Coding Standards Summary

Coding Organisation

Packages

- uk.ac.aber.cs221.gp17
- All Lowercase Letters
- Structure in a hierarchical manner
- Each application should have its own package (1)
- Specific classes to the application should be placed in a sub-packaged called app

Javadoc Comments

- Maintained at all times
- List all packages and their purpose
- Package.html should exist for all packages

Identifier Naming Conventions

General

- Use US spelling (ie. Color, Favorite)
- Names should be self-documenting (indexVariable rather than i)
- Use real world object names for objects
- Use predicate clauses/adjectives for Booleans (ie heatingShouldBeOn)
- Use action verbs for procedures and entries (ie removeNode)

Classes and Interfaces

- Names use Upper Camel Case (ThisIsAClassName)
- For abbreviations, only first letter should be capital (GuiResources rather than GUIResources)

Methods and Variables

- Names are Lower Camel case (thisIsAVariableName)

• Read-Only methods	get<item>()
• Read-Write methods	set<item>(type value)
• Boolean Read-Only methods	is<item>()
• Boolean Read-Write methods	set<item>(boolean value);

Constants

- Names are Full Uppercase with underscores to separate words (CONSTANT NAME)

Class Organisation

File Structure

- All public classes/interfaces are defined in a file with the same name
- Every top-level class should be defined in its own file, regardless of modifier
- Exception : test classes which are not used outside the file

Class Structure

- Every class should have its variables/methods arranged into groups preceded by a comment
- Group related methods together
- See “Class Template” Below

Inner Classes

- Used to break up the complexity of a large class
- Should not be used outside is parent class unless considered an attribute of parent class

Anonymous Classes

- Only used to pass simple implementations of an interface as parameters

Comments

General

- Use “/** */” for Javadoc
- Use “//” for single-line comments
- Avoid using multiline comments unless its commenting out code

Files

- Each file should include
    - Simple Header giving the filename
    - Copyright message
    - Version
    - Date

```
/*
 * @(#) SomeClass.java 1.1 2021/12/15
 *
 * Copyright (c) 2021 Aberystwyth University.
 * All rights reserved.
 *
 */
```

Classes and Interfaces

- Each class/interface requires a Javadoc class header
    - Description providing an overview (Separated from tags by a new line
    - @author tag (Not inner classes)
    - @version tag (Not inner classes)
    - @see tag – (For cross-referencing
    - Anonymnous classes do not need headers

```
/**
* A class that generates new wibbles.
* This class generates new instances that implement the Wibble interface.
* The exact class that is returned depends on the current WibbleSystem
* that is active.
* <p>
* Static getFactory( ) method should be used to create new instances of
* WibbleFactory rather than the constructor, and new wibbles may be
* obtained through the createNewWibble( ) method.
*
* @author Alex McManus
* @author Richard Joseph
* @version 1.1 Initial development.
* @version 1.2 BN998: Now works with modified database structure.
* @see Wibble
* @see WibbleSystem
* @see #getFactory( )
* @see #createNewWibble( )
*/
public class WibbleFactory ...
```

Methods

- Each method requires a javadoc header:
    - Description should cover the purpose / side effects
    - @param tags
    - @return tags
    - @exception tag
    - @see tags
    - Ensure tags of the same type are lined up with one-another

Blocks

- Used to describe group of related code
- Should be one line
- Reside immediately one line above the line being commented on
- Should match the indentation of the line
- Single blank line proceedes the comment
- Single lines of code should not be commented, unless complex/unintuitive

```
// For every node in the list...
for( int i = 0; i < nodes.size( ); i++ ) {
        Node node = nodes.elementAt( i );

         // If the node is a leaf, remove it from the tree and print...
        if( node instanceof LeafNode ) {
                tree.removeNode( node );
                System.out.println( node );

         // ...or if the node is binary, recursively print its children...
        } else if( node instanceof BinaryNode ) {
                 printNode( ((BinaryNode)node).getChild( 0 ) );
                 printNode( ((BinaryNode)node).getChild( 1 ) );

         // ...otherwise, simply print the node.
         } else {
                System.out.println( node );
```

```
   }
}
```

Indentation

General

- Use three spaces – not tabs

Blocks

- Open { block at the end of same line
- Lines inside should be indented
- Closing } block on new line with same indentation of open block

```
for( int i = 0; i < nodes.size( ); i++ ) {
   Node node = (Node)nodes.elementAt( i );

   if( node instanceof LeafNode ) {
      System.out.println( node );
   }
}
```

Classes

- First line should declare the name of the class
- If implementing an interface, place on line below

Methods

- First line
  - Return type
  - Name
  - Parameters
- Exceptions should be thrown on the line below

```
public int getSize( ) {
   ...
}

public String getName( DataConnection connection )
throws SQLException {
   ...
}
```

Language Features

Nested Assignments

- Avoid nested assignments

Exceptions

- Only to be used for exceptional circumstances
- Always throw exceptions of an appropriate class

Method Overloading

- Overloaded methods should perform the same task as usual.

Class Template

```
/*
 * @(#) SomeClass.java 1.1 2021/12/15
 *
 * Copyright (c) 2021 Aberystwyth University.
 * All rights reserved.
 *
 */

package uk.ac.aber.cs221.group07.somepackage;

/**
* SomeClass - A class that does something.
* <p>
* How it is used
*
* @author (name)
* @version 0.1 (put status of version here)
* @see (ref to related classes)
*/
public class SomeClass extends SomeParentClass
implements SomeInterface {

 // ////////// //
 // Constants. //
 // ////////// //

 // //////////////// //
 // Class variables. //
 // //////////////// //

 // ////////////// //
 // Class methods. //
 // ////////////// //

 // /////////////////// //
 // Instance variables. //
 // /////////////////// //

 // ///////////// //
 // Constructors. //
 // ///////////// //
 // ////////////////////// //
 // Read/Write properties. //
 // ////////////////////// //
 // ///////////////////// //
// Read-only properties. //
 // ///////////////////// //

 // //////// //
 // Methods. //
 // //////// //
}
```