HW2: Robot Grid

CSE1102 Spring 2015
Jeffrey A. Meunier
University of Connecticut

1. Introduction

In this assignment you will write a Java program to control a robot moving through a simple obstacle course. This demonstrates many of Java's features, but the code that you need to write is not difficult.

2. Value

This program is worth a maximum of 20 points. See the grading rubric at the end for a breakdown of the values of different parts of this project.

3. Due Date

This project is due by midnight at the end of the day on **Sunday, February 8, 2015**. A penalty of 20% per day will be deducted from your grade, starting at 12:00:01am. See this link for additional information.

4. Objectives

The objectives of this assignment are:

- To use Eclipse to develop a Java program.
- To use an external jar file to provide extra features.
- Learning Java's assignment statements and while loops.
- Using method calls on object instances.
- Seeing lots of other stuff you don't need to understand just yet.

5. Background

Be sure to read finish reading chapter 1 in the book before starting this project. Pay close attention to the following sections:

- How to work with numeric variables
- How to code assignment statements
- How to code arithmetic expressions

Pay attention to how methods are called on instances of a class. The example in section 1.13 starting on page 59 is good for this. The example program shows how a **Scanner** instance is created, then the instance is used to call methods on it.

Be aware that most method calls must be done on an instance of a class. For example, say we have two instances of a **Robot** class stored in variables **r1** and **r2**. Assume that there is a **move** method in that class that is able to move the robot one space straight ahead (whatever one space is). If we wanted to move the **Robot** instance in **r1** by 1 space but leave robot **r2** stationary, then w would write a statement like this:

```
r1.move();
```

A corresponding function written in Matlab might look like this:

```
move(r1)
```

and in Scheme it might look like this:

```
(move r1)
```

So in Java it's almost as if the leading **r1** is another argument to the method/function call, and in a way that's true. In Java it's more like you're saying, "Hey r1, please move one space!"

Start reading chapter 2 which begins on page 99, and read up to the middle of page 102.

Skip ahead to chapter 4 and read pages 249 through 258. This introduces the increment/decrement operators and the **while** loop.

The following subsections have you install an external library file that you will use for the assignment. The library file gives you some classes that you can use that would otherwise be beyond the scope of this course.

5.1. Make a new Eclipse project

Create a new Java project in Eclipse. Name it anything you like, but I recommend using a systematic naming scheme that keeps your projects organized well. For the first two projects I used these names:

- **CSE1102-01-Hello**
- **CSE1102-02-RobotGrid**

The benefit to using the course name prefix is that if you use Eclipse for some other course, those projects will be grouped together.

The benefit to numbering the projects is that otherwise they will appear in alphabetical order. Maybe you like that better, but I like keeping them in the order I created them.

The benefit to using two digits is to keep the numeric ordering in case the number of projects exceeds 9. Otherwise the ordering would look like this:

- **CSE1102-1-Hello**
- **CSE1102-10-FinalProject**
- **CSE1102-2-RobotGrid**

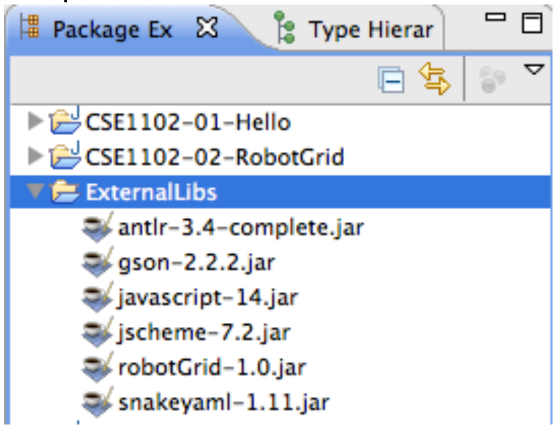
The benefit to not using spaces in the name is that it prevents potential problems in dealing with names that have spaces in them. I won't go into it here. You can use spaces if you prefer. It shouldn't be a problem in this course.

5.2. Make a new general project

After you create the project, also create a new *general project* called **ExternalLibs**. This is where you will place additional library files this semester. I will give you those, and in fact I have one for you for this project.

