Lab 8: Implementation
All Deliverables, unless otherwise noted, due by 11:59 PM on the dates given

1. Key Lab Activities

- Finalize your design
- Auto generate source code stubs from your design in Java (if you want to do so)
- Implement your project in Java
- Perform a code review of a segment of completed Java source code

2. Deliverables and due dates

Deliverable	Responsible Party	Date
Code which is to be code reviewed is delivered to reviewers / instructor via email	Quality Manager	May 6, 2013 23:59
Preliminary source code for the robot controller shall be archived into SVN and labeled as a preliminary demo unit.	Development Manager	May 8/9, 2013
Preliminary demonstration of basic robot functionality using the system	Team Leader	May 9/10, 2013
Summary of the implementation and the code review exercise	Each Team Member	May 10, 2013

3. Introduction

In these lab sessions, you will implement your Lego Mindstorms control system. You will be responsible for translating your design into Java source code.

4. Detailed Steps

4.1. Design Finalization

Before proceeding, take one last look at your design. Is it complete? Is it correct? Are there any deficiencies? Will it work (or so you hope.) Once you can answer these questions, you can generate source code from your design[1]. Your source code should be in a package edu.msoe.se2890.mindstormproject.<yourteamname>.[2]

[1] If you prefer, you can generate all source code manually, but it is not necessary.
[2] Hint: You may want to check your design packaging in EA before exporting your code to make certain your design is configured this way. This may involve moving code into certain packages or otherwise making slight modifications to the model layout.

4.2. Planning

Now, more than ever, it is important to plan your development. Each class that is required b y the system must be implemented by one or more developers. To keep easier track of your status, you might try using the TODO list of the tracker, as working independently and keeping track of what you do is more important now than ever. In your workplan, map each class to a developer, and estimate how long it will take to implement that module. This information should be entered into the workplan by the process manager. During this week, as classes are finished, the completed classes should be marked off by the process manager as being completed, and the workplan should be updated showing the actual effort as well as the completed status of the module.

4.2.1. Implementation Hints

While the exact sequence of implementation should be decided upon by the team, it is recommended that the first classes developed involve the core functionality of the vehicle, namely steering, forward and backward direction control, the GUI, and ignition control.

Being that you will be developing a lot of software in parallel, it may be advantageous to download and install a file difference engine, such as the Beyond Compare tool from Scooter software (http://www.scootersoftware.com/).

4.3. Implementation

At this point, start implementing your project. You may have already started with the UI, but if not, now is your time to shine. Your code must not only be functional, but must also follow proper commenting standards. This includes both overview comments, which describe the purpose for the file, method comments (aka JavaDoc, which describe the parameters for methods, as well as flow comments which describe the flow through a method. A standard, based upon the standard provided by Dr. Taylor, is included in this document.

4.4. Exporting a Project from Enterprise Architect

Enterprise Architect is fully capable of round trip engineering. With round trip engineering, one can automatically create source code from a design model, and as the model develops, one can update the model from the existing source code base.

To export the model, start by opening a source code model. The example shown in this tutorial is available on the course website.

To generate source code, start by selecting the package, code engineering, and generate source code.

