Clean Code

Presented By - Anil Kumar Sakala

Agenda

- We all write poor code and what happens if I write poor code ?
- We all must write good code
- How to write good code

We all write poor code

- Let us look at few classes and learn from them
  - HSCReceiveExamResultsService.class
  - JobOrdEeoRptgUnitDAOImpl.class
  - JobOrdPQQstnDAOImpl.class

- Writing code that works is not enough .
  - We write our code only once but we read our code hundreds of times .
  - Code should be highly readable and maintainable
  - This presentation talks about discipline that needs to be followed when writing code

Bad Code

- Difficult to read ： Bad code is very difficult to read and understand . We struggle to see what your code is doing and keep looking for some hint on what is going but we see more and more senseless code
- Difficult to change : You change something it breaks something
- Productivity goes down
- With each change you make your code becomes more worst
- Finally , Project become un maintainable . Every day you get production defects and in the process of fixing them we make it more worst.
- Finally customer satisfaction is less and huge revenue loss.

Clean Code

- Code is highly readable . You write your code only once but read your code multiple times . Writing code that works is not enough.
- Code is testable.
- Code is maintainable.
- Productivity is high
- Less number of defects in production

Discipline to ensure high quality code

- Define coding practices and ensure every one in your team follows
- Use sonar and coding template techniques to ensure everyone follows coding practices
- Ensure code reviews happen in your project.
- Unit cases are crucial and vital
- Obey Boy Scout Rule – “Over a period of time your code must not degrade . If we all checked-in our code a little cleaner than when we checked it out, the code simply could not rot. The cleanup doesn’t have to be something big. Change one variable name for the better, break up one function that’s a little too large, eliminate one small bit of duplication, clean up one composite if statement. ”

Naming Conventions

- We name everything – files , jars , variables , classes , packages , functions . We do it many times and how well we do it makes lot of difference .
- Finding good name is a tough job and we spend days together to name your kids , similar you need to spend lot of time to get good name for your software elements
- **Use intention revealing names**.
  - Projects name is HRX , HRW ? Do you understand what it means ?
  - FileInputStream , FileInputReader
  - XXXXDao
  - Int d vs int fileAgeInDays

Naming Conventions

```
public List<int[]> getThem() {
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
     if (x[0] == 4)
       list1.add(x);
   return list1;
 }
```

- What this code is doing ?

Naming Conventions

```
public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new ArrayList<int[]>();
  for (int[] cell : gameBoard)
   if (cell[STATUS_VALUE] == FLAGGED)
     flaggedCells.add(cell);
   return flaggedCells;
}
```

```
public List<Cell> getFlaggedCells() {
   List<Cell> flaggedCells = new ArrayList<Cell>();
   for (Cell cell : gameBoard)
     if (cell.isFlagged())
       flaggedCells.add(cell);
   return flaggedCells;
}
```

Naming Conventions

- **Avoid Disinformation**
  - Do not refer to a grouping of accounts as an accountList unless it's actually a List. The word list means something specific to programmers. If the container holding the accounts is not actually a List, it may lead to false conclusions.
    So accountGroup or bunchOfAccounts or just plain accounts would be better.

- **Make meaningful Distinctions**
  - public static void copyChars(char a1[], char a2[])

    vs public static void copyChars(char source[], char destination[])
  - How is NameString better than Name?
  - How is customerObject better than customer?
  - How is money different from moneyAmount

Naming Conventions

- **Use pronounceable names**

```
class DtaRcrd102 {
        private Date genymdhms;
        private Date modymdhms;
        private final String pszqint = ”102”;
         /* … */
       };
```

to

```
  class Customer {
        private Date generationTimestamp;
        private Date modificationTimestamp;;
        private final String recordId = ”102”;
        /* … */
       };
```

Naming Conventions

- **Avoid using member prefixes**
  - In olden days there is a convention to use _ notation for instance variables like m_desc
  - This convention is no more used as editing environment helps you in distinguishing those variables
  - People quickly learn to ignore the prefix (or suffix) to see the meaningful part of the name

- **Class Names**
  - Should be nouns Ex : Customer , WikiPage , Account
  - Descriptive
  - Difficult to name classes that are doing more than one thing

Naming Conventions

- **Method Names**
  - Method names should be verbs like postPayment , deletePage , save.
  - Avoid overloaded constructor by using static factory method

- **Final Words**
  - Finding good names will take time but increases readability of code drastically
  - Doing more than one thing in a class / method makes it very difficult to get correct name
  - Don’t hesitate to change name once you found a better name for a class / method.

