Internal Project
-
Code Inspection

Davide Cremona (matr. 852365), Simone Deola (matr. 788181)

January 5, 2016

Contents

Chapter 1

Classes Assigned

On this code inspection we take care of the open source code of GlassFish. The class analyzed on this document are "StandardService". It is located on the following pattern:
appserver/web/web-core/src/main/java/org/apache/catalina/core/StandardService.java
The documentation of this specific class can be found on this link:
`http://glassfish.pompel.me`.
We focus on the following methods:

- addConnector(Connector connector)
- removeConnector(Connector connector)
- start( )
- stop( )
- initialize( )

Chapter 2

Functional Role of the Classes Assigned

Here will be given the functional role of the StandardService class and the methods assigned to us.

2.1 The Class

StandardService is the implementation of the "Service" interface that represents a set of Connectors that share a single Container.
A Connector is in charge to receive requests from clients and return responses to them. A Connector follows this logic:

1. Receive a request.
2. Create a Request and Response instance and populate it with the right properties.
3. Identify the right Container (in this case the one shared with the Service) to process the Request.
4. Make the Container process the Request.
5. Return to the client the result obtained from the Container.

A Container is in charge to process requests received from clients and return responses.

2.2 Method: addConnector

This method is in charge to add a new Connector to the set of Connectors of the current Service.

2.2.1 Javadoc

```
380     /**
381      * Add a new Connector to the set of defined Connectors, and
             associate it
382      * with this Service's Container.
383      *
384      * @param connector The Connector to be added
385      */
```

2.2.2 Functioning

All the body of this method is in a Synchronized block, so it's thread-safe.
The Connector that have to be added is initialized with the Container and the current Service:

```
389             connector.setContainer(this.container);
390             connector.setService(this);
```

The Connector is actually added to the set of Connectors:

```
391             Connector results[] = new Connector[connectors.length + 1];
392             System.arraycopy(connectors, 0, results, 0,
                    connectors.length);
393             results[connectors.length] = connector;
394             connectors = results;
```

The Connector is initialized if the Service is already initialized:

```
396             if (initialized) {
397                 try {
398                     connector.initialize();
399                 } catch (LifecycleException e) {
400                     log.log(Level.SEVERE, "Connector.initialize", e);
401                 }
402             }
```

If the Connector implements the Lifecycle interface and the current Service is already started, then the start() method of the Connector is called making him run:

```
404             if (started && (connector instanceof Lifecycle)) {
405                 try {
406                     ((Lifecycle) connector).start();
407                 } catch (LifecycleException e) {
408                     log.log(Level.SEVERE, "Connector.start", e);
409                 }
410             }
```

All the listeners on the Connector are notified of the changes:

```
413             support.firePropertyChange("connector", null, connector);
```

2.3 Method: removeConnector

This method is in charge to remove a new Connector to the set of Connectors of the current Service.

2.3.1 Javadoc

```
467     /**
468      * Remove the specified Connector from the set associated from this
469      * Service. The removed Connector will also be disassociated from our
470      * Container.
471      *
472      * @param connector The Connector to be removed
473      */
```

2.3.2 Functioning

All the body of this method is in a Synchronized block, so it's thread-safe.

The index of the Connector that has to be removed is found:

```
480             int j = -1;
481             for (int i = 0; i < connectors.length; i++) {
482                 if (connector == connectors[i]) {
483                     j = i;
484                     break;
485                 }
486             }
```

If the Connector isn't in the set of Connectors associated to this Service, then the method breaks:

```
488             if (j < 0)
489                 return;
```

The interested Connector is stopped:

```
499             ((Lifecycle) connectors[j]).stop();
```

The Connector is actually removed:

```
506             int k = 0;
507             Connector results[] = new Connector[connectors.length - 1];
508             for (int i = 0; i < connectors.length; i++) {
509                 if (i != j)
510                     results[k++] = connectors[i];
511             }
512             connectors = results;
```

All the listeners on the Connector are notified of the changes:

```
515             support.firePropertyChange("connector", connector, null);
```

2.4 Method: start

This method is in charge to initialize and start all the Connectors and the Container associated to this Service.

