M.E.N.A.C.E.

Motorized Embedded Network-connected smArt Car Entity

DIT524 - PROJECT: SYSTEMS DEVELOPMENT

Date 2017-02-23

Isak Magnusson
Kosara Golemshinska;
Laiz H. B. de Figueroa;
Melinda Ivók;
Nina Uljanić;
Rema Salman;
Syeda Elham Shahed.

CODE STANDARDS

CLASS HEADER AND DECLARATION:

When a file is created, the heading below must be created together with the creator's name, both in Java and C++ classes. When someone edits the file, an editor should be added and the changes made should be listed. If the editor is the author, a list of the changes must be provided after the author name.

```
/**
 * Class Explanation
 *
 * @author
 * @editor - brief list of changes
 */
```

COMMENTS AND COMMENTS' STYLES

All classes must contain comments in order to provide guidance for other developers throughout the development process. The style below must be followed.

The code will contain three different kinds of comments:

- Single line comments: when adding a description or some kind of explanation on the same line of code or the line above; style: " // ".
- Block comments: this style will be used for internal and longer explanation comments; style: " /* … */ ".
- JavaDoc comments should be displayed before methods, constructors and when providing classes' descriptions and interfaces.

M.E.N.A.C.E.

Motorized Embedded Network-connected smArt Car Entity

NAMING

All names given by the team, in Java and C++ classes, should be descriptive.

- Classes: CamelCase is to be used.
- Methods: pascalCase is to be used. The first word of the method should be a verb.
- Variables: pascalCase is to be used. The names should be short but still descriptive.
- Constants: Every character should be capitalized. The words should be separated by underscores.

PACKAGE CREATION AND ORGANIZATION

Different packages should be created, based on the function they describe (i.e., libraries, UI, multimedia) and each Java class should be saved into corresponding package.

PACKAGE AND IMPORT STATEMENTS

The first non-comment line in every class should be the package statement to clearly note which package the given class belongs to.

In case of an import within the given class, the import statement should be the second line of code, following the package statement.

To avoid conflicts, the team should not use wildcard imports.

LINE LENGTH AND WRAPPING STYLE

The code developed should not have lines longer than 80 characters. If that is the case, the expression should be broken down. Breaking the expression should be done by following these rules:

- Break after a comma;
- Break before an operator;
- Align the new line with the beginning of the expression;
- In case of confusing code created by following the above rules, every new line should be indented for 8 spaces in addition to the previous indentation of the method

VISIBILITY

To protect data, the team should make the variables private and the methods public.

BLANK LINES

In all the classes created, Java and C++, one blank line should always be used in the following circumstances:

- Between class and interface definitions.
- Between methods.
- Before a block or single line comment.
- Between logical sections inside a method to improve readability.

BLANK SPACES AND TAB

A blank spaces should be used by the team in the following circumstances:

- Should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.
- New line after the method header.
- Should appear after commas in argument lists.
- All binary operators should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. Example:

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
n++;
}
printSize("size is " + foo + "\n");
```

- The expressions in a *for* statement should be separated by blank spaces. Example:

```
for (expr1; expr2; expr3)
```

- Casts should be followed by a blank space. Example:

```
myMethod((int)(cp + 5),  ((int)(i + 3))  + 1);
```

Tabs should be used for indentation rather than blank spaces (1 tab = 4 spaces).

METHODS REUSABILITY

The developers should avoid creating multiple methods for the same function. Existing methods should be used instead.