Functions

- Function should be in relevant class and avoid duplicate code.
- Should be small as small as possible . It can be as small as 10 lines
- Name of the function should describe what is going on in method . If you are finding difficult to name your function it means you are trying to do many things in your function .
- The ideal number of arguments for a function is zero . To reduce number of arguments use objects

```
Circle makeCircle(double x, double y, double radius);
Circle makeCircle(Point center, double radius);
```

- Flag arguments need to be avoided
- In method arguments use interfaces and not classes.
- Add preconditions and encourage fast fail
- Make defensive copies of input arguments if required

Functions

- Blocks within if statements, else statements, while statements, and so on should be one line long.

```
If(isEligibleForPension()){
    calculatePension();
}
```

- Error handling is one thing – A function handling errors should do nothing else . This implies that if the key work try exists in a function , it should be very first word in the function and that there should be nothing after the catch/finally blocks.
- Return empty arrays or collections and not nulls.
- Be careful when returning mutable objects
- Write doc comments for all exposed API [Item 44 : Effective Java]
- Finally, When you write a article or blog you get your thoughts down first, then you refine it until it reads well. The first draft might be clumsy and disorganized, so you wordsmith it and restructure it and refine it until it reads the way you want it to read. Similarly when you write your method it looks long with lot of loops , conditions and other elements in it. Refine the code continuously to make it look better and finally get what you want.

Review what has been learned – Do code review on below code

```
public class Course {

    private Date startDate;
    private Date endDate;
    private String course;
    private List<Employee> enrolledEmployees = new ArrayList<Employee>();

    public Course(String courseName, Date startDate, Date endDate) {
        this.startDate = startDate;
        this.endDate = endDate;
        this.course = courseName;
    }

    public void enroll(Employee employee) {
        enrolledEmployees.add(employee);
    }

    public List<Employee> getEnrolledEmployees() {
        return enrolledEmployees;
    }

    public Date getStartDate() {
        return startDate;
    }

    public Date getEndDate() {
        return endDate;
    }

    public String getCourse() {
        return course;
    }
}
```

Comments

- Comments are bad and we should write comments only if we cannot replace comments with expressive code which is almost possible every time.

```
if(age>0 && totalExperience > 30 &&  isRetired ){ // is eligible for pension
  // calculate pension
  // Write business logic here
}
```

- You can remove comments by replacing it with expressive code

```
if(isEligibleForPension()){
    calculatePension();
}
```

- Comments are bad because we don’t trust our comments. Because there is no compile time protection you can write comment that is misleading or has nothing to do with code you write . They are maintenance headache and code readers trust code and not comment

Bad comments - Mumbling

- When you are writing comment make sure it is best comment

```
public void loadProperties()
{
  try
  {
   String propertiesPath = propertiesLocation +
    "/" + PROPERTIES_FILE;
   FileInputStream propertiesStream = new
    FileInputStream(propertiesPath);
   loadedProperties.load(propertiesStream);
  }
  catch(IOException e)
  {
    // No properties files means all defaults are loaded
  }
}
```

- Who loads defaults ?
- Where are they loaded ?
- Is the author trying to say come back here and load defaults ?

Bad Comments – Redundant comments

```
/**
 * The container event listeners for this Container.
 */
protected ArrayList listeners = new ArrayList();

/**
 * The Logger implementation with which this Container is
 * associated.
 */
protected Log logger = null;
```

```
  /**
   * Default constructor.
   */
  protected AnnualDateRule() {
  }

  /**
    * Returns the day of the month.
    *
    * @return the day of the month.
    */
   public int getDayOfMonth() {
     return dayOfMonth;
  }
```

Bad comments – Journal comments

```
* Changes (from 11-Oct-2001)
* --------------------------
* 11-Oct-2001 : Re-organised the class and moved it to new package
*               com.jrefinery.date (DG);
* 05-Nov-2001 : Added a getDescription() method, and eliminated NotableDate
*               class (DG);
* 12-Nov-2001 : IBD requires setDescription() method, now that NotableDate
*               class is gone (DG);  Changed getPreviousDayOfWeek(),
*               getFollowingDayOfWeek() and getNearestDayOfWeek() to correct
*               bugs (DG);
* 05-Dec-2001 : Fixed bug in SpreadsheetDate class (DG);
* 29-May-2002 : Moved the month constants into a separate interface
*               (MonthConstants) (DG);
* 27-Aug-2002 : Fixed bug in addMonths() method, thanks to N???levka Petr (DG);
* 03-Oct-2002 : Fixed errors reported by Checkstyle (DG);
* 13-Mar-2003 : Implemented Serializable (DG);
* 29-May-2003 : Fixed bug in addMonths method (DG);
* 04-Sep-2003 : Implemented Comparable.  Updated the isInRange javadocs (DG);
* 05-Jan-2005 : Fixed bug in addYears() method (1096282) (DG);
```

Bad comments – Mandated comments

Listing 4-3

```
/**
 *
 * @param title The title of the CD
 * @param author The author of the CD
 * @param tracks The number of tracks on the CD
 * @param durationInMinutes The duration of the CD in minutes
 */
public void addCD(String title, String author,
                  int tracks, int durationInMinutes) {
  CD cd = new CD();
  cd.title = title;
  cd.author = author;
  cd.tracks = tracks;
  cd.duration = duration;
  cdList.add(cd);
}
```

Bad comments – Commented out code

Consider this from apache commons:

```
this.bytePos = writeBytes(pngIdBytes, 0);
//hdrPos = bytePos;
writeHeader();
writeResolution();
//dataPos = bytePos;
if (writeImageData()) {
    writeEnd();
    this.pngBytes = resizeByteArray(this.pngBytes, this.maxPos);
}

else {
     this.pngBytes = null;
}
return this.pngBytes;
```

Bad comments – HTML comments

```
/**
 * Task to run fit tests.
 * This task runs fitnesse tests and publishes the results.
 * <p/>
 * <pre>
 * Usage:
 * &lt;taskdef name=&quot;execute-fitnesse-tests&quot;
 *     classname=&quot;fitnesse.ant.ExecuteFitnesseTestsTask&quot;
 *     classpathref=&quot;classpath&quot; /&gt;
 * OR
 * &lt;taskdef classpathref=&quot;classpath&quot;
 *             resource=&quot;tasks.properties&quot; /&gt;
 * <p/>
 * &lt;execute-fitnesse-tests
 *     suitepage=&quot;FitNesse.SuiteAcceptanceTests&quot;
 *     fitnesseport=&quot;8082&quot;
 *     resultsdir=&quot;${results.dir}&quot;
 *     resultshtmlpage=&quot;fit-results.html&quot;
 *     classpathref=&quot;classpath&quot; /&gt;
 * </pre>
 */
```

Bad comments – Too much information

```
/*
  RFC 2045 - Multipurpose Internet Mail Extensions (MIME)
  Part One: Format of Internet Message Bodies
  section 6.8.  Base64 Content-Transfer-Encoding
  The encoding process represents 24-bit groups of input bits as output
  strings of 4 encoded characters. Proceeding from left to right, a
  24-bit input group is formed by concatenating 3 8-bit input groups.
  These 24 bits are then treated as 4 concatenated 6-bit groups, each
  of which is translated into a single digit in the base64 alphabet.
  When encoding a bit stream via the base64 encoding, the bit stream
  must be presumed to be ordered with the most-significant-bit first.
  That is, the first bit in the stream will be the high-order bit in
  the first 8-bit byte, and the eighth bit will be the low-order bit in
  the first 8-bit byte, and so on.
*/
```

Bad comments – Closing brace comments

Listing 4-6 wc.java

```
public class wc {
  public static void main(String[] args) {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line;
    int lineCount = 0;
    int charCount = 0;
    int wordCount = 0;
    try {
       while ((line = in.readLine()) != null) {
         lineCount++;
         charCount += line.length();
         String words[] = line.split("\\W");
         wordCount += words.length;
       } //while
       System.out.println("wordCount = " + wordCount);
       System.out.println("lineCount = " + lineCount);
       System.out.println("charCount = " + charCount);
    } // try
    catch (IOException e) {
      System.err.println("Error:" + e.getMessage());
    } //catch
  } //main
}
```

Good comments – Informative comments

```
// format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher = Pattern.compile(
  "\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

```
public void testConcurrentAddWidgets() throws Exception {
  WidgetBuilder widgetBuilder =
    new WidgetBuilder(new Class[]{BoldWidget.class});
    String text = """bold text""";
    ParentWidget parent =
      new BoldWidget(new MockWidgetRoot(), """bold text""");
    AtomicBoolean failFlag = new AtomicBoolean();
    failFlag.set(false);