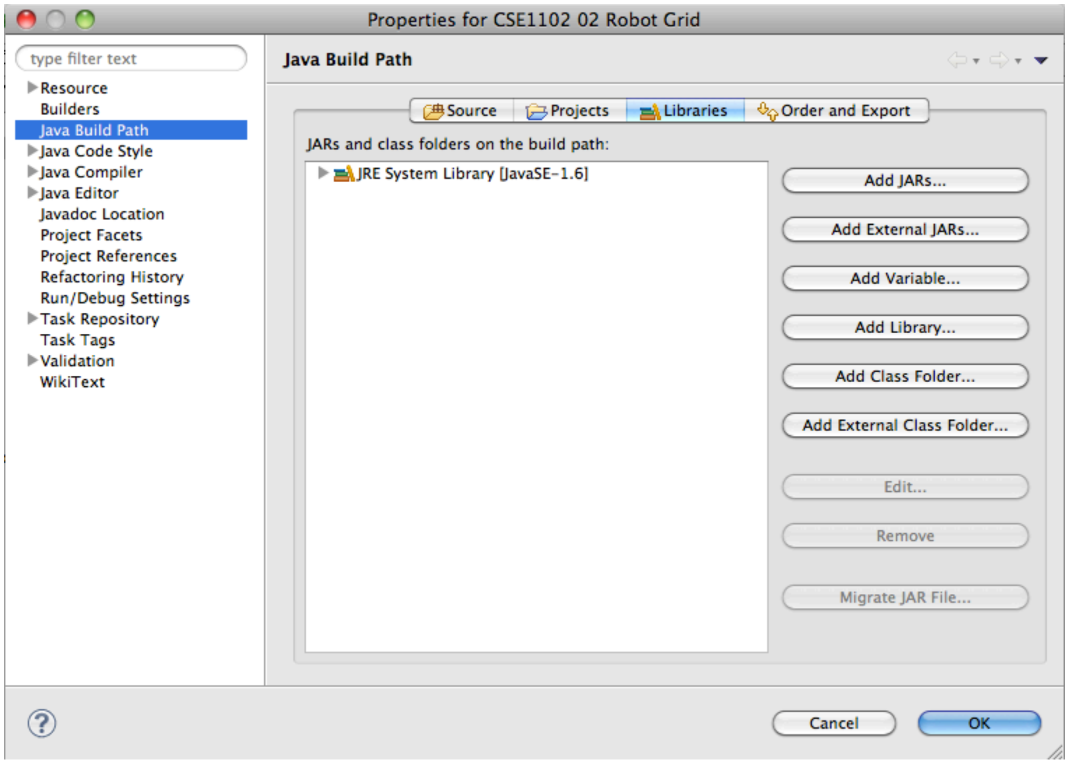
5.3. Install robotGrid.jar

Download this file:
https://dl.dropboxusercontent.com/u/2234170/courses/CSE1102/jars/robotGrid-1.0.jar
Drag & drop it into the **ExternalLibs** folder in Eclipse.

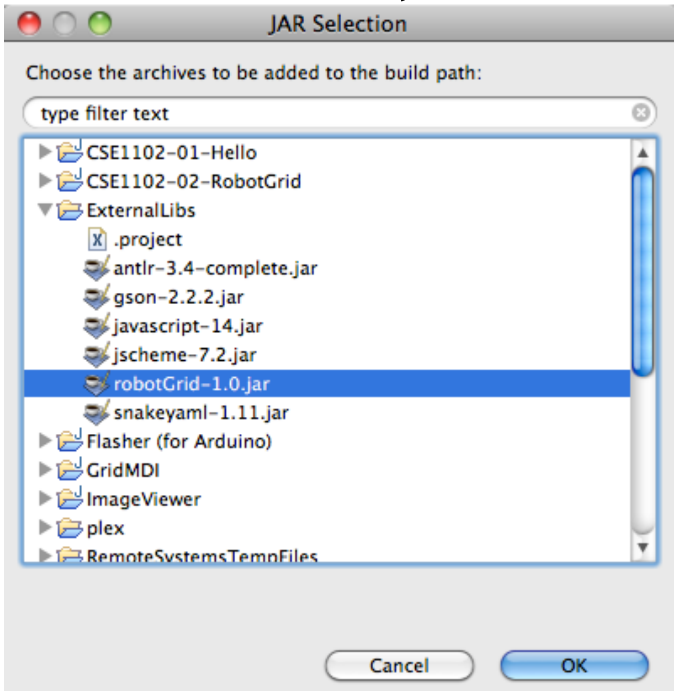


I already have several other jar files in my **ExternalLibs** folder, but you will have only this one. Each jar file contains other Java classes that someone has already written that your program will use. Unfortunately, copying the jar file into the folder is just the first step — there's more you need to do to set it up. In order to make it work with your project, you need to tell Eclipse to actually go look in that jar file for classes.

