PANDA INC

Coding Standards

Description

Short document defining the coding standards for Panda Inc.

Quinton Swanepoel
Keaton Pennels
Azhar Patel
Tshepo Malesela

The following coding standards have been agreed upon amongst all members of Panda Inc. These standards should be applied to the code produced by every member of the team to ensure consistency and readability when integrated.

1. Naming Conventions

1.1 Variables

- Variables should be named to best describe their purpose.
- Variable should be named using camel case.
- **Example:**

```
int userCount;
boolean isPlatinumMember;
double averageTime;
```

1.2 Functions

- Functions should be named using verbs that best describe their purpose.
- Functions should always start with a capital letter.
- If a function name contains more than one word then each word should start with a capital letter. The class name should not contain any special characters.
- **Example:**

```
public void PopulateList(){

}
```

1.3 Classes

- Classes should be named using nouns that best describe their purpose.
- Classes should always start with a capital letter.
- If a class name contains more than one word then each word should start with a capital letter. The class name should not contain any special characters.
- **Example:**

```
class Members{

}
```

2. Commenting Conventions

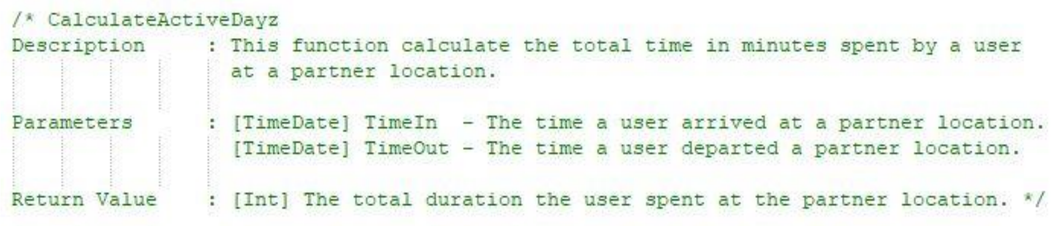
2.1 Variables

- Variable comments should only be left if the variables purpose is not immediately evident.
- All variable comments should be tabulated in the same column.
- Variable comments should be brief.
- **Example:**

```
int memberCount;
double averageUserRateOfChange;        //Average rate at which the user changes policies.
string currentMemberName;
boolean isPlatinumMember;              //True if current member is a platinum account holder
```

2.2 Functions

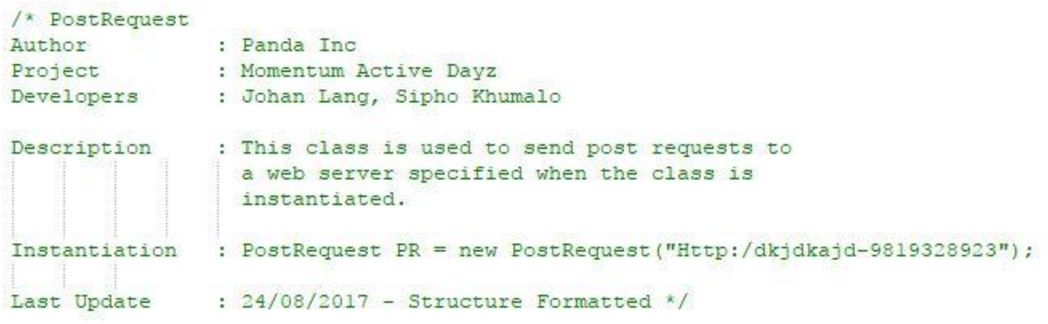
- Every function should contain a comment preceding its implementation containing the following details
    - Author of the code (Company/Organisation)
    - Projects name that the class belongs to.
    - Comma separated list of names and surnames of the developers who worked on the class.
    - A description of the purpose of the class.
    - The code used to instantiate the class.
    - The date of the last update/revision of the class and a short description of the revision that was made (separated by a dash).
- **Example:**



```
/* CalculateActiveDayz
Description      : This function calculate the total time in minutes spent by a user
                   at a partner location.

Parameters       : [TimeDate] TimeIn  - The time a user arrived at a partner location.
                   [TimeDate] TimeOut - The time a user departed a partner location.

Return Value     : [Int] The total duration the user spent at the partner location. */
```