2.4.1 Javadoc

```
578     /**
579      * Prepare for the beginning of active use of the public methods of
             this
580      * component. This method should be called before any of the public
581      * methods of this component are utilized. It should also send a
582      * LifecycleEvent of type START_EVENT to any registered listeners.
583      *
584      * @exception LifecycleException if this component detects a fatal
             error
585      *  that prevents this component from being used
586      */
```

2.4.2 Functioning

If the Service is already started, then the information is logged:

```
590         if (started) {
591             if (log.isLoggable(Level.INFO)) {
592                 log.log(Level.INFO, SERVICE_STARTED);
593             }
594         }
```

If the Service is not already initialized, then it will be initialized:

```
596         if( ! initialized )
```

```
597             init();
```

The BEFORE_START_EVENT, STARTING_SERVICE and START_EVENT are logged to the listeners and the "started" flag is set as true:

```
600         lifecycle.fireLifecycleEvent(BEFORE_START_EVENT, null);
601
602         if (log.isLoggable(Level.INFO)) {
603             log.log(Level.INFO, STARTING_SERVICE, this.name);
604         }
605         lifecycle.fireLifecycleEvent(START_EVENT, null);
606         started = true;
```

Connectors and the Container are started:

```
609         if (container != null) {
610             synchronized (container) {
611                 if (container instanceof Lifecycle) {
612                     ((Lifecycle) container).start();
613                 }
614             }
615         }
```

```
618         synchronized (connectorsMonitor) {
619             for (int i = 0; i < connectors.length; i++) {
620                 if (connectors[i] instanceof Lifecycle)
621                     ((Lifecycle) connectors[i]).start();
622             }
623         }
```

All the listeners are notified of the changes:

```
626         lifecycle.fireLifecycleEvent(AFTER_START_EVENT, null);
```

2.5 Method: stop

This method is in charge to stop every Connector and the Container associated to this Service. It is also in charge to avoid the public methods to be called from the outside.

2.5.1 Javadoc

```
631     /**
632      * Gracefully terminate the active use of the public methods of this
633      * component. This method should be the last one called on a given
```

```
634      * instance of this component. It should also send a LifecycleEvent
635      * of type STOP_EVENT to any registered listeners.
636      *
637      * @exception LifecycleException if this component detects a fatal
             error
638      *  that needs to be reported
639      */
```

2.5.2 Functioning

If the Service is not already started then it cannot be stopped:

```
643         if (!started) {
644             return;
645         }
```

The BEFORE_STOP_EVENT, STOP_EVENT and STOPPING_SERVICE are logged to the listeners and the "started" flag is set as false:

```
648         lifecycle.fireLifecycleEvent(BEFORE_STOP_EVENT, null);
649
650         lifecycle.fireLifecycleEvent(STOP_EVENT, null);
651
652         if (log.isLoggable(Level.INFO)) {
653             log.log(Level.INFO, STOPPING_SERVICE, this.name);
654         }
655         started = false;
```

Connectors and Container are stopped:

```
658         synchronized (connectorsMonitor) {
659             for (int i = 0; i < connectors.length; i++) {
660                 if (connectors[i] instanceof Lifecycle)
661                     ((Lifecycle) connectors[i]).stop();
662             }
663         }
```

```
666         if (container != null) {
667             synchronized (container) {
668                 if (container instanceof Lifecycle) {
669                     ((Lifecycle) container).stop();
670                 }
671             }
672         }
```

All the listeners are notified of the changes:

```
675         lifecycle.fireLifecycleEvent(AFTER_STOP_EVENT, null);
```

2.6 Method: initialize

This method is in charge to initialize the Service for the active use.