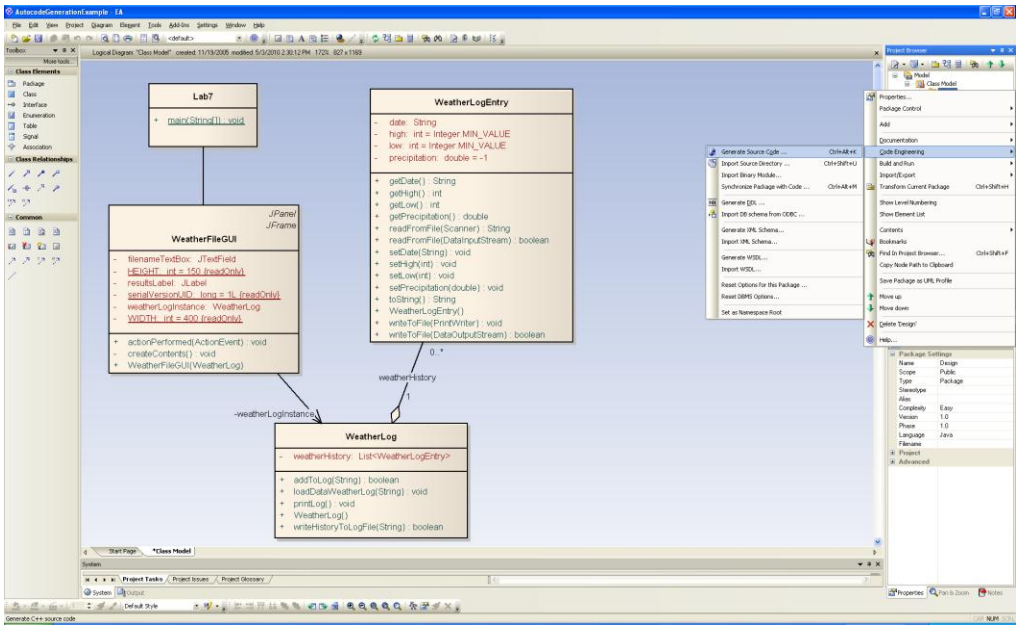

Figure 1: Sample UML Diagram.

Once this has been done, select the output directory and the files that you wish to generate, as is shown below.

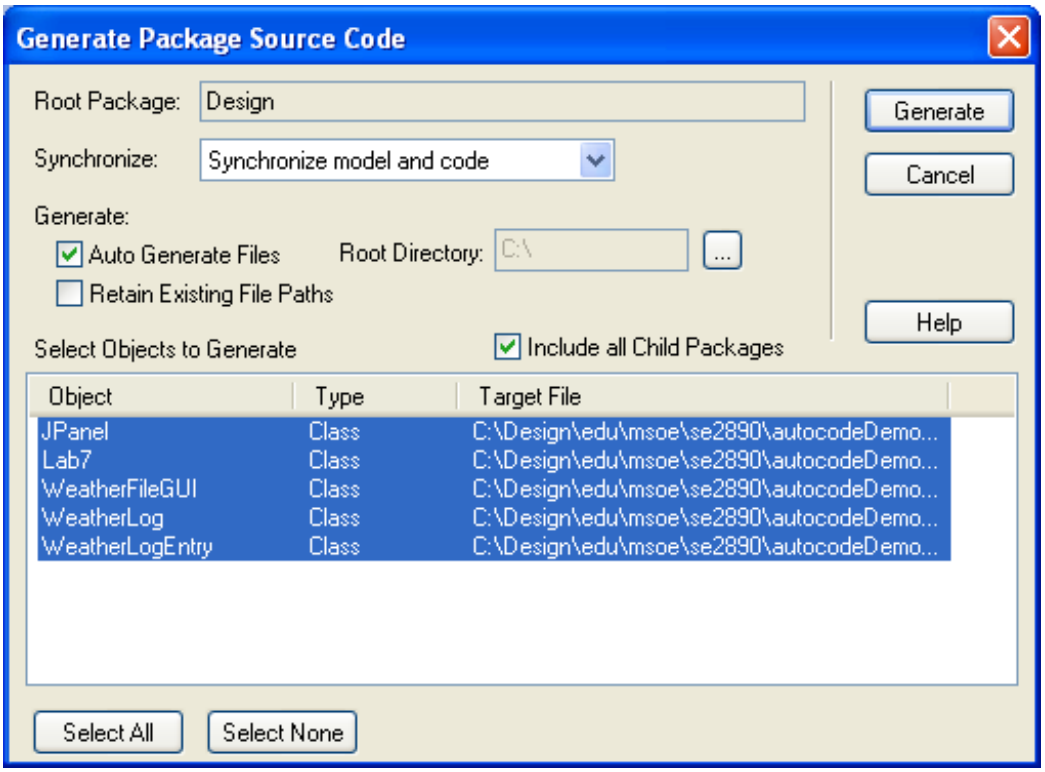

Figure 2: UML Autocode generation Panel.