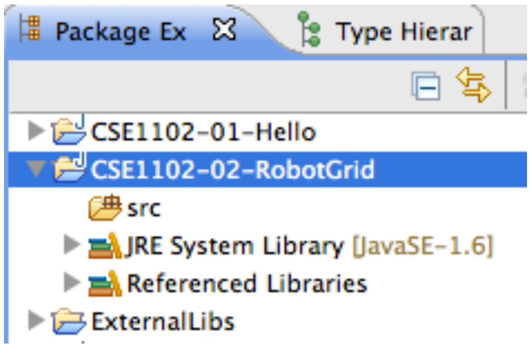
Make sure that **CSE1102-02-Robot Grid** is selected in the **Package Explorer** list on the left of the screen (just click on it once). From the menu choose **Project > Properties**, and then choose **Java Build Path** from the list of items on the left. Make sure **Libraries** is selected near the top.



Click the button **Add Jars…**, and in the window that appears, expand the **ExternalLibs** folder and select the **robotGrid-1.0.jar** file.



Click **OK**, and then click **OK** again to close the **Properties** window. A new entry appears in your project folder called **Referenced Libraries**:



Inside that folder you can verify that the **robotGrid-1.0.jar** file is now shown, meaning that it is now part of your project.

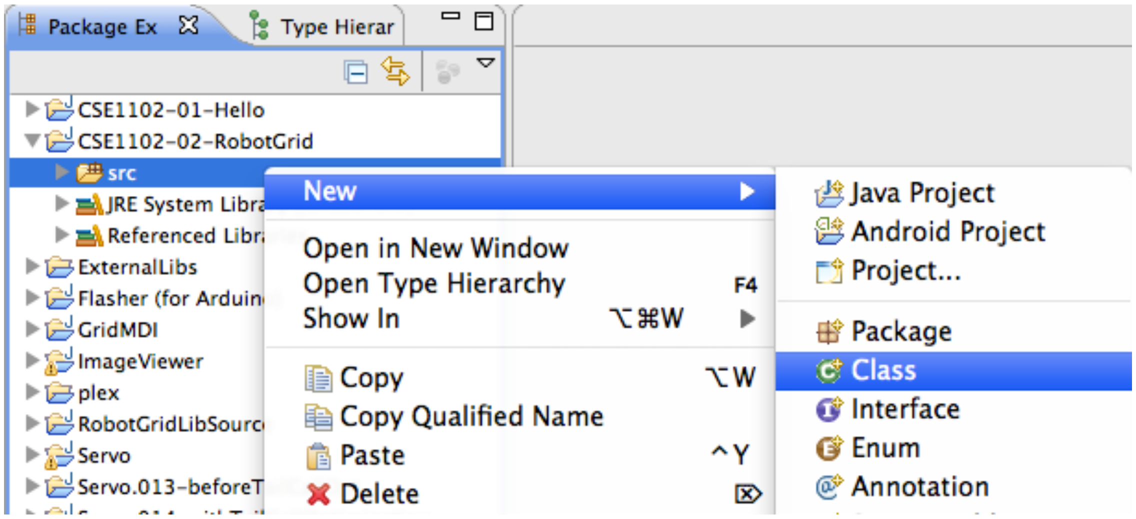
5.4. Make sure it works

Create a new class in this project. There are two ways to do it:

1. Click on the **C+** icon in the tool bar:

   The icon means *add a class*. It has nothing to do with the C++ language.

2. Or you could right-click on where it says **src** in the package explorer, then select **New** > **Class**.



Name the class **Test** and press **Finish**. Eclipse creates a new class that has an empty body. Delete all of that, and copy and paste the following class in its place. Do not worry about understanding this program. I will routinely give you complex programs to use, but you will need to understand only small parts of them.

