Defensive Programming *Defend Your Code*

Prepared by Arun VG

About the Author :

Feel free to check my details in
http://in.linkedin.com/in/arunvg

Contents

Abstract

"Any code of your own that you haven't looked at for six or more months might as well have been written by someone else ". Meaningful, the quote points that programming is just not about finding solutions, but finding solutions that sustain. Sustain the adversity of the environment where the program is used, sustain the modifications during scaling and also as the quotes say sustain the change of programmer.

Defensive programming is an approach to improve the software and source code (the ultimate deliverable) in terms of quality, comprehensibility and sustainability for ever changing user behavior and design specifications. The approach asks the programmer to makes as few assumptions as absolutely necessary while writing/defining a program and protects his/her piece of code from unintended inputs.

The paper intends to demonstrate some basic techniques that we should practice to prevent unexpected run-time errors and also exemplify the usage of tools that would help us to ensure applying these techniques, keeping the cost of effort to minimal.

NB:
The examples in the paper are biased to JAVA as the programming language and tools are generally used in java development environments.

The document don't serve as a coding guide line or adds to the existing ones .It is a part of the existing coding guide line and just highlights some techniques that are defensive.

Defensive programming Techniques:

When do road accidents take place? Surely an accident can occur when we are not driving properly, not obeying the rules. But can obeying rules assure safe driving, off course the other drivers who share the road with us should also drive properly and the road should be in a proper condition. Both of these we cannot depend on. The only way to be safe is to be proactive and take care of all possible intricacies on the road. Similarly often for a programmer, the code he delivers is part of a larger system and need to work handling all the complexities handled by the system. Thus a programmer cannot make a narrow assumption about the change in input or the state of the system at the time which this piece of code is being used. Making a narrow assumption means meeting accidents.

Defensive programming techniques ask the programmer to makes as few assumptions as absolutely necessary while writing/defining a program and protects his/her piece of code from unintended inputs and changes in the environment where it is working. It asks the programmer to be proactive rather than reactive by just providing solution to a specified problem.

Defensive programming techniques are intended to solve problems like

- Invalid input data
- Change of environments in which the software is running
- Changes in source code for scaling the features or for bug fixing.

Invalid input Data:

In the subatomic level of a software system, the processes are done in a method/function .There fore the input to a method/function is very crucial. The construct of the method should clearly define what type of input it handles and what type of output it is going to deliver. This information is easily available in the lines of program itself provided they are written in a readable format. But for more complex method/functions it should be mandatory that the pseudo code is available as a comment itself. There by ensuring that the user using this function is very much ware of what is going to happen to the input he/she has given and also about the output.

For example

```
/**
 * Method prints the input value to the System.out
 * @param value - can be any string value
 */
public void printInput(String value){
      System.out.println(value);
}
```

The construct defines that input should be a String and the type of output is readable but still is added in the comment. There is no chance of an invalid input in run time as compiler (JAVA compiler) assures that this method can be called by passing only a String object. So in this example the post condition and preconditions of the function is satisfied.
Now check this

```
/**
 * Sets the refresh rate.
 *
 * @param  rate refresh rate, in frames per second.
 * the value should be rate <= 0 or rate > MAX_REFRESH_RATE.
 */
 public void setRefreshRate(int rate) {
     setRefreshInterval(1000/rate);
 }
```

The example shows a Setter method in a UI component .Here the comment specifies that the input rate should be value should be rate <= 0 or rate > MAX_REFRESH_RATE , but a comment does not make it mandatory. Still a user who didn't read the comment can write logic that would pass an invalid rate value. So better is make the method handle the situation.

```
/**
 * Sets the refresh rate.
 *
 * @param  rate refresh rate, in frames per second.
 * @throws IllegalArgumentException if rate <= 0 or
 *           rate > MAX_REFRESH_RATE.
 */
 public void setRefreshRate(int rate) {
     // Enforce specified precondition in public method
     if (rate <= 0 || rate > MAX_REFRESH_RATE)
         throw new IllegalArgumentException("Illegal rate: " + rate);

     setRefreshInterval(1000/rate);
 }
```

Now the updated method throws an IllegalArgumentException if the condition for the input value is not satisfied .This forces the people using this method have to handle the situation.

