CS 4053/5053
Homework 01 – Building Basic Graphics Applications with JOGL
Due Tuesday 2023.02.07 at 11:00pm.

All homework assignments are individual efforts, and must be completed entirely on your own.

The purpose of assignment you will make sure your system is set up correctly for completing the homework assignments. You will then learn how to develop a basic graphics application using OpenGL, Java, and Gradle. Specifically, you will install the necessary development software on your system, verify that you can build and run the provided code, add a small amount of new code to animate simple sets of points and lines using the JOGL API, and learn how to clean up the code for submission.

Learning about Gradle

All of the homework assignments will involve building and running Java code using the *Gradle* build tool. Visit https://docs.gradle.org/current/userguide/userguide.html to get a sense of what Gradle does and how it works. You **don't** have to install Gradle on your system. You won't even have to do any Gradle scripting. I've done that for you.

In the `homework01` download, go into the `ou-cs-cg` directory in `Build`. Read the `about.txt` file to learn about running Gradle tasks on the command line and the tasks you're most likely to need. Feel free to take a peek at the `build.gradle` script file. The *Alternative Start Scripts* section at the end describes how to create additional scripts to run alternative main() classes. You won't need to, but the possibility exists. Don't otherwise change `build.gradle`!

Installing a JDK

All code for the homework assignments will be written using Java 8. Unfortunately, not all recent versions of Java work consistently (or at all) with JOGL, which makes it much more difficult to get everything set up correctly across systems. Java 8 still works essentially everywhere with little or no pain, and it has all the features we need to learn JOGL for CG development anyway.

You will need to have a Java 8 JDK installed on your system. I recommend you install *Oracle JDK 8u351*. Go to https://www.oracle.com/java/technologies/downloads/ and scroll down to the Java 8 section. Download the JDK for your system architecture. Links to installation instructions and online documentation are a little farther down. I recommend actually downloading the JDK documentation ("Documentation Download") for fast reference while coding even when offline. Keep the JOGL API handy too. Bookmark the APIs in your browser favorites for easy access.

Search "*install JDK 8 on <my system>*" to find other Java 8 JDKs. OpenJDK and Adoptium are popular alternatives. You are welcome to use them if they work with JOGL, but beware; JOGL depends on some low-level Java APIs and not all JDKs may provide the necessary support. Regardless of which JDK you choose, your system is unique, so your path to successful installation and configuration may vary. In general, try the following steps:

1. Install a Java 8 JDK on your system if there isn't one already. Make sure to install a **JDK** (Java Development Kit), not just a **JRE** (Java Runtime Environment).

2. If your system has multiple versions of Java installed, make sure it is set up to use JDK 8 and not any other version. Search "*switch java version on <my system>*" for help with that. The top hits, at medium.com for Mac and happycoders.eu for Windows, are very helpful.

3. You may need to restart your command line or even reboot for changes to be visible.
4. Confirm the version of Java by running the '`java -version`' and '`javac -version`' commands on the command line. Both should display a version number starting with '`1.8`'.
5. Use Gradle on the command line to build and run the code. See the next section for how.

Building and Running Applications from the Command Line

The `base` program is very...well, basic! Its purpose is to help you learn one way of organizing JOGL code for simple graphics applications. Read through the code in the corresponding class, `Base.java` . You can find the source code deep in the `src` directory. The package structure is organized to hold code both for your homework assignments and for the example applications that we'll see in class. (The source code for the various examples will be revealed progressively; the download for each homework assignment will include an updated `ou-cs-cg` directory with the additional code and a suitably modified `build.gradle`.)

To build the code, open a command line. Navigate to the `ou-cs-cg` directory. Run the Gradle `installDist` task. **You need to be in the `ou-cs-cg` directory for Gradle tasks to work.** Refer to `Build/ou-cs-cg/about.txt` for details on Gradle tasks and command syntax.

To run the code, open a command line. