

## 2. Coding Standards on a Page (or three)

Opt - Indicates Optional Rule for Cambridge Courses.

```
/**
 * MyJavaCodingStandards.java - © BAESystems Detica 2013
 *
 * Not Protectively Marked
 */

import java.util.List;
import java.util.ArrayList;

/**
 * Does cool things. Positively chilly in fact. My kids would say
 * the functionality is "epic". Probably with an American accent.
 * Thanks TV.
 */
class MyJavaCodingStandards {
    Logger log = LoggerFactory.getLogger(MyJavaCodingStandards.class);

    public static final int RANGE_MIN = 0;
    public static final int RANGE_MAX = 100;

    private AnotherClass classVariable;

    MyJavaCodingStandards(){
        classVariable = new AnotherClass();
    }

    public void randomNumber(Integer x){
        if( null == x ){
            throw new IllegalArgumentException("`x` cannot be null");
        }

        MyObject obj = classVariable.getDodgyObject();

        if( null == obj ){
            throw new DodgyObjectException();
        }

        if( x > RANGE_MIN && x < RANGE_MAX ){
            ...
        }
    }

    public void avoidAutoboxing(int x){
        Integer intToAutoBox = x;

        int i = intToAutoBox;

        public static float methodExample(int x){
            int [] a = new int[20];

            float floatValue = (float) x;

            return floatValue;
        }
    }
}
```

Find More Detail in Appendix C.1.5

All code files must have a header with copyright and protective markings **Opt**

Imported classes should listed explicitly, avoid java.util.\* **C.7.3**

All code files must have a description **Opt** **C.1.5**

CamelCase Class names

Constant—All caps with “\_” separators. Avoid magic numbers, declare named constants. **B.5.3**  
**C.2.7**  
**C.11.4**

Member variables should be declared private. Use getter and setter methods. **C.5.3**

Avoid TAB, Use 4 Spaces **Opt**

All input parameters shall be checked for null values. **Opt**  
Similarly, if you call a method that returns an object. **B.1.1**

**Code Defensively!**

All input parameters shall be checked for range validity **Opt** **B.1.3**

Be wary of automatic features such as autoboxing. Null is probably equivalent to 0 in most cases. But the designers of java cannot know that for sure so it is set to null. Autoboxing will throw a NullPointerException if ‘x’ is null as it will call x.intValue(); **B.1.2**

Use this method modifier order, avoid ‘static public float’

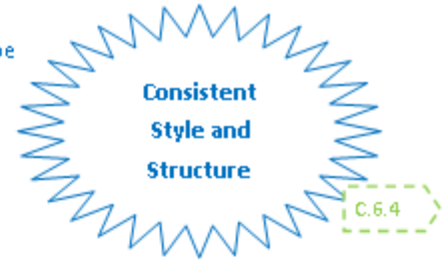
Array specifiers must be attached to the type not the variable. **Opt** **C.10.2**

Type conversions must always be done explicitly. Never rely on implicit type conversion.

Access to classes, methods and variables should be restricted and the final keyword should be used to prevent subclasses exposing functionality that was intended to be protected

```
final public int computeMeaningOfLife(int x){
```

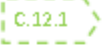
Opt



Place constants on left, to...  
Avoid assignment errors Opt

```
    if( RANGE_MAX == x ){  
        int sum = 0;  
        for ( int i = 0 ; i < 100 ; i++ ){  
            ...  
        }  
    }
```

Only loop control statements must be included in the for () constructions. Do not do for ( int i=0, sum=0; i<100; i++ ). Opt



The use of do-while loops can be avoided.

```
    boolean isDone=false;  
    while ( !isDone ) { ... }
```

Loop variables should be initialised immediately before the loop. Opt



```
    boolean isOK = readFile(fileName);  
    if( isOK ) {  
        ...  
    } else {  
        ...  
    }
```

The most likely case should be put in the if-part and the least likely in the else-part of an if statement. Opt

```
    boolean detica = true;  
    return true == detica ? 42 : 0;
```

Use of the ternary operator (?) should be kept simple. Opt

The use of return should be reserved for the end of the method.

Opt



```
public Danger getSomethingDangerous() throws SomeException {
```



```
    Connection conn = null;  
    try{  
        conn = getConnection();  
    } catch (FileNotFoundException e){
```

Don't handle coding errors with Exceptions. Opt

Use checked exceptions carefully, only throw checked Exceptions if you believe the calling code can take suitable steps to handle and recover. Opt



Make use of existing Exceptions.

```
        log.error("Connection file not found");  
        throw new SomeException(e);  
    } catch (DALErrorException e){  
        log.error("DAL Connection could not be established");  
        throw new SomeException(e);
```

Be specific when catching Exception. Consider carefully about where you catch and handle Exceptions.



```
    } finally {  
        DBUtil.closeConnection(conn);  
    }
```

Don't just swallow Exceptions. Do something meaningful

Top level Exceptions handler must log coherent errors messages



Always clean up after Exceptions

```
    return conn.getDanger();  
}
```



```
import java.util.List;
import java.util.ArrayList;
```

The class and interface declarations should have the following form.

```
public class DatabaseStuff extends Database implements Cloneable, Serializable {
```

C.2.15

```
public executeQuery(String dodgyUserInput){
```

Use prepared statements. Allowing the execution of any String passed into the method leave the Database completely vulnerable to exploit.

...

```
Statement st = null;
```

```
rs = st.exectueQuery("Select * from Table where x =" + dodgyUserInput);
```

Opt

```
String result = fetchOneRowOnly(rs);
```

...

Think about Performance, think how the code will perform on the real datasets. For Example, don't select all then filter, use correct SQL.



B.4

Opt

```
/**
 * Return lateral location of the specified position.
 * If the position is unset, NAN is returned
 *
 * @param x X coordinate
 * @param y Y coordinate
 * @param zone Zone of position
 * @return Lateral location
 * @throws IllegalArguementException If zone is <= 0
 */
```

C.4.3

JavaDoc comments should have the following form.

```
private double computeLocation(double x, double y, int zone)
throws IllegalArguementException{
```

B.2.9

Opt

```
switch (condition){
```

The switch statement should have the following form.

```
case 1 :
```

```
statements;
```

```
/* Falls through */
```

A comment should be added when no break is include.

```
case 2 :
```

```
statements;
```

```
break;
```

```
default :
```

Every switch state ment should include a default case.

```
statements;
```

```
break;
```

Opt

C.5.7

```
Matrix4x4 matrix;
```

```
double cosAngle;
```

```
double sinAngle;
```

Variables in declarations can be left aligned.

C.6.4

Variable names must be mixed case starting with lowercase.

```
// Create a new identity matrix
```

```
matrix = new Matrix4x4();
```

C.6.2

```
// Precompute angles for efficiency
```

```
cosAngle = Math.cos(angle);
```

```
sinAngle = Math.sin(angle);
```

Logical units within a block should be separated by one blank line and commented correctly.

```
// Specify matrix as a rotation transformation
```

```
matrix.setElement(1, 1, cosAngle);
```

```
return 0.0;
```

Use // for all non-JavaDoc comments, including Multi-line comments. The comments should be indented relative to their position in the code.

Opt

C.4.5

C.4.6

Opt

Remove all commented out code

C.1.1

C.1.4

Files should not exceed 1000 lines in length. Do not include more than one class per file. File content must be kept within 120 columns.