Consider another example

```
/**
 * Get the balance after withdrawal of the input amount
 * Balance expected to be greater than amount
 * @param balance - available balance
 * @param amount - amount to be deducted
 * @return
 */
public double getBalanceAfterWithDrawal(double balance , double amount){
        return balance - amount;

 }
```

Here there is a chance of an invalid data, balance should be greater than the amount. But the condition would have been handled by another piece of code before calling this method. Throwing an illegal argument exception would be inappropriate because we have to handle it where this method is being called. But in a production system the programmer is sure that the condition will not occur .So throwing an exception certainly means adding some code which always comes with a cost. Here come the use of **assert.**

Assertion facility is added in Java from its 1.4 versions .It can be used to validate the assumptions made about the state of the program at designated locations of the code. The most importance thing is that we can switch on or off this facility, there by in development environment we can switch it ON and in a production we can left it to be the default i.e. OFF.

So consider the above example

```
/**
 * Get the balance after withdrawal of the input amount
 * balance expected to be greater than amount
 * @param balance - available balance
 * @param amount - amount to be deducted
 * @return
 */
public double getBalanceAfterWithDrawal(double balance , double amount){
        assert balance >= amount;
        return balance - amount;

 }
```

Simple, as it looks it doesn't contain any hidden charges. So here using the assert keyword we have given a Boolean condition whereby if `balance >= amount` it returns true, `otherwise if` false and the Assert facility is switched ON; AssertionError will be thrown at runtime. So in the development environment where there is potential threat, the issue is at least notified and in Production as the assert facility is turned off there is no cost incurred.

Assert facility by default is turned off . The argument -eanbleassertion or -ea will enables assertion, while -disableassertion or -da will disable assertions at runtime.
The following command will run AssertionDemo with assertion enabled.
java -ea AssertionDemo
or
java -enableassertion AssertionDemo

The above mentioned approaches can be used to define the Precondition and Post-conditions of a method.
Advantages :

- Prevention of run time errors
- Effective documentation
- Scalability

At a method level other than invalid input data there are a lot common loop holes like

- Numeric overflow
- Infinite loops
- Null pointer exception , etc...

External System dependency:

A software system often is not a standalone piece of software. It would be part of a larger system serving a business solution. The software system we have would be communicating with other systems as well. At this level also there can be issues not only with invalid data's but also availability of other systems, the response time etc…

A common problem we face on a day to day basis is while communicating with a database. Communicating with other systems always come with problems like communication failures. The systems which are using the service should gracefully exit even if it cannot communicate with databases.
In case of a database and SQL Exception is thrown on any communication failures. (An SQL exception is also thrown when the input data to the DB is invalid). Usually the strategy can be logging the error and throw it as a custom runtime exception, handling it showing a message and a graceful exit.

For example :

```
/**
 * Checks whether the user is an active user in the system.
 * if present return true
 *
 * @param user_id - internal id for the user
 * @param dbCon  - a valid connection object
 * @return
 */
public static boolean isUserPresent(String user_id, Connection dbCon){

	String query =  getCustomerPresentQuery();

	PreparedStatement pStmtQuery = null;
	ResultSet rsQuery = null;
	try {
		pStmtQuery = dbCon.prepareStatement(query);
		pStmtQuery.setString(1, user_id);

		rsQuery = pStmtQuery.executeQuery();

		if(rsQuery.next()){
			return true;
		}

	} catch (SQLException e) {
		MyLogger.getInstance().fatal(e);// Log the error using a Logger
                                                utility
		throw new MyInternalException(e);// Throw the exception as a
                                                 custom defined run time
                                                 exception

	}finally{
		DBUtils.getInstance().closeResource(rsQuery);
		DBUtils.getInstance().closeResource(pStmtQuery);
	}
	return false;
}
```

The example is limited to JDBC but can be extended to any communication. The approach for failure cases need to be finalized in the design itself.

Change in source code:

Any code written is not written for ever, it has to be changed either for scaling or in adverse conditions for fixes. Another intricacy is the person who is changing the code may not be the one who is changing it. If the system is complex and if the piece of code is reused in many situations, the impact may be larger enough for the person to perceive the impact. Thus changing the code adds to the probability of bugs and large amount of testing is required.

In these situations where a piece of code is used frequently (more like an API) it is better to write automated test cases. In java the most commonly used test case framework is JUNIT .There is definitely a cost incurred but the advantage comes when there is a change, we can get the impact just by running the test suite for the piece of code.

