Java Coding Standards

Alexander John D. Jose

ADVANSE S17

I. Classes

1. Only one class per source file.
2. Classes must have no wildcard (*) import. All imported packages must be precisely what you need, i.e, `import java.util.ArrayList;`
3. Classes must always have a constructor, even if it is left empty.
4. Global variables must always be `private` or `protected`. It cannot be declared as `public`. Proper getters and setters function must be constructed to access these private variables.
5. Global variables that have to be initialized will have to do so inside the constructor.

```
private String name;

public Student()
{
  name = new String();
}
```

6. All unused variables must be removed.
7. Extended class must always have the implementation of the super class's abstract function on top of non-abstract functions.

II. Code Formatting

1. Only one code in a single line.

```
this.attack = 40;
this.defense = 60;
this.type = "Water";
```

2. Braces must always be inserted on a new line.

3. There must be exactly two spaces after the braces.

```
public int getCurrentHP()
{
  return healthPoints;
}
```

4. Spaces must always be placed in between equations.

```
if (powerPoints - amount > -1)
{
  newPP = powerPoints - amount;
}
```

5. Function and variables names must be in camel case.

III. Functions

1. Functions that have a void data type must not return anything.
2. `private` functions must be on the lower part of the Class, followed by `protected` functions.
3. `abstract` functions must be below on top of non-abstract `public` functions, regardless of its access type.

```
public abstract void attackEnemy(Pokemon enemy);

public String getName()
{
  return name;
}
```

4. `public` functions must be placed below `abstract` functions. Or in other terms, `public` functions must be place in between the `abstract` functions and `private` functions.
5. `@Override` is used for functions that must be overridden such as `abstract` functions

IV. Conditional Statements

1. Conditional statements must always have braces even if the content inside is a single code.

```
if (powerPoints - amount > -1)
{
  newPP = powerPoints - amount;
}
```

2. `switch` statements must always have a `default` category.
3. Conditional statements that uses `boolean` should avoid the comparison `if (isDead == true)` rather, use `if (isDead).`

V. Loops

1. For the `for` loop, counters that will not be used later on can be declared inside the `for` loop. For example:

```
for (int i = 0; i < array.size(); i++)
{
  sum = i + sum;
}
```

   The code above would not use the variable `i` after the loop so it is best declared inside the `for` loop.

2. Contrary to item 1, variables that will be used after the `for` loop must be declared outside.
3. Avoid nested loops at all cost. Limit all nested loops to second degree.
4. Braces must always be used even though the loop contains a single statement.

VI. Arrays

1. Arrays must be initialized in one line. For example;
   `int[] numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};`
2. There must be no C style array declaration.
3. The use of ArrayList versus arrays is greatly encouraged.

VII. Source Control/Online Repository

1. Always get the latest version of the project first before pushing in the source control.

2. Always resolve conflict/s first before editing or checking in the source control.
3. If a code review is required, don't push it into the main branch. Push it into a shelveset first. (Visual Studio thing, I am not aware if it there is a 'sheveset' in GitHub or other repository).
4. Remove unnecessary comments on the files that needs to be pushed in the source control. These may include codes to print a variable in the console, commented non-working codes, etc.
5. Don't push every files you have edited such local configurations for the database. This might break other developers' configuration for their databases.

VIII. Tools

1. Eclipse Mars.
2. Java SDK 8.