CAB302 Software Development
Assignment 2: Test-Driven Development and Graphical User Interface Programming
Submission Guide
Semester 1, 2016

Due date: Thursday, June 2, 2016 (late in Week 13)
Weighting: 35%. Part 1: 17.5% Part 2: 17.5%
Assessment type: Group assignment, working in pairs
Version: 1.0 Wednesday, June 1, 11:27PM

Introduction

The overall submission process is essentially the same as for assignment 1 – using the submission link on Blackboard and uploading a zip file to be called `asgn2.zip`. Details of this process are laid out below.

The technical requirements for the project have been laid out near exhaustively in the documents in the Assignment 2 directory on the BB site. This document is to provide you with a final checklist on structure, and guidance on some issues of housekeeping and coding style. These are considered in turn below.

Source Code Quality: Some guidelines on acceptable Java code appear at the end of this document, and these are generally consistent with the requirements of the earlier assignment.

Statement by the Team: It is a mandatory requirement of this assignment that you prepare a short (***definitely*** well under 1 page) ***TEXT*** document called `statement.txt` containing the following information:

- The names and student numbers of the team members
- A summary of the contribution by each team member
- A statement that the summary is agreed by both students
- A statement of completeness, including any known bugs which have not been resolved. In particular, you should state whether or not you have attempted the charting version of the GUI, or just left it at the `JTextArea` version.
- Anything else you feel that we need to know in order to grade your assignment appropriately.

Note that the statement file must be in basic text format, and called `statement.txt`. It must ***not*** be in Word, Open Office or PDF format. Please do not under any circumstances use this approach to apply for an extension or to offer medical or personal reasons for a delayed or incomplete submission. Applications for special consideration and assignment extensions *must* be made centrally.
Please see:
https://www.student.qut.edu.au/studying/assessment/late-assignments-and-extensions

Git or other Repo Log File: It is mandatory that you supply us with evidence that you have undertaken some sort of consistent version management. Most people will use git, but whatever, we require that you submit a log file – text *only* - to be called `repo.log`, and included with the others as shown in figure 2 below. We need only a snapshot history. We don't want to see everything at once. I recommend the following commands for git, to be executed at the bash command line prompt. The first command is to set an alias to allow you to generate a quick summary again and again. The remaining commands generate the file for submission.

First we set up a global alias (at the git bash command line):

```
$ git  config --global alias.pretty "log --pretty=format:'%h : %s' --graph"
```

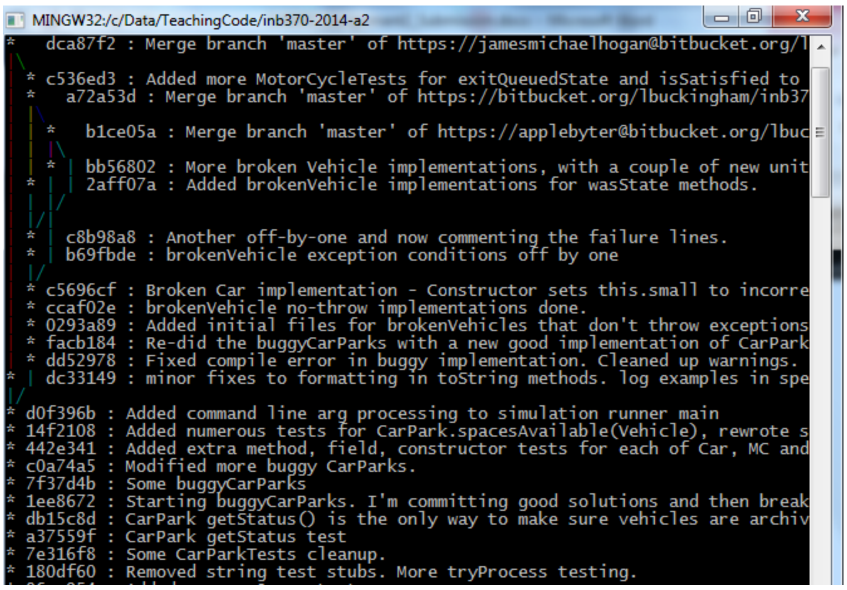
Then we use it to generate the summary log:

```
$ git pretty > repo.log
```

