# 2 Code organization (Mariusz)

**Each new application should be given a new package**. This package should contain the **top-level application class**, and any other **classes that may be of use in other code**. Classes that are specific to the application (dialogs, for example), **should be placed in a sub-package called app.**

Objects that are **not specific** to a particular application **should be in a separate package.** For example, a Diary class might be associated with the Heating Control application, but is also used in the Heating Booking Entry system - it is given its own package (uk.ac.aber.cs221..diary)

*So broadly speaking we follow the same methodology as we did before, seperate the app components into the packages*

Javadoc comments **should be maintained at all times,** listing all packages and their purpose. **package.html files should exist for all packages**. Programmers should check this and the Design Specification to find out w**hich packages should be used for a new class.** Addition of new packages is a design issue.

# 3 Identifier Naming Conventions (Mariusz)

### 3.1 General

When choosing a name we should follow those steps:

• Choose names that are **as self-documenting as possible** indexVariable rather than i.

• Use real world object names for objects, e.g. diaryEntry.

• Use predicate clauses or adjectives for boolean objects or functions, e.g. heatingShouldBeOn.

• Use action verbs for procedures and entries, e.g. removeNode.

• **Use constants rather than variables for constant values.**

### 3.2 Classes and Interfaces - capital letter public class StateMachine ...

public class DataManager ...

For abbreviations (like PAU) we use only Capital letter on first one   
  
public class PauEditor ...

public class GuiResourceBundle …

### 3.3 Methods / variables

Method and variable names start with a lower-case letter, and use capitals to separate words (rather than underscores).

public void buildTree( Node root );

The naming of methods should follow the JavaBeans™ convention. This means that properties should have a get( ) method (or is( ) for booleans), and read-write properties should also have a set( ) method.

For example:

// Read-only size property. public int getSize( );

// Read-write name property. public String getName( );

public void setName( String name );

// Boolean readOnly property. public boolean isReadOnly( );

public void setReadOnly( boolean b );

Indexed properties should normally have get and set methods that allow you to access individual values, or an entire array

### 3.4 Constants

Sane as normal variable

public class File {

public static final String pathSeparator = "\";

...

}

# 4. Class Organisation

**File Structure**

* Every top-level class should be defined in its own file
  + Keep the size of files small
  + Easier to locate classes

**Class Structure**

* Every class should have own variables and method arranged in groups

*public* *class* Application() {

*/\*\**

*\* Java doc comment*

*\*/*

*public* *void* translate() {

variables…; *// Add comment*

Code; *// Add comment what the code does*

}

*/\*\**

*\* Java doc comment*

*\*/*

*public* *void* addNewEnglish() {

variables…; *// Add comment*

Code; *// Add comment what the code does*

}

}

**Inner Class**

* A class in a class
* Should not be used by code outside the parent class unless the inner class could be considered an attribute of the parent class.

*public* *class* Application {

*class* className {

…….

*public* *class* className {

….

}

}

}

**Anonymous**

* Only used to pass simple implementations of an interface as parameters to a method

*public* *class* Application {

*Runnable* runnable = new Runnable() {

*public* *void* run() {

System.err.println(“I’m running!”);

}

};

*Thread* thread = new Thread(runnable);

thread.start();

}

*\*Example from the QA.09 documentation (page 6 of 12)*

# 5. Comments

**Files**

* Header, copyright message, version and date

/\*

\* @(#) SomeClass.java 1.1 2016/10/18

\*

\* Copyright © 2016 Aberystwyth University.

\* All rights reserved.

\*

\*/

*\*Example from the QA.09 documentation (page 7 of 12)*

**Classes and Interfaces**

* javadoc on each class header
  + Description - Overview of the class, separated from the tags by an empty line.
  + @author tag must include except for inner class**.**
  + @version tag must include for **each version** except for inner class.
  + @see tags should be used to **cross-reference related classes**.
  + Anonymous do not need headers.

*\*Example from the QA.09 documentation (page 7 of 12)*

**Methods**

* javadoc need to be included in each method
  + Description : Purpose of method & any side-effect
  + Parameters & Return values: @param & @return tags
  + Same tags need to be **line up**
  + Type of exception: @exception tag (even if there is already a tag for one of the exception’s superclasses)
  + Cross-reference related method or class: @see tags
  + Methods in anonymous & skeletal test classes **do not need header**

**Blocks -** Used to describe a group of related code

* Should be on **one line**
* reside immediately above the block being commented
* Extra lines should each begin with the double slash
* Should be intended to match the indentation of the line of code
* Small blocks of code that do a specific job should be commented but not individual lines unless it is complex or not intuitive
* Often put useful comments before control structures (for loops, ifs, while, etc.)

*\*Example on QA.09 documentation (page 8, 9 of 12)*

# Ad.6 (Sophie)

Indentation

* Standard unit 3 spaces, no tabs

Blocks

* ‘{‘ end of preceding line, the block should be indented, ‘}’ same level as the line that started the block

Example:

for(...){

…

if(...){

…

}

}

* if ‘{’ used in part of a compound statement (else if) should be used in whole statement

i.e:

if(...){

…

} else if(...){

…

} else {

…

}

or

if (...)

…

else if

…

else

…

Classes

* First line of class contains name and parent if there is one. Implements should be on the next line

i.e

public class ExampleIndent extends SeQa

implements Something{

…

}

Methods

* First line of method contains return type, name and parameters, any exceptions thrown stated on next line.

i.e

public String getName( DataConnection connection )

throws SQLExeption {

…

}

\*Can change this example if needed\*

# Ad. 7(Sophie)

Language features

* No nested assignment

i.e

a=b+(c=d\*e))

* Exceptions should never be used to communicate the result of a method.

Where they are needed throw exception of the appropriate class, if the class doesn’t exist a new one should be defined.

* Overload methods share the same name but take different parameters

A class often gives a number of overload methods, all must do the same thing

Appendix (Sophie)

Class template:

File header

Package path

Class header

Class decoration

Constants

Class variable

Class methods

Instance variables

Constructors

Read/Write properties

Read-Only properties

Methods