2.6.1 Javadoc

```
680     /**
681      * Invoke a pre-startup initialization. This is used to allow
             connectors
682      * to bind to restricted ports under Unix operating environments.
683      */
```

2.6.2 Functioning

If the Service is already initialized then the SERVICE_HAS_BEEN_INIT message is notified to the listeners and the method will return; otherwise, the "initialized" flag is set as true:

```
688         if (initialized) {
689             if (log.isLoggable(Level.INFO)) {
690                 log.log(Level.INFO, SERVICE_HAS_BEEN_INIT);
691             }
692             return;
693         }
694         initialized = true;
```

The Object Name is initialized and the Service is registered in the current Server:

```
696         if( oname==null ) {
697             try {
698                 // Hack - Server should be deprecated...
699                 Container engine=this.getContainer();
700                 domain=engine.getName();
701                 oname=new ObjectName(domain +
                        ":type=Service,serviceName="+name);
702             } catch (Exception e) {
703                 String msg =
                        MessageFormat.format(rb.getString(ERROR_REGISTER_SERVICE_EXCEPTION),
                        domain);
704                 log.log(Level.SEVERE, msg, e);
705             }
706
707
```

```
708         }
709         if( server==null ) {
710             // Register with the server
711             // HACK: ServerFactory should be removed...
712
713             ServerFactory.getServer().addService(this);
714         }
```

Associated Connectors are initialized:

```
718         synchronized (connectorsMonitor) {
719                 for (int i = 0; i < connectors.length; i++) {
720                     connectors[i].initialize();
721                 }
722         }
723     }
```

Chapter 3

Issues Found

This chapter is divided in sections. In each Section there are listed all the issues found inspecting the code of the methods assigned to us. The first Section ("Class Issues") is for the issues that are related to the entire class and not to a specific method.

3.1 Class Issues

On this section are spotted all the issues founded on the class following the assigned checklist. It is considered the points of the checklist that can be considered only as "global error". The "specific error" issues are considered only for the method assigned and can be found on the following sections.

- **Checklist C.01**

```
90      private static final ResourceBundle rb =
            log.getResourceBundle();
```

"rb" does not mean anything.

```
151     private LifecycleSupport lifecycle = new LifecycleSupport(this);
```

"lifecycle" does not suggest that is a lifecycleSupport.

```
168     protected int debug = 0;
```

"debug" does not suggest that is a debug level variable.

```
200     private NotificationBroadcasterSupport broadcaster = null;
```

"broadcaster" does not suggest that is a NOTIFICATION broadcaster.

```
279     public int getDebug() {
```

Does not suggest that is the debug level.

```
291     public void setDebug(int debug) {
```

Does not suggest that is the debug level.

```
745     protected ObjectName oname;
```

"oname" is meaningless.

- **Checklist C.07**

```
90      private static final Logger log = StandardServer.log;
91      private static final ResourceBundle rb =
            log.getResourceBundle();
```

```
138     private static final String info =
139         "org.apache.catalina.core.StandardService/1.0";
```

```
182     protected final Object connectorsMonitor = new Object();
```

Final attributes but not uppercase and separated by an underscore.

- **Checklist C.08**

  Lines 87 to 132, 203 to 214 and 355 to 374 have an additional space character at the beginning of the line. These spaces make the indentation not correct.

- **Checklist C.12**

```
77  /**
78   *  implementation of the <code>Service</code> interface. The
79   * associated Container is generally an instance of Engine, but
         this is
80   * not required.
81   *
82   * @author Craig R. McClanahan
83   */
84
85  public class StandardService
86         implements Lifecycle, Service
87  {
```

The line 84 is a blank line that should not divide the javadoc from the prototype of the method.

```
205     /**
206      * Construct a default instance of this class.
207      */
208
209     public StandardService() {
```

The line 208 is a blank line that should not divide the javadoc from the prototype of the method.

```
364     /**
365      * Set the <code>NotificationBroadcasterSupport</code> that
             sends notification for this Service
366      *
367      * @param broadcaster The new NotificationBroadcasterSupport
368      */
369
370     public void setBroadcaster(NotificationBroadcasterSupport
             broadcaster) {
```

The line 369 is a blank line that should not divide the javadoc from the prototype of the method.

The last line of the class (line 756) is useless.

