Taxi Service Coding Policy

Version 1.2

Revision History

Date	Version	Description	Author
2012-10-31	1.0	Initial version	Karlo Zanki
2012-11-18	1.1	Added a section relative to Java/Android	Fabio Kruger
2013-01-02	1.2	Final version	Fabio Kruger

TABLE OF CONTENTS

1. Introduction

1.1 Purpose of this document

The goal of this document is to provide a coding convention for all developers working on the project. The team working on this project consists of 8 students, each used to develop the code in his own way, name objects in different style and arrange lines as it suits him. To avoid the code to become messy and hard to understand for different members of the team, we need to define some conventions we all are going to respect. This will reduce the risk of misunderstanding and speed up the development process.

1.2 Document organization

The document is organized as follows:

- Section 1: Introduction, general description of the contents and main purpose of this document.
- Section 2: Programming good practices, useful advices for good programming.
- Section 3: Coding conventions, coding conventions that are going to be used on this project.

1.3 Intended Audience

The intended audience is:

- Members of the taxi service team.
- DSD course supervisors

1.4 Scope

This document will be used during the work on the Taxi Service project but it can also serve as a reference to some future projects.

1.5 References

[1] Best practices in programming: http://net.tutsplus.com/tutorials/html-css-techniques/top-15-best-practices-for-writing-super-readable-code/
[2] XML Documentation: http://msdn.microsoft.com/en-us/magazine/cc302121.aspx
[3] C# coding conventions: http://msdn.microsoft.com/en-us/library/vstudio/ff926074.aspx

2. Programming good practices

2.1 Self-explainable code

Try to write the code so it is understandable by itself. You can achieve this with smart naming of the methods and variables. In places where code is not understandable by itself write comments to describe it better.

2.2 Separation of functions

Each element of the code (variable, method, class) should be responsible for only one thing. Don't complex methods. Simple methods are easier to understand and change if needed. This way you reduce coupling and increase cohesion.

2.3 Programming to interface

-Whenever you have interaction between two classes that is not trivial to understand, and is probably going to be changed during time, it is good to extract interfaces. Extracting interfaces looses the coupling and enables you to change one without affecting the other and also to test one class without other one being even implemented.

2.4 Defensive programming

During the development you need to expect that anything that can go wrong will go wrong. You need to predict all possible inputs into your program and make sure they don't cause failures in your system. User will always give wrong inputs, make your code robust.

2.5 Hard-coded values

No hard-coded values except 0 and 1 are allowed in the program. It is better to create and use constants, so that the changes are centralized.

2.6 Refactoring

Whenever you make important changes in your code refactor it. This way you make your code easy to maintain and avoid spaghetti code which is fragile and immovable.

2.7 Duplicating the code

DON'T duplicate the code. Extract it to a method and call the method instead.

3. Coding conventions

In this project we use mainly two programming languages: Java and .Net. Java will be used in the client applications, both for the customer and the taxi driver. The applications are written for Android devices, therefore the conventions will be extended to take in account also the best practice for Android devices. The server component is written in .Net.

3.1 Java / Android

3.1.1 File suffixes:

Java utilizes the following suffixes:

- .java: for source code files
- .class: for Java byte code files.

In addition, Android stores a great amount of configuration values, properties and application specifications are defined in XML files.

3.1.2 Source file format:

Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.

Java source files have the following ordering:

- Beginning comments: All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice.
- Package and Import statements:
- Class and interface declarations: they are composed of the following elements in sequence:
  - Class/interface documentation comment
  - Class/interface statement
  - Class/interface implementation comment
  - Class static variables
  - Instance variables
  - Constructors
  - Methods

3.1.3 Naming conventions:

- Package: The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.
- Classes: Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).
- Interfaces: Interface names should be capitalized like class names.
- Methods: Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
- Variables: Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
- Constants: The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging).

3.2 .Net

3.2.1 Naming conventions

Each name in the program (variable, class, method) should be simple and understandable to any person, not just to programmers. None of the names should contain more than 30 characters and should be written in PascalCase.

3.2.2 Public classes, methods, fields and properties

Names of the public classes, methods, fields and properties Names should start with upper case.

```
public class TaxiDriver {
}
```

3.2.3 Private, protected classes, methods, fields and properties

Names of the public classes, methods, fields and properties Names should start with lower case.

```
private class cityZone {
}
```

3.2.4 Interfaces

Names of the interfaces should start with capital I.

```
interface ITaxiDrive {
}
```

3.2.5 Boolean variables and methods

Boolean variables and methods should start with prefix *is/has*

```
taxi.isAssignedToADrive();
```

3.2.6 Constants

The name of constants should be formed of uppercase letters. Underscores can be used to separate words.

```
public const int SERVER_PORT = 1200;
```

3.2.7 Language

English will be used for names of the program elements.

3.2.8 Database tables

Database tables names should be in singular with upper case and shouldn't contain any prefix or sufix.

3.2.9 Comments

3.2.10 Comments language

All comments should be in English.

3.2.11 XML comments

Use comment documentation because it will make other developers easier to understand what your methods do and how should they be used. It also allows you to automatically create XML documentation.