Clicking on generate will then generate skeleton code for the given class.

Auto generation of code is not necessary. You may find it easier to simply implement the code from scratch. However, it is available as an option that may speed the generation of your project.

```
package Design;

/**
 * @author schilling
 * @version 1.0
 * @created 03-May-2010 2:31:10 PM
 */
public class WeatherLog {

          private List<WeatherLogEntry> weatherHistory;
          public WeatherLogEntry m_WeatherLogEntry;



          public void finalize() throws Throwable {

          }

          /**
           * This is a simple constructor which will just instantiate  ArrayList of
           * WeatherLogEntries.
           */
          public WeatherLog(){

          }

          /**
           * This method will add the contents of the file to the log. The file is a text
           * file which has delimited weather entries stored in text format.
           * @return The return will be true if the file is successfully added. False if an
           * error occurs.
           *
           * @param filename    This is the name of the file that is to be used.
           * @exception FileNotFoundException If the file can not be found this exception
           * will be thrown.
           */
          public boolean addToLog(String filename)
            throws FileNotFoundException{
                    return false;
          }

          /**
           * This method will load data from the given file.
           *
           * @param previousHistory    This is the filename for the binary file which stores
           * the previous history.
           * @exception FileNotFoundException This exception will be thrown if the given
           * filename does not exist.
           */
          public void loadDataWeatherLog(String previousHistory)
            throws FileNotFoundException{

          }

          /**
           * This method will print out to the console all entries within the weather log.
           */
          public void printLog(){

          }

          /**
           * This method will create a binary file representing the history log stored
           * within this file. First it will write the total number of WeatherLogEntry
           * instances stored within the array list. When that has been written, it will
           * iterate through the list of files, storing them all in succession one at a time.
           *
           * @return The return will be true if the write is successful and false otherwise.
           *
           * @param filename    This is the filename that is to be used for the binary file.
           * @exception FileNotFoundException This exception will occur if there is a
           * problem writing to the named file.
           */
          public boolean writeHistoryToLogFile(String filename)
            throws FileNotFoundException{
                    return false;
          }
}
```

Figure 3: Autogenerated Source Code

5. Code Review

During class on May 8, 2013, you are responsible for performing a code review on a segment of your implemented source code. In essence, the person who implemented the code is responsible for going with the quality manager and finding 2 – 3 other students in class who will review their Java code. All people who are not quality managers are responsible for participating in a code review of someone else's code. A google doc will be made available to indicate who you wish to have review your code and what the description of the code is that will be reviewed.

As issues are found, it is the responsibility of the quality manager to record them in GForge in order to ensure that they are tracked to resolution.

A checklist is available on the course website of areas to check.

6. Deliverables:

1. The instructor should be notified of which code is going to be reviewed and who the reviewers will be for the code.
2. The code which is to be code reviewed shall be e-mailed to all reviewers in order that they can read through the code prior to the code review in lab.
3. Each person in the lab shall submit, via e-mail, a two to three sentence summary of the lab experience. This summary shall include things that went well, things that went wrong, and things that can be improved. This is very informal, but must be submitted by each person. Additionally, if a person who is reviewing the item in the review is not at tentative to the process, that should be noted by the person leading the review and quality manager's write-ups.
4. Source code for the Robot controller shall be archived in SVN. (Note: The code does not have to be fully functioning by this date, as week #9 will deal with testing of the code and further implementation. However, the code that is committed to the repository shall compile without warning.)
5. Updated workplan: The planning / process manager shall upload a current workplan indicating the effort put into the project and specific tasks performed during the week.

7. Coming Attractions (Trailers…)

During week #9, you will be testing your design and implementation. While it is advantageous to have some level of functionality before then, we will develop formal approaches to testing during that lab.

However, if you discover significant defects in the code prior to starting testing, it is definitely advantageous to log them in the defect tracking system in order that they not be overlooked.

8. Source Code Commenting Requirements

8.1. Packaging

All Java source code sub mitted shall be packaged according to the following schema:

edu.msoe.se2890.rccar.<yourteamname>.

