Coding Conventions

We are following the coding conventions provided by the professor in his slides.

The link to the conventions is :
https://www.oracle.com/technetwork/java/javase/documentation/codeconventions-141855.html

We have summarized the conventions that we are planning to follow for the code. Please refer these conventions when naming any variable, file or class and also while putting braces for block structured code.

1. Java Source Files

- Each Java source file contains a single public class or interface.
- Java source files have the following ordering:

    - Beginning comments : for documentation, use Javadoc comments to describe the class and its functionality
    - Package and Import statements
    - Class and interface declarations

    1) Class/interface documentation comment
    2) class or interface statement
    3) Class (static) variables
    4) Instance variables
    5) Constructors
    6) Methods

2. Indentation

- Four spaces(1 tab) should be used as the unit of indentation.

3. Comments

- Two kinds of comments: implementation comments and documentation comments
- Comments should not be enclosed in large boxes drawn with asterisks or other characters.
  Comments should never include special characters such as form-feed and backspace.

4. Declarations

- Do not put different types on the same line.
    - EG int foo, fooarray[]; //WRONG!
- One declaration per line is recommended

    int level; // indentation level
    int size; // size of table

    is preferred over

    int level, size;
- Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.
- Put declarations only at the beginning of blocks. Don't wait to declare variables until their first use.
- Class and Interface Declarations :
- When coding Java classes and interfaces, the following formatting rules should be followed:
    - No space between a method name and the parenthesis "(" starting its parameter list
    - Open brace "{" appears at the end of the same line as the declaration statement
    - Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"

EG:

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
      ivar1 = i;
      ivar2 = j;
    }

   int emptyMethod() {}
}
```

- Methods are separated by a blank line

5 Declarations

- **Simple Statements**
  Each line should contain at most one statement. Example:

```
argv++;       // Correct
argc--;       // Correct
argv++; argc--; // AVOID!
```

- **Compound Statements**
  Compound statements are statements that contain lists of statements enclosed in braces "{ statements }".

- **return statements**
  A return statement with a value should not use parentheses unless they make the return value more obvious in some way

- **if, if-else, if else-if else Statements**
  The if-else class of statements should have the following form:

```
if (condition) {
   statements;
}

if (condition) {
   statements;
} else {
   statements;
}

if (condition) {
   statements;
} else if (condition) {
   statements;
} else {
   statements;
}
```

◦ If more than 2 if-else blocks, try to use Switch case for simplicity.

- **for statements**
  A for statement should have the following form:

```
for (initialization; condition; update) {
   statements;
}
```

- **while statements**

  A while statement should have the following form:

```
while (condition) {
   statements;
}
```

- **do-while statements**

  A do-while statement should have the following form:

```
do {
   statements;
} while (condition);
```

- **switch statements**

  A switch statement should have the following form:

```
switch (condition) {
case ABC:
   statements;
   /* falls through */
case DEF:
   statements;
   break;
case XYZ:
   statements;
   break;
default:
   statements;
   break;
}
```

- **Try catch statements**

  A try statement should have the following form:

```
try {
   statements;
} catch (ExceptionClass e) {
   statements;
}
```

6 White Space

I. Blank Lines: One blank line should always be used in the following circumstances:

- Between methods
- Between the local variables in a method and its first statement
- Before a block or single-line comment
- Between logical sections inside a method to improve readability

II. Blank Spaces : Blank spaces should be used in the following circumstances:

- A blank space should appear **after commas in argument lists**.
- All binary operators except . 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++;
 }
```

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

```
for (expr1; expr2; expr3)
   myMethod((byte) aNum, (Object) x);
   myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

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

7 Naming Conventions

Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase meaningful names	package controllers; package models; package views;
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.	public class MapGenerator;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	public void allocateCountries();
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 arei, j, k, m, and n for integers; c, d, and e for characters.	int i; char c; float myWidth;

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_").	public static final int *CONTINENT_VALUE* = 7;

8 Programming Practices

Referring to Class Variables and Methods

Avoid using an object to access a class (static) variable or method. Use a class name instead. For example:

```
classMethod();          //OK
AClass.classMethod();   //OK
anObject.classMethod(); //AVOID!
```

Constants

Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

Variable Assignments

Avoid assigning several variables to the same value in a single statement. It is hard to read. Example:

```
fooBar.fChar = barFoo.lchar = 'c'; // AVOID!
```

Do not use the assignment operator in a place where it can be easily confused with the equality operator. Example:

```
if (c++ = d++) {       // AVOID! (Java disallows)
   ...
}
</blockquote>
```

should be written as

```
if ((c++ = d++) != 0) {
   ...
}
```

Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler. Example:

```
d = (a = b + c) + r;       // AVOID!
</blockquote>
```

should be written as

```
a = b + c;
d = a + r;
```

9. Miscellaneous Practices

Parentheses

- It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others-you shouldn't assume that other programmers know precedence as well as you do.

```
if (a == b && c == d)     // AVOID!
if ((a == b) && (c == d)) // RIGHT
```

Returning Values

- Try to make the structure of your program match the intent. Example:

```
if (
        booleanExpression) {
   return true;
} else {
   return false;
}
```

should instead be written as

```
return
     booleanExpression;
```

Similarly,

```
if (condition) {
   return x;
}
return y;
```

should be written as

```
return (condition ? x : y);
```

Expressions before `?' in the Conditional Operator

- If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized. Example:

```
(x >= 0) ? x : -x;
```