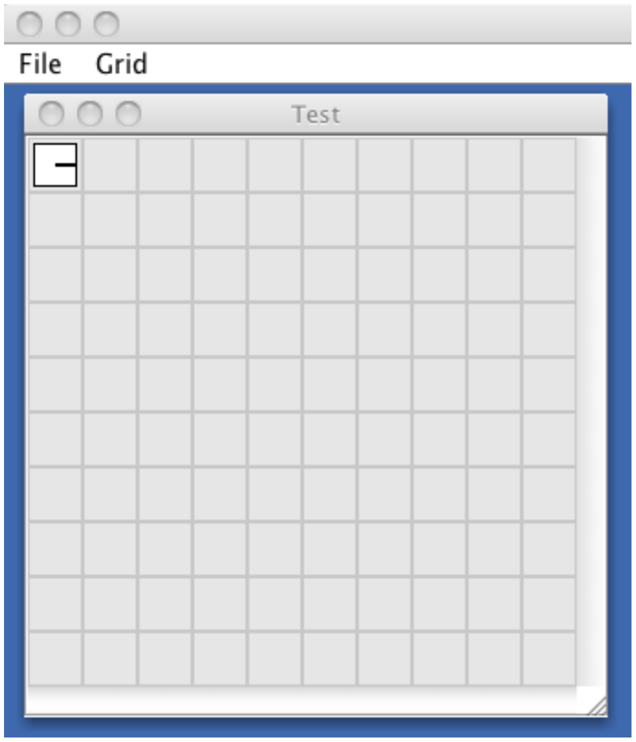
```
import robotgrid.Grid;
import robotgrid.GridServer;
import robotgrid.Heading;
import robotgrid.Robot;
import robotgrid.Utils;

public class Test
{

  public static void main(String[] args)
  {
    GridServer server = new GridServer();
    Grid grid = server.createGrid("Test", 10, 10 );
    Robot robot1 = new Robot("Robot 1", Heading.East);
    grid.placeObject(robot1, 0, 0);
    // pause a bit before things start to happen
    Utils.sleep(500);
  }

}
```

Save the program and then run it. To run it, choose **Run** > **Run** from the menu. If it asks you how to run the project, choose **Java Application**. This is what you should see:



Nothing should move, but if you see this then you've done everything correctly.

5.5. What is it?

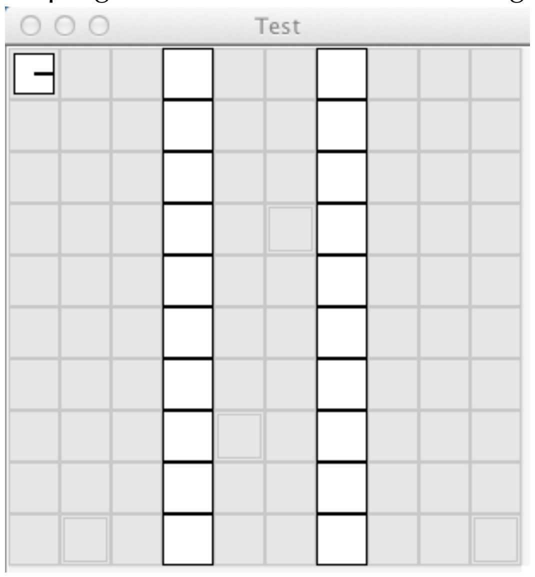
In the example program there are three primary things you should notice:

1. The main window. This has a blue background.
2. The grid window inside the main window. This contains a bunch of gray squares.
3. The robot. This is the white square that has a direction indicator. In this example the robot starts by pointing toward the east.

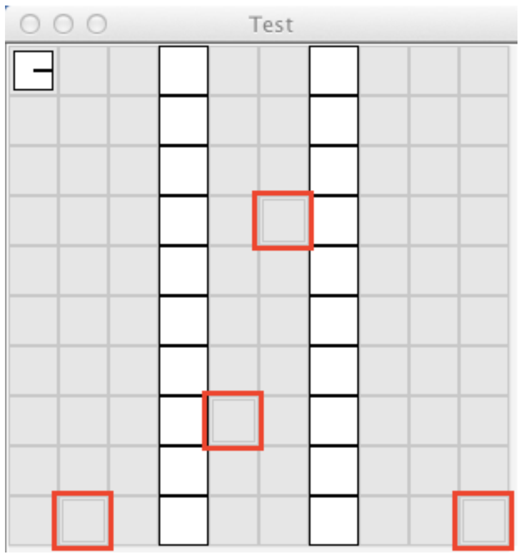
In this project you will write a program to move the robot around the grid and interact with the environment. In subsequent projects, you may have robots interact with each other and use things that it finds on the floor.