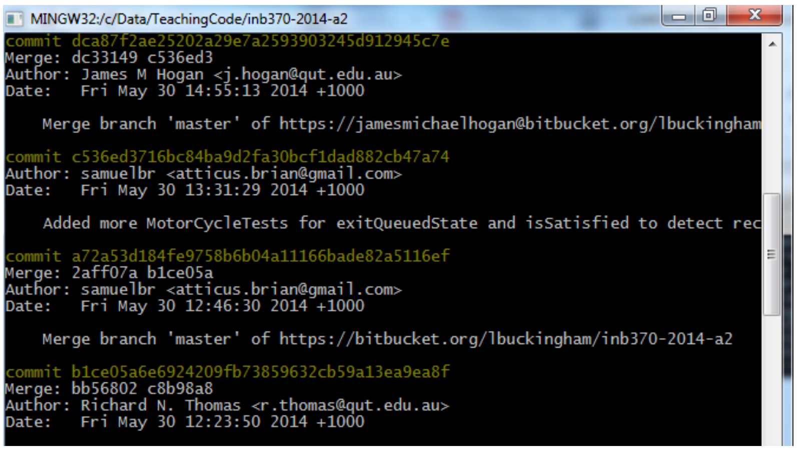
Before finally appending a complete log which we can look at as needed:

```
$ git log >> repo.log
```

Note that we will usually just look at the summary. The full log with all the dates is there for checking, and if we don't like that, we will ask to inspect your repo. The log file should be placed in the top directory of the project along with the `statement.txt` file and, if appropriate, the `tests.pdf` file. Please check that your output begins with a summary form, and then has a corresponding detailed history. An example is shown here for an older piece of code:



```
*   dca87f2 : Merge branch 'master' of https://jamesmichaelhogan@bitbucket.org/l
|\
| * c536ed3 : Added more MotorCycleTests for exitQueuedState and isSatisfied to
| *   a72a53d : Merge branch 'master' of https://bitbucket.org/lbuckingham/inb37
| |\
| | *   b1ce05a : Merge branch 'master' of https://applebyter@bitbucket.org/lbuc
| | |\
| | * | bb56802 : More broken Vehicle implementations, with a couple of new unit
| * | | 2aff07a : Added brokenVehicle implementations for wasState methods.
| | |/
| |/|
| * | c8b98a8 : Another off-by-one and now commenting the failure lines.
| * | b69fbde : brokenVehicle exception conditions off by one
| |/
| * c5696cf : Broken Car implementation - Constructor sets this.small to incorre
| * ccaf02e : brokenVehicle no-throw implementations done.
| * 0293a89 : Added initial files for brokenVehicles that don't throw exceptions
| * facb184 : Re-did the buggyCarParks with a new good implementation of CarPark
| * dd52978 : Fixed compile error in buggy implementation. Cleaned up warnings.
* | dc33149 : minor fixes to formatting in toString methods. log examples in spe
|/
* d0f396b : Added command line arg processing to simulation runner main
* 14f2108 : Added numerous tests for CarPark.spacesAvailable(Vehicle), rewrote s
* 442e341 : Added extra method, field, constructor tests for each of Car, MC and
* c0a74a5 : Modified more buggy CarParks.
* 7f37d4b : Some buggyCarParks
* 1ee8672 : Starting buggyCarParks. I'm committing good solutions and then break
* db15c8d : CarPark getStatus() is the only way to make sure vehicles are archiv
* a37559f : CarPark getStatus test
* 7e316f8 : Some CarParkTests cleanup.
* 180df60 : Removed string test stubs. More tryProcess testing.
```