- **Checklist C.15**

```
85  public class StandardService
86         implements Lifecycle, Service
```

Line break occurs after a space.

- **Checklist C.17**

  All the class lines have an offset of one space character from their level of indentation.

- **Checklist C.23**

```
96      public static final String SERVICE_STARTED =
            "AS-WEB-CORE-00251";
```

```
102     public static final String STARTING_SERVICE =
            "AS-WEB-CORE-00252";
```

```
108     public static final String STOPPING_SERVICE =
            "AS-WEB-CORE-00253";
```

```
114     public static final String SERVICE_HAS_BEEN_INIT =
            "AS-WEB-CORE-00254";
```

```
122     public static final String ERROR_REGISTER_SERVICE_EXCEPTION =
            "AS-WEB-CORE-00255";
```

```
130     public static final String FAILED_SERVICE_INIT_EXCEPTION =
            "AS-WEB-CORE-00256";
```

No javadoc for public static attribute.

```
268    public ObjectName getContainerName() {
```

```
418    public ObjectName[] getConnectorNames() {
```

```
725    public void destroy() throws LifecycleException {
```

```
735    public void init() {
```

```
747    public ObjectName getObjectName() {
```

```
751    public String getDomain() {
```

No javadoc for public method.

```
279    public int getDebug() {
```

```
305    public String getInfo() {
```

```
315    public String getName() {
```

```
337    public Server getServer() {
```

```
358     public NotificationBroadcasterSupport getBroadcaster() {
```

```
445    public Connector[] findConnectors() {
```

```
536     public String toString() {
```

```
563     public List<LifecycleListener> findLifecycleListeners() {
```

Better to use "@return" javadoc command instead of writing in the description field.

```
455     public Connector findConnector(String name) {
```

Better to use "@return" javadoc command instead of writing in the description field, no "@param" javadoc field for tne "String name" parameter.

- **Checklist C.25**

```
89      private static final Logger log = StandardServer.log;
90      private static final ResourceBundle rb =
            log.getResourceBundle();
```

Private static attribute stated before a public static one.

```
145     private String name = null;
```

```
151     private LifecycleSupport lifecycle = new LifecycleSupport(this);
```

```
157     private Server server = null;
```

```
162     private boolean started = false;
```

Private attribute stated before a protected one.

```
744     protected String domain;
745     protected ObjectName oname;
```

Protected attribute not in the attributes section.

- **Checklist C.26**

```
744     protected String domain;
745     protected ObjectName oname;
746
747     public ObjectName getObjectName() {
748         return oname;
749     }
750
```

```
751     public String getDomain() {
752         return domain;
753     }
```

These methods and attributes are not grouped by functionality, scope or accessibility.

- **Checklist C.39**

  Note on issue C.39: not everytime a new array is created his elements are initialized using the constructor, but in these cases it's not a problem because the elements are copied from another array that was initialized before.

3.2 Method: addConnector

On this section are spotted all the issues founded on the method addConnector following the assigned checklist. The method addConnector is located from the line 380 to the line 427 on the StandardService class.

- **Checklist C.33**

```
391             Connector results[] = new Connector[connectors.length +
                    1];
```

These declarations are not at the top of the code block.

- **Checklist C.51**

```
390             connector.setService(this);
```

"this" is of the type StandardService, where the parameter of setService is "Service" (see the Javadoc for setService() of the class Connector).

```
392             System.arraycopy(connectors, 0, results, 0,
                    connectors.length);
```

Connectors and result are of the type "Connector" where the types of the first and the third parameters of System.arraycopy are "Object" (see the Javadoc for arrayCopy() of the class System).

```
400                     log.log(Level.SEVERE, "Connector.initialize", e);
```

The type of "e" is "LifecycleException" where the type of the third parameter of Logger.log() is "Throwable" (see the Javadoc for log() of the class Log).

```
408                     log.log(Level.SEVERE, "Connector.start", e);
```

The type of “e” is "LifecycleException" where the type of the third parameter of Logger.log() is "Throwable" (see the Javadoc for log() of the class Log).

```
413             support.firePropertyChange("connector", null,
                    connector);
```

Type of the the third parameter is “Object” and "connector" is of the type "Connector" (see the Javadoc for firePropertyChange() of the class PropertyChangeSupport).

