prefix as in:

```
void setFound(boolean isFound);
```

There are a few alternatives to the `is` prefix that fits better in some situations. These are `has`, `can` and `should` prefixes:

```
boolean hasLicense();
boolean canEvaluate();
boolean shouldAbort = false;
```

Transport4Yout1 convention:

```
isSet, isVisible, isFinished, isFound, isOpen

boolean hasLicense();
boolean canEvaluate();
boolean shouldAbort = false;
```

3.1.13 Plural form of names should be used on names representing a collection of objects.

Plural form should be used on names representing a collection of objects. This enhances readability since the name gives the user an immediate clue of the type of the variable and the operations that can be performed on its elements.

Transport4Yout1 convention:

```
Collection<Point>  points;
```

```
int[]               values;
```

3.1.14 Using of n *prefix*

n prefix should be used for variables representing a number of objects.

Transport4Yout1 convention:

```
nPoints, nLines
```

3.1.15 Exception class names

Exception classes should be suffixed with *Exception*.

Transport4Yout1 convention:

```
class AccessException extends Exception{
  :
}
```

3.2 Statements

3.2.1 Loop variables

Loop variables should be initialized immediately before the loop.

Transport4Yout1 convention:

```
isDone = false;           // NOT: bool isDone = false;
while (!isDone) {         //      :
  :                       //      while (!isDone) {
}                         //        :
                          //      }
```

3.3 Layout

3.3.1 Basic indentation

Indentation should be TAB

Transport4Yout1 convention:

```
for (i = 0; i < nElements; i++){
  a[i] = 0;
}
```

3.3.2 Block layout

Transport4Yout1 convention:

```
while (!done) {
```

```
  doSomething();
  done = moreToDo();
}

//NOT LIKE THIS:
while (!done)
  {
    doSomething();
    done = moreToDo();
  }
```

3.3.3 Method definitions

Transport4Yout1 convention:

```
public void someMethod() throws SomeException {
  ...
}
```

3.3.4 If-else block

Transport4Yout1 convention:

```
if (condition) {
  statements;
}

if (condition) {
  statements;
}
else {
   statements;
}

if (condition) {
  statements;
}
else if (condition) {
  statements;
}
else {
  statements;
}
```

3.3.5 for *statement*

Transport4Yout1 convention:

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

3.3.6 while *statement*

Transport4Yout1 convention:

```
while (condition) {
  statements;
}
```

3.3.7 do-while *statement*

Transport4Yout1 convention:

```
do {
  statements;
} while (condition);
```

3.3.8 try-catch *statement*

Transport4Yout1 convention:

```
try {
  statements;
}
catch (Exception exception) {
  statements;
}

try {
  statements;
}
catch (Exception exception) {
  statements;
}
finally {
  statements;
}
```

3.3.9 Single statement if-else, for *or* while *statements*

Always use brackets.

Transport4Yout1 convention:

```
if (condition){
  statement;
}

while (condition){
  statement;
}

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

3.4 Comments

3.4.1 Comments language

Transport4Yout1 convention:

All comments should be written in English.

3.4.2 Javadoc comments

A readable form is important because this type of documentation is typically read more often inside the code than it is as processed text.

Transport4Yout1 convention:

```
/**
 * Return lateral location of the specified position.
 * If the position is unset, NaN is returned.
 *
 * @param x    X coordinate of position.
 * @param y    Y coordinate of position.
 * @param zone Zone of position.
 * @return     Lateral location.
 * @throws IllegalArgumentException  If zone is <= 0.
 */
public double computeLocation(double x, double y, int zone) throws
IllegalArgumentException{
  ...
}
```

Javadoc of class members can be specified on a single line as follows:

```
  /** Number of connections to this database */
  private int nConnections_;
```

3.4.3 Space after comment identifier

Improves readability by making the text stand out.

Transport4Yout1 convention:

```
// This is a comment    NOT: //This is a comment

/**                     NOT: /**
 * This is a javadoc           *This is a javadoc
 * comment                     *comment
 */                           */
```

3.4.4 Non-JavaDoc comments, including multi-line comments

Use *//* for all non-JavaDoc comments, including multi-line comments.

Transport4Yout1 convention:

```
// Comment spanning
// more than one line.
```

3.4.5 Indentation of comments

Comments should be indented relative to their position in the code

Transport4Yout1 convention:

```
while (true) {        // NOT:  while (true) {
  // Do something               // Do something
  something();                    something();
}                               }
```