Where yourteamname represents the name of the developing team. This CAN NOT include spaces or other control characters.

8.2. Documentation

All source code submitted shall meet the minimum documentation standards outlined below.

- The beginning of each source file should contain:
  - The name of the file/class.
  - The date it was originally written.
  - The original author.
  - A modification log (when appropriate) which describes:
    - What modifications were made.
    - When the modifications were made.
    - Who made the modifications.
- All Java classes must have a "class description" comment in javadoc style with **`@version`** and **`@author`** tags.
- All classes shall also indicate the team which created the file within the comment block.
- All fields must be declared individually and have an associated javadoc comment.
- All methods must be proceeded by a javadoc comment that makes appropriate use of the following tags:
  - @param - Parameter listings
  - @return - return value description
  - @throws - Exceptions thrown and conditions under which they will be thrown
- Within each method, documentation should be provided for any code whose purpose is not immediately obvious to someone with your current level of programming knowledge.

9. Example Documentation

```
/*
 * Course: SE2890
 * Section: 1
 * Term: Spring 2009
 * Author:
 * Team Wonderful
 * Assignment: Lab 1
 * Date: 04/09/08
 */

package edu.msoe.se2890.rccar.team_wonderful;

import java.blah.blahdy.BlahBlah;

/**
 * Class description goes here.
 *
 * @version    1.82 18 Mar 1999
```

```
 * @author     Firstname Lastname
 */
public class Blah extends SomeClass {
    /* A class implementation comment can go here. */


    /** classVar1 documentation comment */
    public static int classVar1;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static Object classVar2;

    /** instanceVar1 documentation comment */
    public Object instanceVar1;

    /** instanceVar2 documentation comment */
    protected int instanceVar2;

    /** instanceVar3 documentation comment */
    private Object[] instanceVar3;

    /**
     * ...constructor Blah documentation comment...
     * @author
     */
    public Blah() {
        // ...implementation goes here...
    }

    /**
     * ...method doSomething documentation comment...
     * @author
     */
    public void doSomething() {
        // ...implementation goes here...
    }

    /**
     * ...method doSomethingElse documentation comment...
     * @author
     * @param someParam description
     * @return Returns the number of widgets
     */
    public int doSomethingElse(Object someParam) {
        // ...implementation goes here...
        return i;
    }
}
```

Appendix A: Preliminary Grading Rubric

	Weight Factor	Rubric Score	
			Process Management - Code reviewers assigned. - Code to be reviewed denoted to the instructor. - Party responsible for developing code that will be reviewed aware of commitment. - Code distributed to reviewers in a timely fashion.
	1	4	Code - Code implementation generally matches design submitted previous week. - Code follows proper Java naming conventions - Code compiles cleanly without errors or warnings from the SVN archive - Code packages into meaningful packages based upon subsystem(s) - Appropriate scope applied to all variables and methods.
	2	1	Comments - All classes have header with name, date, original author, and a modification log describing changes - All classes have javaDoc comments at class level which explains the purpose for the class. - javaDoc comments include team name. - All methods have javaDoc comment that makes appropriate use of the following tags: - o @param - Parameter listings - o @return - return value description - o @throws - Exceptions thrown and conditions under which they will be thrown - Complex algorithms or other behavioral aspects of the program have individual comment lines as necessary.
	2	4	Work plan - Work plan committed into archive at least twice, once during the initial planning and once at the end of the lab session with actual effort amounts recorded. - Work plan shows roughly equal contributions to project (at least during estimation) - Workplan shows approximately even distribution of work. - Workplan shows work being done by all team members.
	1	2	SVN Repository - SVN repository shows commits being made by all team members. - SVN repository comments are meaningful and correct. - Individual commits have meaningful comments explaining what was changed and why.
	1	4	Individual Participation

	Weight Factor	**Rubric Score**	
			- Individual Participation in lab appears approximately correct. - Individual participation logged on workplan, both in terms of planned effort and actual effort. - Code indicates development by team member, either through code comments, svn checkin, or other mechanism.