These lines of code contains implicit types conversions.

3.3 Method: removeConnector

On this section are spotted all the issues founded on the method removeConnector following the assigned checklist. The method removeConnector is located from the line 467 to the line 518 on the StandardService class.

- **Checklist C.02**

  In this method it's declared "j", an integer variable that is not used for temporary use. Replace with a significative name.

- **Checklist C.11**

```
488             if (j < 0)
489                 return;
```

```
509                 if (i != j)
510                     results[k++] = connectors[i];
```

These lines of code contains if statements with only one statement to execute and are not surrounded by curly braces.

- **Checklist C.18**

```
474     // START SJSAS 6231069
```

```
477     // END SJSAS 6231069
```

```
491             // START SJSAS 6231069
```

```
500             // END SJSAS 6231069
```

```
502             // START SJSAS 6231069
```

```
505             // END SJSAS 6231069
```

These comments don't explain what the code are doing.

- **Checklist C.19**

```
475     //public void removeConnector(Connector connector) {
```

```
492             /*if (started && (connectors[j] instanceof Lifecycle)) {
493                 try {
494                     ((Lifecycle) connectors[j]).stop();
495                 } catch (LifecycleException e) {
496                     log.error("Connector.stop", e);
497                 }
498             }*/
```

```
503             /*connectors[j].setContainer(null);
504             connector.setService(null);*/
```

Commented code does not contain a reason for being commented out.

- **Checklist C.23**

```
476     public void removeConnector(Connector connector) throws
            LifecycleException{
```

No "@throws" javadoc field for the exception "LifecycleException"

- **Checklist C.33**

```
506             int k = 0;
507             Connector results[] = new Connector[connectors.length -
                    1];
```

Declarations are not at the top of the block.

- **Checklist C.40**

```
482             if (connector == connectors[i]) {
```

Two objects are compared using "==" and not equals().

- **Checklist C.44**

```
480         int j = -1;
481         for (int i = 0; i < connectors.length; i++) {
482             if (connector == connectors[i]) {
483                 j = i;
484                 break;
485             }
486         }
```

There is a "break;" into the for() block. Use another iteration block.

```
510                 results[k++] = connectors[i];
```

It's better to explicitly increment 'k' before the assignment with a separed statement.

These lines of code does not avoid "Brutish Programming".

- **Checklist C.51**

```
515         support.firePropertyChange("connector", connector,
                null);
```

The type of the second parameter has to be "Object" and "connector" is of the type "Connector".

- **Checklist C.56**

```
480         int j = -1;
481         for (int i = 0; i < connectors.length; i++) {
482             if (connector == connectors[i]) {
483                 j = i;
484                 break;
485             }
486         }
```

Loops are not correctly formed expecially the termination expression at the line 484 (break).

3.4 Method: start

On this section are spotted all the issues founded on the method start following the assigned checklist. The method start is located from the line 578 to the line 628 on the StandardService class.

- **Checklist C.11**

```
596         if( ! initialized )
597             init();
```

```
620                 if (connectors[i] instanceof Lifecycle)
621                     ((Lifecycle) connectors[i]).start();
```

These lines of code contains if statements with only one statement to execute and are not surrounded by curly braces.

- **Checklist C.51**

```
603             log.log(Level.INFO, STARTING_SERVICE, this.name);
```

The type of the third parameter is "Object" where this.name is String.

3.5 Method: stop

On this section are spotted all the issues founded on the method stop following the assigned checklist. The method stop is located from the line 631 to the line 677 on the StandardService class.

- **Checklist C.11**

```
660                 if (connectors[i] instanceof Lifecycle)
661                     ((Lifecycle) connectors[i]).stop();
```

These lines of code contains if statements with only one statement to execute and are not surrounded by curly braces.