6. Assignment

Read through all of the subsections in this section before you start in order to make sure you understand how you must proceed with the assignment. For this assignment you will create a program that moves a robot through a simple obstacle course that looks like this:



The white squares form a wall that blocks the robot. There are four triggers on the floor that are difficult to see in that image above. They are here:



1. The trigger in the lower left opens a secret doorway in the first wall so that the robot can pass through it.
2. The trigger in the lower middle opens a secret doorway in the second wall.
3. The trigger in the upper middle is a checkpoint that simply pops up a message.
4. The trigger in the upper left is the finish square.

It doesn't matter in what order the robot touches the middle two triggers.

Here's a video I made of my solution to the problem.
https://dl.dropboxusercontent.com/u/2234170/courses/CSE1102/videos/RobotGrid.mov
Your solution does not need to work exactly like mine as long as the triggers are activated in the right order and the robot doesn't waste too much time doing it.

6.1. Comment block

Add this comment block at the top of the class file and fill in the correct information.

```
// Project 2: Robot Grid
// CSE1102 Spring 2015
// (your name goes here)
// (the current date goes here)
// TA: (your TA's name goes here)
// Section: (your section number goes here)
// Instructor: Jeffrey A. Meunier
```

6.2. The setup method

Start with the program you used in section 5.4. Copy & paste this **setup** method into the **Test** class, but make sure it's below the main method. You don't need to understand this method or anything in it.

```
  private static void setup(final Grid grid)
  {
    for(int n=0; n<10; n++)
    {
      Wall wall = new Wall();
      grid.placeObject(wall, 3, n);
      grid.placeObject(wall, 6,  n);
    }
    Layer layer = grid.getLayer(null);
    TriggerCell tc1 = new TriggerCell(layer, 1, 9, "") {
      @Override
      public void activate()
      {
        grid.removeObject(3, 5);
      }
    };
    TriggerCell tc2 = new TriggerCell(layer, 4, 7, "") {
      @Override
      public void activate()
      {
        grid.removeObject(6, 5);
      }
    };
    layer.setCell(1, 9, tc1);
    layer.setCell(4, 7, tc2);
    TriggerCell tc3 = new TriggerCell(layer, 5, 3, "Checkpoint");
    layer.setCell(5, 3, tc3);
    TriggerCell tc4 = new TriggerCell(layer, 9, 9, "Finished!");
    layer.setCell(9, 9, tc4);
  }
```

You will need to add a few more **import** statements in order to get this to work. Hover your mouse over the errors and see if Eclipse can fix the errors for you.

Thus, the Test class will end up having this structure. There are two methods inside the Test class, but no method is inside any other method.

```
public class Test
{
  public static void main(String[] args)
  {
    ...
  }

  private static void setup(final Grid grid)
  {
    ...
  }
}
```

Now add a call to the **setup** method in your main method, just below the **grid.placeObject** method call. It looks just like this:

```
    grid.placeObject(robot1, 0, 0);
    setup(grid);
    // pause a bit before things start to happen
    Utils.sleep(500);
```

This will set up the obstacle course for your robot to run.

6.3. How to make the robot move

The code you must write is not terribly complicated, and you will simply be adding lines inside the main method that I gave you already. I'll guide you through it in steps. Read this whole section before you start to type.

The methods you will need for this solution are as follows:

- **move()** - moves the robot one space forward
- **move(n)** - moves the robot n spaces forward
- **turnRight()** - turn the robot 90 degrees to the right (clockwise)
- **turnRight(n)** - turn the robot *n* * 90 degrees to the right
- **turnLeft()** - turn the robot 90 degrees to the left (counter-clockwise)
- **turnLeft(n)** - turn the robot *n* * 90 degrees to the left

At first you will write a solution using the **move(n)** method whenever you need to move the robot more than one space. Then I will ask you to rewrite the program using loops in place of the **move(n)** method calls.