    //This is our best attempt to get a race condition
   //by creating large number of threads.
    for (int i = 0; i < 25000; i++) {
      WidgetBuilderThread widgetBuilderThread =
        new WidgetBuilderThread(widgetBuilder, text, parent, failFlag);
      Thread thread = new Thread(widgetBuilderThread);
      thread.start();
    }
    assertEquals(false, failFlag.get());
  }
```

Good comments – Warning of consequences

```
// Don't run unless you
// have some time to kill.
public void _testWithReallyBigFile()
{
  writeLinesToFile(10000000);

  response.setBody(testFile);
  response.readyToSend(this);
  String responseString = output.toString();
  assertSubString("Content-Length: 1000000000", responseString);
  assertTrue(bytesSent > 1000000000);
}
```

Good comments – TODO comments & Amplification

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
protected VersionInfo makeVersion() throws Exception
{
  return null;
}
```

```
String listItemContent = match.group(3).trim();
// the trim is real important.  It removes the starting
// spaces that could cause the item to be recognized
// as another list.
new ListItemWidget(this, listItemContent, this.level + 1);
return buildList(text.substring(match.end()));
```

Java Docs

- API contract and comments are different . API contract play a key role when defining public APIs.
- Look at this method signature

  Public List<Employee> getTerminatedEmployees(String branch)

  Look at this method signature
- Method signature doesn't answer many questions
  - What happens if branch is null or empty
  - Will this method return null or empty list
- Look at how API contracts are defined for existing core API's to understand better way

Formatting

- Code formatting is important as it makes your code readable.
- It is important your team agrees to a set of formatting rules and apply them in your project consistently to increase readability of code
- News Paper Metaphor :Think of a well-written newspaper article. You read it vertically. At the top you expect a headline that will tell you what the story is about . The first paragraph gives you a synopsis of the whole story, hiding all the details while giving you the broad-brush concepts. How to write good code . Similar lines file name should give high level about the contents of file as you read from top to bottom it should contain more details.
- Vertical formatting is about how big your file should be and how many methods it can have .
- Horizontal formatting is about how long your code statement can exists . Recommended is 120 , it is all about readability.

Illustrate importance of formatting

```
public class FitNesseServer implements SocketServer { private FitNesseContext
context; public FitNesseServer(FitNesseContext context) { this.context =
context; } public void serve(Socket s) { serve(s, 10000); } public void
serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = new
FitNesseExpediter(s, context);
sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
catch(Exception e) { e.printStackTrace(); } } }
```

```
public class FitNesseServer implements SocketServer {
  private FitNesseContext context;


public FitNesseServer(FitNesseContext context) {
  this.context = context;
}

public void serve(Socket s) {
  serve(s, 10000);
}

public void serve(Socket s, long requestTimeout) {
  try {
    FitNesseExpediter sender = new FitNesseExpediter(s, context);
    sender.setRequestParsingTimeLimit(requestTimeout);
    sender.start();
  }
  catch (Exception e) {
    e.printStackTrace();
  }
 }
}
```

Error Handling

- Very big discussion and deserves complete presentation on it . I will discuss only few points , more details can be found in detailed presentation.
- Use unchecked exceptions : “The debate is over. For years Java programmers have debated over the benefits and liabilities of checked exceptions. When checked exceptions were introduced in the first version of Java, they seemed like a great idea. The signature of every method would list all of the exceptions that it could pass to its caller. Moreover, these exceptions were part of the type of the method. Your code literally wouldn’t compile if the signature didn’t match what your code could do”
- Understand when to suppress and when to propagate and when to retry is very important
- Don’t return null
- Don’t pass null
- Use exception rather than return code
- Log and throw is anti pattern
- Empty catch blocks is anti pattern

Classes & Interfaces

- Encapsulation : We like to keep our variables and utility functions private
- Keep it small and only relevant methods should be present in that class
- Favor composition over inheritance
- Don’t use constant interface . This is anti pattern.
- Single responsibility principle : Every class should focus on one and only one responsibility . We want our systems to be composed of many small classes, not a few large ones. Each small class encapsulates a single responsibility, has a single reason to change, and collaborates with a few others to achieve the desired system behaviours
- Unit test your classes
- Focus on thread safety

Conclusion

- References :
  - Effective Java : Joshua Bloch
  - Clean code : Robert C. Martin
  - Refactoring - Improving the design of existing code : Martin Fowler