2.3 Classes

- Every class should contain a comment preceding its implementation containing the following details
    - Author of the code (Company/Organisation)
    - Projects name that the class belongs to.
    - Comma separated list of names and surnames of the developers who worked on the class.
    - A description of the purpose of the class.
    - The code used to instantiate the class.
    - The date of the last update/revision of the class and a short description of the revision that was made (separated by a dash).
- **Example:**

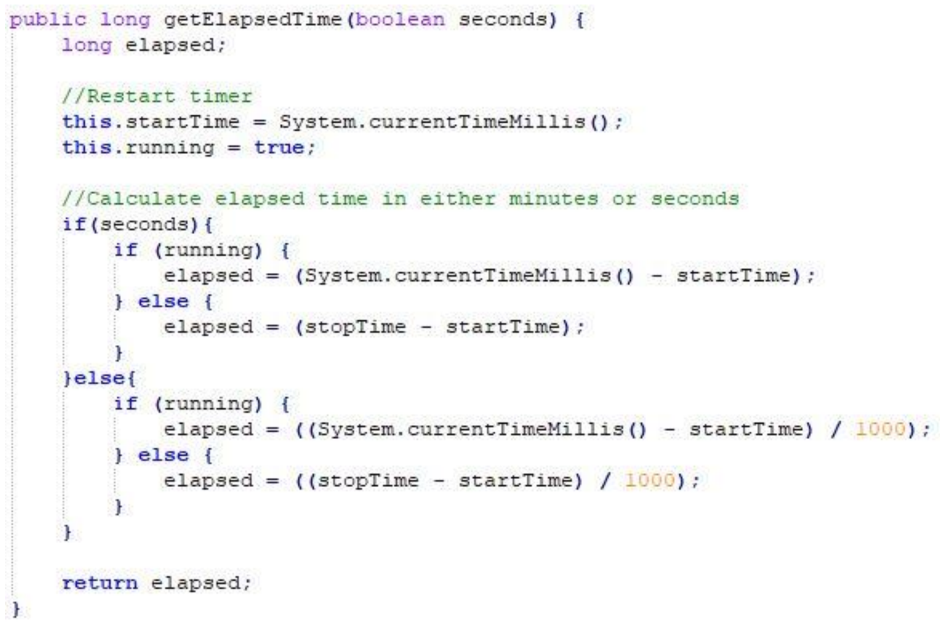


```
/* PostRequest
Author           : Panda Inc
Project          : Momentum Active Dayz
Developers       : Johan Lang, Sipho Khumalo

Description      : This class is used to send post requests to
                   a web server specified when the class is
                   instantiated.

Instantiation    : PostRequest PR = new PostRequest("Http:/dkjdkajd-9819328923");

Last Update      : 24/08/2017 - Structure Formatted */
```

3. Structuring Conventions

- All variables should be declared at the beginning of the function or class.
- All functions/classes/programming structures should contain the opening bracket after its name/application and closing bracket on a separate line at the end of its implementation.
- All coding should be tabulated in a neat format by nesting code with a single tab where necessary.

- No unnecessary empty lines should be added unless separating sections of code for readability reasons. In this case a comment must be left to before each section to briefly state the following section of code’s purpose.
- An empty line should precede a return statement.
- An empty line should be placed after variable declarations to separate the variable declarations from the rest of the functions/classes code.
- **Example:**



```java
public long getElapsedTime(boolean seconds) {
    long elapsed;

    //Restart timer
    this.startTime = System.currentTimeMillis();
    this.running = true;

    //Calculate elapsed time in either minutes or seconds
    if(seconds){
        if (running) {
            elapsed = (System.currentTimeMillis() - startTime);
        } else {
            elapsed = (stopTime - startTime);
        }
    }else{
        if (running) {
            elapsed = ((System.currentTimeMillis() - startTime) / 1000);
        } else {
            elapsed = ((stopTime - startTime) / 1000);
        }
    }

    return elapsed;
}
```