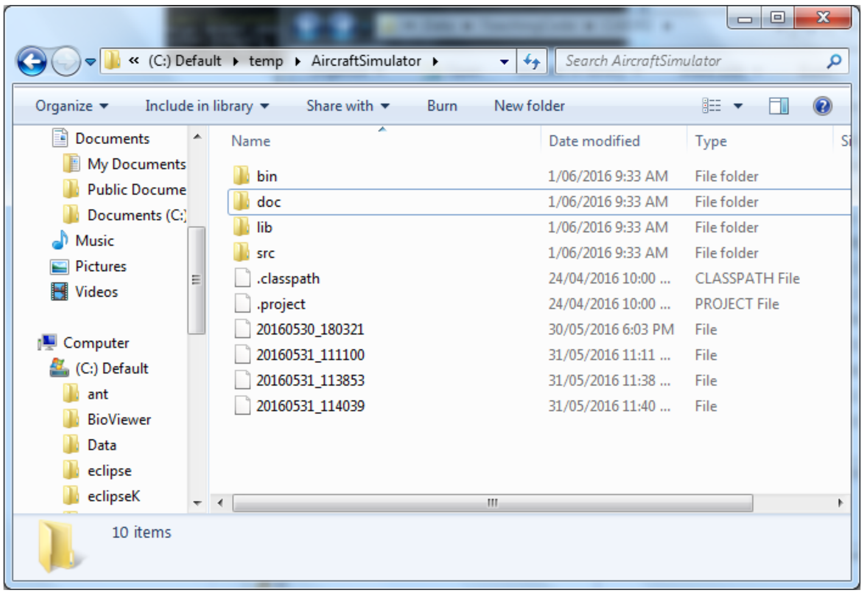
```
commit dca87f2ae25202a29e7a2593903245d912945c7e
Merge: dc33149 c536ed3
Author: James M Hogan <j.hogan@qut.edu.au>
Date:   Fri May 30 14:55:13 2014 +1000

    Merge branch 'master' of https://jamesmichaelhogan@bitbucket.org/lbuckingham

commit c536ed3716bc84ba9d2fa30bcf1dad882cb47a74
Author: samuelbr <atticus.brian@gmail.com>
Date:   Fri May 30 13:31:29 2014 +1000

    Added more MotorCycleTests for exitQueuedState and isSatisfied to detect rec

commit a72a53d184fe9758b6b04a11166bade82a5116ef
Merge: 2aff07a b1ce05a
Author: samuelbr <atticus.brian@gmail.com>
Date:   Fri May 30 12:46:30 2014 +1000

    Merge branch 'master' of https://bitbucket.org/lbuckingham/inb370-2014-a2

commit b1ce05a6e6924209fb73859632cb59a13ea9ea8f
Merge: bb56802 c8b98a8
Author: Richard N. Thomas <r.thomas@qut.edu.au>
Date:   Fri May 30 12:23:50 2014 +1000
```

GUI Test Results: Your GUI tests need to be presented using the template previously uploaded to Blackboard. This is given to you as a pptx and as a pdf, with the idea that you will choose the one appropriate to your operating system. Our markers will work on a mix of linux and windows, so we ask that you export even the powerpoint as a pdf. The file should be called `GUITests.pdf`.

Management of submission files: In previous years we have used exports from eclipse to assemble the archive, but I have found OS level instructions lead to far fewer errors at a package level, and it is much easier to handle additional files at the OS level rather than in eclipse.

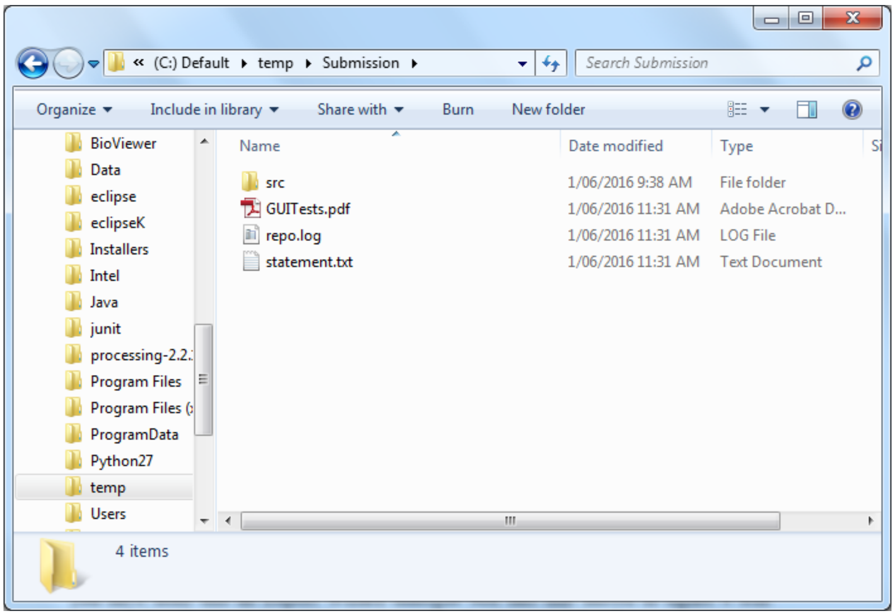
Creating the Project Archive

Your entire submission will be contained in a single zip archive. Your solution to the assignment should follow the project structure in `AircraftSimulator`, which contains the Java packages shown below. We will capture the entire src directory and then add the files at the top level. Begin by navigating to the eclipse workspace using the operating system file manager:



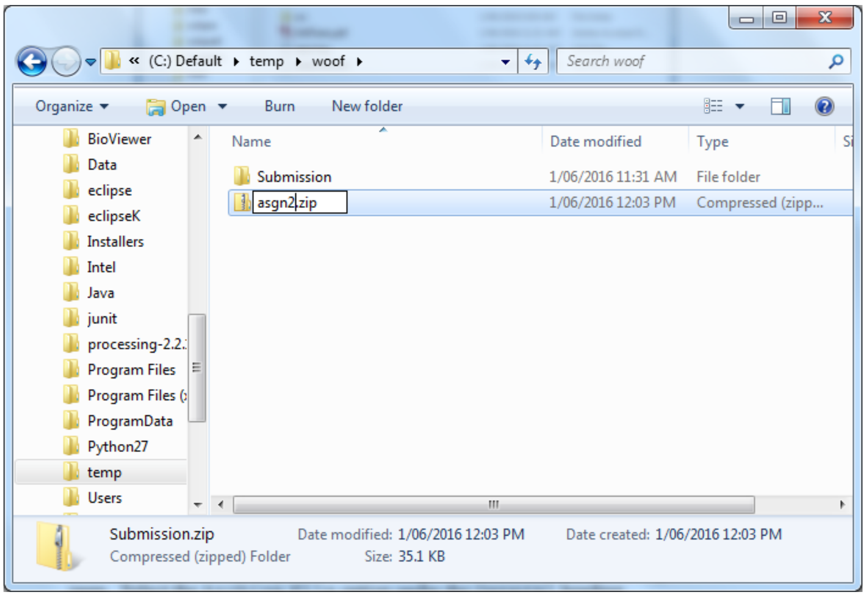
Name	Date modified	Type
bin	1/06/2016 9:33 AM	File folder
doc	1/06/2016 9:33 AM	File folder
lib	1/06/2016 9:33 AM	File folder
src	1/06/2016 9:33 AM	File folder
.classpath	24/04/2016 10:00 ...	CLASSPATH File
.project	24/04/2016 10:00 ...	PROJECT File
20160530_180321	30/05/2016 6:03 PM	File
20160531_111100	31/05/2016 11:11 ...	File
20160531_113853	31/05/2016 11:38 ...	File
20160531_114039	31/05/2016 11:40 ...	File

Here the folder includes a few recent log files. We care only about the src directory. Select it and copy it in its entirety to a temporary directory as shown – you may call the temporary directory what you like. Add in the `GUITests.pdf`, `repo.log` and `statement.txt` files as shown.



Name	Date modified	Type
src	1/06/2016 9:38 AM	File folder
GUITests.pdf	1/06/2016 11:31 AM	Adobe Acrobat D...
repo.log	1/06/2016 11:31 AM	LOG File
statement.txt	1/06/2016 11:31 AM	Text Document

Step up one level and create a zip file, renaming as shown to `asgn2.zip`. This is your final submission for upload to Blackboard.