Notice that the **move** method is being called on the **robot1** variable. If we wanted to have a second robot we could call the variable **robot2**, and then you'd be able to tell each of them to move differently.

6.4. Write your solution

Type the statements inside the main method just below the **Utils.sleep(500);** line.

```
    // pause a bit before things start to happen
    Utils.sleep(500);
    your program statements go here!
  }
```

Have the robot drive over all the trigger squares on the grid. It doesn't really matter what order you use, as long as it hits all of them.

First: Solve the problem using just the methods I showed you above. My solution has about 20 statements below the **Utils.sleep(500)** method call, but yours may have more or fewer. Verify that your solution works.

6.5. Rewrite your solution

Now you will change your program so that instead of using the **move(n)** method, it will use the **move()** method inside a **while** loop. In other words, in this second version you're not allowed to use the **move(n)** method. Your TA will want to see this changed version that contains the loops.

Here's how you make the robot move some number of squares using a loop:

```
1.  create a variable and set it to 0
2.  while the variable is less than the number of squares to move
3.  {
4.    robot1.move();
5.    increment the variable (increment means add 1 to the variable)
6.  }
```

Lines 3, 4, and 6 must appear exactly as I show you.

The remaining lines must be replaced with Java code.

- Line 1 must declare and initialize a new **int** variable. Choose a name that you like. You can change the name later if you decide you don't like the name.
- Line 2 must be a while loop.
- Line 5 can be solved by reading pages 249 to 253 in the book (use *postincrement*).

If you prefer to place the opening curly brace at the end of the previous line instead of at the beginning of its own line, then you may do that. You must be consistent throughout your program.

You will need to use several loops in the program, but you will *not* need to write a loop within a loop. You will not create any other methods. You will not need to use any conditional statements (if/else).

For the second loop that you use, you should re-use the same variable that you used for the first loop. The second time you assign a value to the variable, you must not re-declare the variable. For example, say I used this statement before the first loop:

int n = 0;

Recall that it's a *variable definition*. The second time I used the variable, I would type this assignment statement:

```
        n = 0;
```

Typing the word **int** before it would be an error because the variable has already been defined. Go ahead and try it, though, to see what Eclipse tells you. It's important to become familiar with the error messages in Eclipse.

You are allowed, but not required, to use different kinds of loops if you know them already or feel like learning them on your own (like the **for** loop, and thus your variables may be declared differently), but you must use at least 1 **while** loop.

Finish the program now!

7. Report

Create a Microsoft Word or OpenOffice Word file (or some other format that your TA agrees on — ask him or her if you are not sure). Save the file with the name **CSE1102 Homework 2** with a .doc or .docx format.

At the beginning of the document include this information:

Robot Grid
CSE1102 Project 2, Spring 2015
Your name goes here
The current date goes here
TA: *Your TA's name goes here*
Section: *Your section number goes here*
Instructor: Jeffrey A. Meunier

Be sure to replace the parts that are underlined above.

Now create the following three sections in your document.

1. **Introduction**
   In this section copy & paste the text from the introduction section of this assignment. (It's not plagiarism if you have permission to copy something. I give you permission.) It wouldn't hurt to rewrite it a bit, but you don't have to.
2. **Output**
   Run your program and copy & paste a screen shot of the program showing the **Finished!** message. You do not need to write anything. *Windows users*: When you take a screen shot, open Paint and paste the image into that application, then save the file as a JPG or PNG image file, not a BMP file (you may also crop out anything outside the Java program). Only then may you copy & paste that image into your

