2. Coding Standards on a Page (or three)

```java
/**
 * 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 "spic". 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.getMyObject();
        if (null == obj) {
            throw new说服ObjectException();
        }

        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;
        }
    }

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

    All code files must have a description Opt

    CamelCase class names

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

    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.

    All input parameters shall be checked for range validity Opt

    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 an NullPointerException if 'x' is null as it will call x.intValue();

    Use this method modifier order, avoid 'static public float'

    Array specifiers must be attached to the type, not the variable.

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

    Opt - Indicates Optional Rule for Cambridge Courses.
```
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.

```java
final public int computeMeaningOfLife(int x) {
    if (x == RANGE_MAX) {
        // Place constants on left, to...
        int sum = 0;
        for (int i = 0; i < 100; i++) {
            // Avoid assignment errors
            only loop control statements must be included in the for() constructions. Do not do for (int i=0, sum=0; i<100; ++i). Opt
            ...}
        }
        boolean isDone = false;
        // The use of do-while loops can be avoided.
        while (!isDone) {
            ...}
    }
    boolean isOK = readFile(fileName);
    if (isOK) {
        // The most likely case should be put in the if-part and the least likely in the else-part of an if statement. Opt
        ...}
    else {
        ...}
    boolean gotcha = true;
    return true == gotcha ? 42 : 0;
}
```

```java
public void getSomethingDangerous() throws SomeException {
    Connection conn = null;
    try {
        // Don't handle coding errors with Exceptions. Opt
        conn = getConnection();
    } catch (FileNotFoundException e) {
        log.error("Connection file not found");
        throw new SomeException(e);
    } catch (DLEException e) {
        log.error("DLE Connection could not be established");
        throw new SomeException(e);
    } finally {
        // Be specific when catching Exception. Consider carefully about where you catch and handle Exceptions.
        DBUtil.closeConnection(conn);
        // Don't just swallow Exceptions. Do something meaningful!
        // Top level Exceptions handler must log coherent error messages
        return conn.getDanger();
    }
}
```
import java.util.List;
import java.util.ArrayList;

public class DatabaseStuff extends Database implements Cloneable, Serializable {

    public executeQuery(String dodgyUserInput){
        ... Statement st = null;
        rs = st.executeQuery("Select * from Table where x =" + dodgyUserInput);
        String result = fetchOneRowOnly(rs);
        ...

        // 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 IllegalArgumentException If zone is <= 0
        */
        private double computeLocation(double x, double y, int zone)
                throws IllegalArgumentException{
            switch (condition){
                case 1 :
                    statements;
                /* Falls through */
                case 2 :
                    statements;
                    break;
                default :
                    statements;
                    break;
            }

            Matrix4x4 matrix;
            double cosAngle;
            double sinAngle;

            // Create a new identity matrix
            matrix = new Matrix4x4();

            // Precompute angles for efficiency
            cosAngle = Math.cos(angle);
            sinAngle = Math.sin(angle);

            // Specify matrix as a rotation transformation
            matrix.setElement(1, 1, cosAngle);
            return 0.0;
        }

        Matrix4x4 matrix;
        double cosAngle;
        double sinAngle;

        // Create a new identity matrix
        matrix = new Matrix4x4();

        // Precompute angles for efficiency
        cosAngle = Math.cos(angle);
        sinAngle = Math.sin(angle);

        // Specify matrix as a rotation transformation
        matrix.setElement(1, 1, cosAngle);
        return 0.0;
    }

    public void executeNonQuery(String dodgyUserInput){
        /* Do something */
        ... Statement st = null;
        rs = st.executeUpdate("Insert into Table set x =" + dodgyUserInput);
        String result = fetchOneRowOnly(rs);
        ...

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

        // Think about performance, think how the code will per-
        form on the real datasets. For example, don't select all
        then filter, use correct SQL.

        // JavaDoc comments should have the following form.
        // Opt The switch statement should have the following form.
        // Opt A comment should be added when no break is include
        // Opt Every switch statement should include a default case.

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

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

        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.