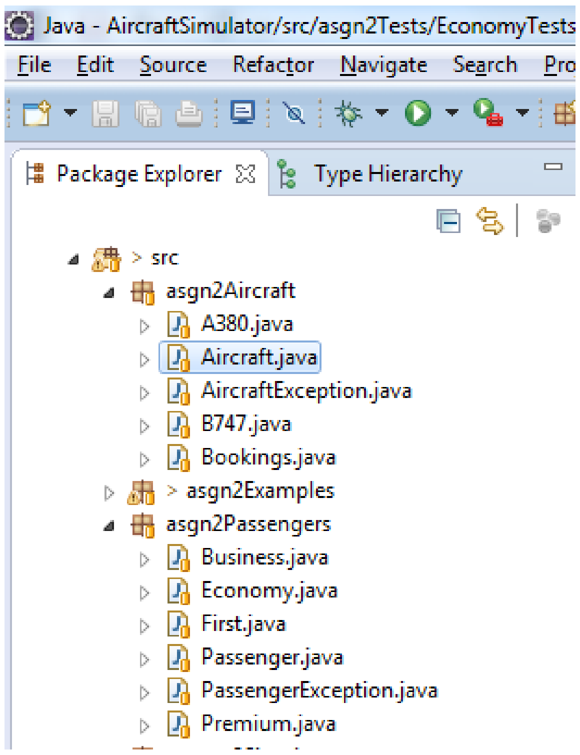


Name	Date modified	Type
Submission	1/06/2016 11:31 AM	File folder
asgn2.zip	1/06/2016 12:03 PM	Compressed (zipp...

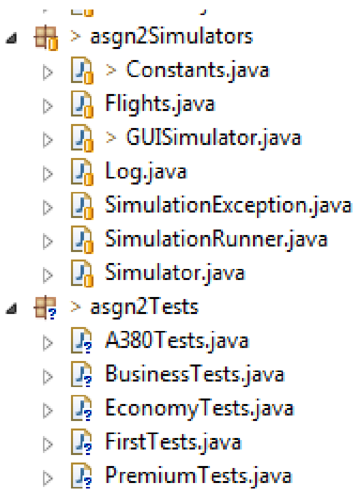
Submission.zip Date modified: 1/06/2016 12:03 PM Date created: 1/06/2016 12:03 PM
Compressed (zipped) Folder Size: 35.1 KB

Appendix: Files Required

The screenshots below from eclipse show the packages and the files that should be in them. You may have some additional files in the `asgn2Simulators` package as discussed. Some of you may also have included additional files in `asgn2Tests` – please make sure you follow the standard naming conventions. Don't worry about the examples package – we will ignore it, but it makes instructions so much easier to just have you copy the entire source directory:


Java - AircraftSimulator/src/asgn2Tests/EconomyTests
File Edit Source Refactor Navigate Search Pro
Package Explorer
Type Hierarchy

```
> src
  asgn2Aircraft
    A380.java
    Aircraft.java
    AircraftException.java
    B747.java
    Bookings.java
  > asgn2Examples
  asgn2Passengers
    Business.java
    Economy.java
    First.java
    Passenger.java
    PassengerException.java
    Premium.java
```



```
  > asgn2Simulators
    > Constants.java
    Flights.java
    > GUISimulator.java
    Log.java
    SimulationException.java
    SimulationRunner.java
    Simulator.java
  > asgn2Tests
    A380Tests.java
    BusinessTests.java
    EconomyTests.java
    FirstTests.java
    PremiumTests.java
```

Guidelines and Checklist for Java Source Files

Introduction: SUN provided comprehensive – indeed *inaccessibly* comprehensive – guidelines for source code written in the Java language. These are not now maintained by Oracle, but an archive version is available, and various people have produced their own take on what is important and what isn't. This is our current take on the matter.

Some of these issues are more important than others. We will not be using the lists that follow in any direct way to assign you a mark for code quality. Rather, the two are correlated – if you deal with ***most*** of the issues discussed below, you will have very good code which will get you a good mark. So don't stress if you haven't covered absolutely everything. And whilst we try to keep to the best practice we can, we sometimes aren't perfect either – we have been lax on the file context, for example.

The easiest approach to ensuring a coding standard is to use an example of high quality source code as a model, and to offer some checklist guidelines highlighting the important issues.

```java
/*
 * This file forms part of the CargoSystem Project
 * Assignment Two - CAB302 2015
 *
 */
```

Simple file comment showing context

```java
package asgn2Exception;
```

`package`, followed by import statements as needed

```java
/**
 * This class represents any exceptions thrown by any of the freight
 * container or cargo manifest classes.
 *
 * @author CAB302
 * @version 1.0
 */
public abstract class CargoException extends Exception {
```

Class header comment; use of javadoc for author and version

Consistent indentation – usually a fixed number of spaces. NO tabs…

Concise method comment; both classes and methods open braces on the same line as the definition. Closing brace is aligned with the method or class name

```java
  /**
    * Constructs a new CargoException object with a helpful message
    * describing the problem. (The message can be retrieved from
    * the exception object via the <code>getMessage</code> method inherited
    * from class <code>Throwable</code>.)
    *
    * @param message an informative message describing the problem
    */
  public CargoException(String message) {
        super("CargoException: " + message);
  }

}
```

Javadoc for parameters, return values. Separate methods by using blank lines.