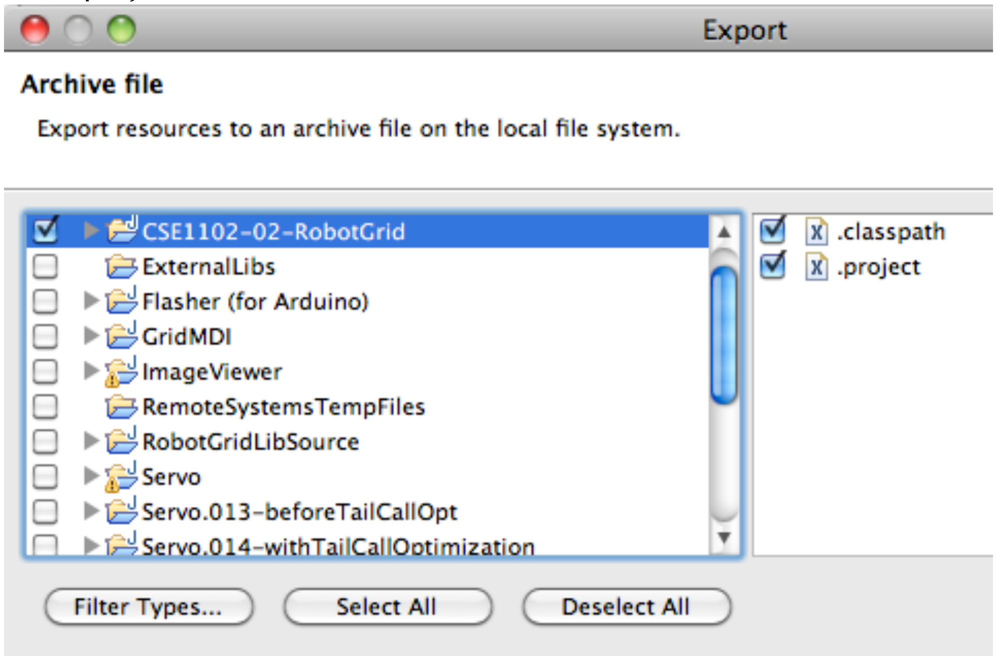
document. The problem is that if you go straight from screen shot to Word, it will paste as a BMP image, which is huge, and your document will be several megabytes instead of several kilobytes.

3. **Source code**
   Copy & paste the contents of your **Test.java** file here. You do not need to write anything.

8. Project export

Export your project from Eclipse following these steps:

1. Right-click on the **CSE1102-02-RobotGrid** project folder in the **Package Explorer** window on the left side of the screen in Eclipse.
2. Select **Export**.
3. Chose **General > Archive File**.
4. Make sure the **CSE1102-02-RobotGrid** folder is selected, and also the **.classpath** and **.project** files. It should be that way already. It looks like this (ignore all the other projects I have there):



Export

Archive file

Export resources to an archive file on the local file system.

- [✓] CSE1102-02-RobotGrid
- [ ] ExternalLibs
- [ ] Flasher (for Arduino)
- [ ] GridMDI
- [ ] ImageViewer
- [ ] RemoteSystemsTempFiles
- [ ] RobotGridLibSource
- [ ] Servo
- [ ] Servo.013-beforeTailCallOpt
- [ ] Servo.014-withTailCallOptimization

- [✓] .classpath
- [✓] .project

5. Press the **Browse…** button and save the file onto your desktop with the name **CSE1102-02-RobotGrid.zip**.
6. Make sure these options are selected:
   (•) Save in zip format
   (•) Create directory structure for files
   [√] Compress contents of file
7. Press the **Finish** button.

9. Submission

Submit the following things things on HuskyCT:

1. The **Eclipse project zip file** that you just saved to your desktop.
2. The **MS Word document**.

If for some reason you are not able to submit your files on HuskyCT, email your TA before the deadline. Attach your files to the email.

10. Grading Rubric

Your TA will grade your assignment based on these criteria:

- (2 points) The comment block at the beginning of the class file is correct. You may use a different formatting as long as it contains at least the information I asked for.
- (6 points) The program shows a working solution.
- (4 points) The program does not use incorrect statements to generate the solution. That means that you must solve the problem the way I asked you to solve it: use at least one while loop, re-use the loop counter variable, etc.
- (4 points) The program is formatted neatly. Use consistent indentation and brace placement. Follow how the book does it if you're not sure. Ask your TA if you are not sure, but don't just send your entire program to your TA and ask if it's correct. Ask only if you're unsure of some part, and send only that part.
- (4 points) The document contains all the correct information and is formatted neatly.

11. Getting help

Start your project early, because you will probably not be able to get help in the last few hours before the project is due.

- If you need help, post a message on Piazza immediately.
- In second order of priority, send e-mail to your TA.
- Go to the office hours for (preferably) your TA *or any other TA*. I suggest you seeing your own TA first because your TA will know you better. However, don't let that stop you from seeing any TA for help.
- Send e-mail to Jeff.
- Go to Jeff's office hours.