For example :

```
/**
 * Add value 2 with value 1
 * @param value1
 * @param value2
 * @return
 */
public double add (double value1,double value2){
      return value1 + value2;
}
```

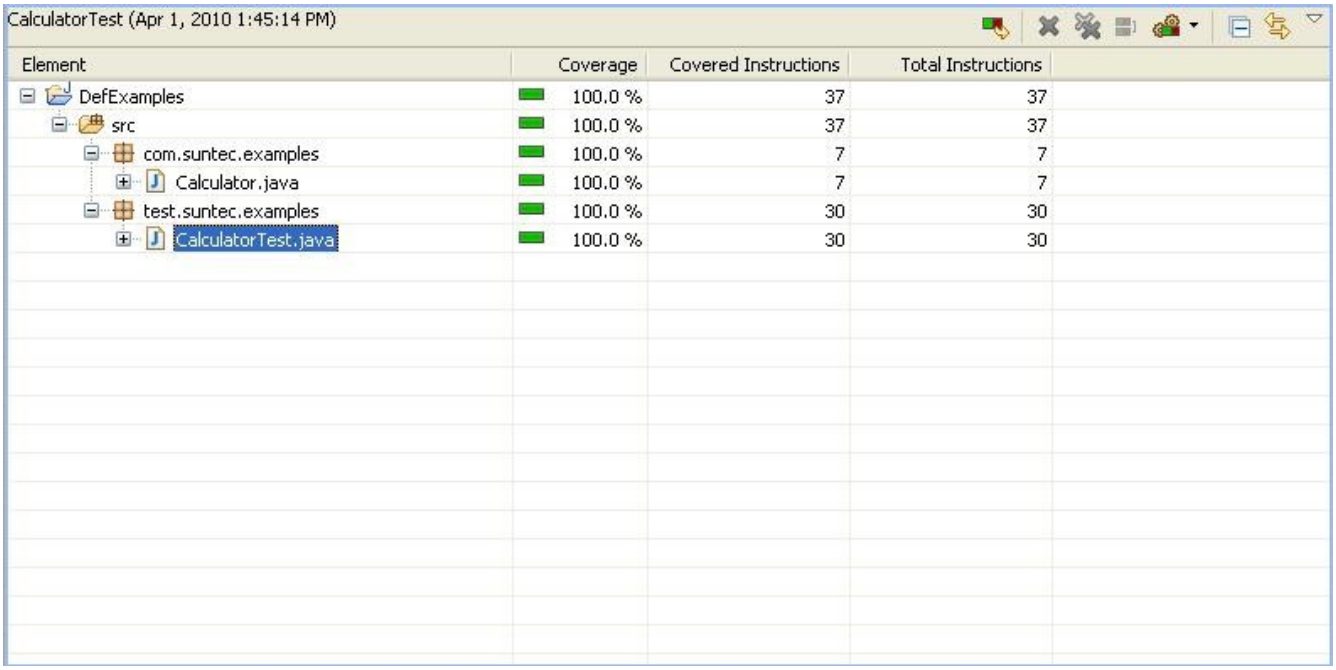
This method just add two values we can use junit to write testcases

```
public class CalculatorTest extends TestCase {

      public CalculatorTest(String  name) {
            super(name);
      }
      /**
       * Test Case 1
       */
      public void test1AddValues(){
            Calculator calculator = new Calculator();
            assertEquals(2.0, calculator.add(1, 1));
      }
      /**
       * Test Case 2
       */
      public void test2AddValues(){
            Calculator calculator = new Calculator();
            assertEquals(1.3, calculator.add(1.1, 0.2));
      }
}
```

The junit coverage plug-ins are available and can be used to view the code coverage .

Scree shot for Calculator



When there is a change in the code, the programmer can run the test cases to see whether the current features are not affected and also can improve the test cases for the feature he/she is adding.

Conclusion :

Due to stringent time lines and aggressive development methods like Agile processes , the source code is no longer static .It is subjected to vast changes both inwardly and also from external sources. There fore the practice of Defensive programming techniques is the need of the hour. Any team need to practice these techniques and be proactive towards being defensive.

The word “Programmer” is losing its shine, look forward to the word Developer .We are always developing a system and coding a program is just a part of it.

Happy programming , err Happy Developing :-)

References :

- **Wikipedia :**http://en.wikipedia.org/wiki/Defensive_programming
- Code Complete 2nd Edition - Chapter 8
- Googled results of “Defensive + programming + techniques”