- **Checklist C.51**

```
653             log.log(Level.INFO, STOPPING_SERVICE, this.name);
```

Type of the third parameter is "Object" where this.name is String.

3.6 Method: initialize

On this section are spotted all the issues founded on the method initialize following the assigned checklist. The method initialize is located from the line 680 to the line 723 on the StandardService class.

- **Checklist C.13**

```
703                 String msg =
                        MessageFormat.format(rb.getString(ERROR_REGISTER_SERVICE_EXCEPTION),
                        domain);
```

Limit of 80 characters exceeded.

```
739             String msg =
                    MessageFormat.format(rb.getString(FAILED_SERVICE_INIT_EXCEPTION),
                    domain);
```

Limit of 80 characters exceeded.

- **Checklist C.18**

```
687         // Service shouldn't be used with embedded, so it doesn't
                matter
```

```
698                 // Hack - Server should be deprecated...
```

```
711             // HACK: ServerFactory should be removed...
```

These three comments don't explain what the code do. These comments are directives to the developers.

- **Checklist C.23**

```
684     public void initialize()
685             throws LifecycleException
```

No "@throws" javadoc field for the exception "LifecycleException"

- **Checklist C.51**

```
703                 String msg =
                        MessageFormat.format(rb.getString(ERROR_REGISTER_SERVICE_EXCEPTION),
                        domain);
```

"domain" is a String and it's implicitly converted into "Object"

```
704                 log.log(Level.SEVERE, msg, e);
```

“e” is an Exception and it’s implicitly converted into “Object”

```
713             ServerFactory.getServer().addService(this);
```

“this” is a StandardService and it’s implocitly converted into ”Service”

- **Checklist C.53**

```
702             } catch (Exception e) {
```

”MalformedObjectNameException” should be caught instead of ”Exception”

Chapter 4

Other Problems

1.
```
418     public ObjectName[] getConnectorNames() {
419         ObjectName results[] = new ObjectName[connectors.length];
420         for( int i=0; i<results.length; i++ ) {
421             // if it's a coyote connector
422             //if( connectors[i] instanceof CoyoteConnector ) {
423             //
                    results[i]=((CoyoteConnector)connectors[i]).getJmxName();
424             //}
425         }
426         return results;
427     }
```

   This method does nothing, it returns only an empty array.

2. The two methods "init()" and "initialize()" perform the same functionality and are public. This means that if i want to call the initialization process i don't know what is the right method. Make the initialization univocal, to do that eliminate one method or make one of it private.

3.
```
496                 log.error("Connector.stop", e);
```

   Can't find this function (log.error(*)) in the documentation.

4. The two function addConnector() and removeConnection() are not fully commented, the first part of each class is not so explanatory and need some comment for the steps of the algorithms.

Chapter 5

Additional Material

5.1 Code Inspection Checklist

This is the checklist used for the code inspection:

- **Naming Conventions**
  - C.01) All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.
  - C.02) If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.
  - C.03) Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;
  - C.04) Interface names should be capitalized like classes.
  - C.05) Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().
  - C.06) Class variables, also called attributes, are mixed case, but might begin with an underscore ('_') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: _windowHeight, timeSeriesData.
  - C.07) Constants are declared using all uppercase with words separated by an underscore. Examples: MIN_WIDTH; MAX_HEIGHT;
- **Indention**
  - C.08) Three or four spaces are used for indentation and done so consistently.
  - C.09) No tabs are used to indent.

- **Braces**
  - C.10) Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block).
  - C.11) All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example: Avoid this:

```
1                    if ( condition )
2                       doThis();
```

    Instead do this:

```
1                    if ( condition ) {
2                       doThis();
3                    }
```

- **File Organization**
  - C.12) Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).
  - C.13) Where practical, line length does not exceed 80 characters.
  - C.14) When line length must exceed 80 characters, it does NOT exceed 120 characters.
- **Wrapping Lines**
  - C.15) Line break occurs after a comma or an operator.
  - C.16) Higher-level breaks are used.
  - C.17) A new statement is aligned with the beginning of the expression at the same level as the previous line.
- **Comments**
  - C.18) Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
  - C.19) Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.
- **Java Source Files**
  - C.20) Each Java source file contains a single public class or interface.

