Naming Conventions:

Variables :- **lowerCamelCase**

1. Use full English descriptions for names. Avoid using abbreviations.

```
don't  - fName, lName, and mi.
do     -  firstName, lastName, and middleInitial
```

2. Avoid overly long names (greater than 15 characters).

```
don't -  setTheLengthField
do    -  should be shortened to setLength
```

3. Avoid names that are very similar or differ only in case.

```
don't -  product, products, and Products in the same program
```

4. Giving meaningful names is a way of self-documenting the code.

```
don't - double tax1;   // sales tax rate
        double tax2;   // income tax rate
do    - double salesTaxRate;  // no comments required due to
        double incomeTaxRate; // self-documenting variable names
```

5. When appropriate, give meaningful names for your loop iterators rather than i, j.
   (cases when iterator isn't meant as array index)

Class :- **UpperCamelCase**. *Follow other general variable naming conventions.*

Constants :- **ALL_UPPER_CASE**

1. Separate words with the underscore

```
don't  -  taxRate , TAXRATE.
do     -  TAX_RATE
```

2. Avoid using unnamed constants(magic numbers) for any number other than 0 and 1.

```
don't  -  day = (3 + numberOfDays) % 7; //NO! uses magic no.s
do     -  final int WEDNESDAY = 3;
          final int DAYS_IN_WEEK = 7;
          day = (WEDNESDAY + numberOfDays) % DAYS_IN_WEEK;
```

Methods:

1. Begin method names with a **strong action verb**
   (eg: `deposit`).
   If the verb is not descriptive enough by itself, include a noun
   (eg: `addInterest`).
   Add adjectives if necessary to clarify the noun
   (eg:`convertToEuroDollars`).

2. Use the prefixes `get` and `set` for ***getter*** and ***setter*** methods.

   eg: `getBalance` and `setBalance` to access or change the instance variable `balance`.

3. If the method **returns a boolean value**, use `is` or `has` as its name prefix

   eg: `isOverdrawn` or `hasCreditLeft` when they return true or false values.

4. Avoid the use of the word ***not*** in the boolean method name, use the `!` operator instead.

   eg: use `!isOverdrawn` instead of `isNotOverdrawn`.

Commenting

Ref: *http://www.docjar.net/html/api/java/util/Collections.java.html*

Comment the details of nontrivial or non obvious design decisions.

1. Add header comments providing identification information about the program and author.

   In eclipse , type in /** and hit Enter. Comment block will be created. Now

   follow the template below:

```
/**
*@description : This class provides...
```

*@author :

*@version/date :

*/

Class ABC { ... }

Add a description to every class you write just before the class definition.

2. Add single-line comments providing overviews or summaries of chunks of code.

eg:
```
// Compute the exam average score for the midterm exam
sumOfScores = 0;
for (int i = 0; i < scores.length; i++)
    sumOfScores = sumOfScores + scores[i];
average = float(sumOfScores) / scores.length;
```

3. Add trailing comments that provide information for one line of code if appropriate.

Always add comments while you are coding rather than waiting until the program is finished :)

Formatting

White Space

1. Use a white space between a keyword and parentheses

dont - while(a>b), for(...)
do - while (a > b)

2. Do not use white space between a function name and its parenthesis

dont - myFun (arg1,arg2)
do - myFun(arg1, arg2)

3. Use a white space before and after an operator.

don't - for(i=0;i<n;i++)
do - for (i = 0; i < n; i++)

Blank Lines

1. Use blank lines to separate logical groups of program statements. (usually proceeded with a single-line summary comment)

2. Use one blank line at the beginning of a function. Use two blank lines before the
start of each new method within a class.

Indentation

1. Use **four spaces** for indentation to indicate nesting of control structures.

Line Length

Avoid lines longer than **80 characters** When an expression will not fit on a single line of 80 characters, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Align the new line 4 spaces under it parent line.

```
do -            someMethod(longExpression1,      longExpression2,
longExpression3,
                longExpression4, longExpression5);
       longName1 = longName2 * (longName3 + longName4 - longName5)
                + 4 * longname6;
```

References :

http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

http://www.cwu.edu/~gellenbe/javastyle/index.html