The next example is drawn from an older project, and shows the structure of loops and conditionals. The use of PRE and POST conditions is not mandatory, but is helpful in technically complex code. Similarly, javadoc is helpful for private methods for internal use, but you need to use a command line switch to generate it. For the assignment, it is compulsory for public methods, optional for private methods. The data structure is a retrieval Trie, designed for storing and accessing substrings.

```
   /**
     * Auxiliary method to locate base in sibling list.
     * PRE: tn != null, key initialised
     * POST: return=true AND tn.key=key<br>
     *    OR return=false AND (tn.fs.key > key OR tn.fs=null)
     * @param tn - treenode ref to start of list
     * @param key - search key
     * @return sibling node - node corresponding to key or
     */
  private NodeReturn findSibling(TrieNode tn,char key) {
      if (tn.key==key) {
         //special case - first node holds key
         return new NodeReturn(true,tn);
      } else if (tn.key>key) {
         //special case - first node holds larger key; need new lmc
         return new NodeReturn(false,true,tn);
      } else {
         //traverse the list
         while ((tn.fs!=null) && (tn.fs.key <= key)) {
            tn=tn.fs;
         }

         //null or found it
         if (tn.key < key) {
            return new NodeReturn(false,tn);
         } else {
            return new NodeReturn(true,tn);
         }
      }
  }
```

CamelCasing for classnames; mixedCase (CamelCasing but lowercase first letter) for identifiers.

Light commenting. Let the code tell the story as far as possible. Only use comments like these when there is something to say.

Note alignment of conditionals. Use block structure even for simple ifs:

```
if (condition) {
   do something;
}
```

Don’t use:

```
if (condition)
   do something;
```

Java Source Language Code Checklist

Adapted and extended by JMH, largely from SUN guidelines. Other OOP checklists have been consulted.

1. **Source file contains a single class or interface – excepting inner or anonymous classes for event handling. We use class to denote both class and interface in what follows.**
2. **File begins with a context comment indicating project and environment**
3. `package` **statement is first processed statement.**
4. `import` **statements follow after two lines of whitespace**
5. **Use wildcard imports if more than two imports are drawn from the same package**
6. `class` **declaration should be preceded by a header comment outlining its purpose; use javadoc attributes for author and revisions**
7. **Header comments are intended to convey implementation-free descriptions of the class and methods – remember that these may appear in javadoc.**
8. **Field and static declarations at the head of the class, prior to the constructor.**
9. **Initialisation of reference fields usually to be performed in the constructor.**
10. `public` **methods to follow constructor(s)**
11. `package, protected` **and** `private` **methods to follow** `public` **methods in that order.**
12. **Single statement on each line – excepting perhaps for initialisation**
13. **Sparing use of local comments within methods.**
14. **Local variables to be declared immediately following the method declaration (i.e. on the next line) and to be followed by a blank line**
15. **Use blank line after defined structures such loops and conditionals to aid clarity (exception to this rule at end of method, closing lots of braces).**
16. **Use blank line(s) to separate logically distinct structures. Ideally refactor to incorporate into separate methods.**
17. **Use single blank character to follow commas in argument lists and either side of binary operators such as '+"**
18. **Handle all checked exceptions arising from library methods called, and handle** ***all exceptions knowingly thrown (i.e. using the*** *`throw`* ***statement)*** **by the present code.**

Example use of standard structures:

```
     try {
         BufferedReader br=new BufferedReader(new FileReader(fName));
         return br;
     } catch (FileNotFoundException e) {
         bioviewer.util.Error.FatalError(e);
         return null; //for the compiler only
     }
```

Correct form – note the alignment of braces.

```
public class CargoException extends Exception
{

      /**
       * Constructs a CargoException without a detail
       * message.
       */
      public CargoException()
      {
            super();
      }

      …

}
```

Alternative format for placement of braces – opening brace on new line. Common among C++ and some C# programmers. Don't use this style.