- C.21) The public class is the first class or interface in the file.
- C.22) Check that the external program interfaces are implemented consistently with what is described in the javadoc.
- C.23) Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

- **Package and Import Statements**
  - C.24) If any package statements are needed, they should be the first non- comment statements. Import statements follow.

- **Class and Interface Declarations**
  - C.25) The class or interface declarations shall be in the following order:
    1. Class/interface documentation comment.
    2. Class or interface statement.
    3. Class/interface implementation comment, if necessary.
    4. Class (static) variables.
       (a) First public class variables.
       (b) Next protected class variables.
       (c) Next package level (no access modifier).
       (d) Last private class variables.
    5. Instance variables.
       (a) First public instance variables.
       (b) Next protected instance variables.
       (c) Next package level (no access modifier).
       (d) Last private instance variables.
    6. Constructors.
    7. Methods.
  - C.26) Methods are grouped by functionality rather than by scope or accessibility.
  - C.27) Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

- **Initialization and Declarations**
  - C.28) Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).
  - C.29) Check that variables are declared in the proper scope.
  - C.30) Check that constructors are called when a new object is desired.
  - C.31) Check that all object references are initialized before use.
  - C.32) Variables are initialized where they are declared, unless dependent upon a computation.

  - C.33) Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces "{" and "}"). The exception is a variable can be declared in a 'for' loop.

- **Method Calls**
  - C.34) Check that parameters are presented in the correct order.
  - C.35) Check that the correct method is being called, or should it be a different method with a similar name.
  - C.36) Check that method returned values are used properly.

- **Arrays**
  - C.37) Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).
  - C.38) Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
  - C.39) Check that constructors are called when a new array item is desired.

- **Object Comparison**
  - C.40) Check that all objects (including Strings) are compared with "equals" and not with "==".

- **Output Format**
  - C.41) Check that displayed output is free of spelling and grammatical errors.
  - C.42) Check that error messages are comprehensive and provide guidance as to how to correct the problem.
  - C.43) Check that the output is formatted correctly in terms of line stepping and spacing.

- **Computation, Comparisons and Assignments**
  - C.44) Check that the implementation avoids "brutish programming: (see `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html`).
  - C.45) Check order of computation/evaluation, operator precedence and parenthesizing.
  - C.46) Check the liberal use of parenthesis is used to avoid operator precedence problems.
  - C.47) Check that all denominators of a division are prevented from being zero.

  - C.48) Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
  - C.49) Check that the comparison and Boolean operators are correct.
  - C.50) Check throw-catch expressions, and check that the error condition is actually legitimate.
  - C.51) Check that the code is free of any implicit type conversions.

- **Exceptions**
  - C.52) Check that the relevant exceptions are caught.
  - C.53) Check that the appropriate action are taken for each catch block.

- **Flow of Control**
  - C.54) In a switch statement, check that all cases are addressed by break or return.
  - C.55) Check that all switch statements have a default branch.
  - C.56) Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

- **Files**
  - C.57) Check that all files are properly declared and opened.
  - C.58) Check that all files are closed properly, even in the case of an error.
  - C.59) Check that EOF conditions are detected and handled correctly.
  - C.60) Check that all file exceptions are caught and dealt with accordingly.

Chapter 6

Appendix

6.1 Used Software

To create this document we have used some common softwares:

- For the latex we have used two different softwares:
  - Simone Deola: TexShop, provided with the MacTex package (link)
  - Davide Cremona: Sublime Text editor (link) with LaTeXTools (link) and the Basic Package of MacTex (link)

6.2 Document References

- Javadoc of GlassFish: `http://glassfish.pompel.me`
- Glassfish website: `https://glassfish.java.net`
- Brutish programming definitions: `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html`

6.3 Hours of Work

- Cremona Davide (852365): 15 Hours
- Deola Simone (788181): 15 Hours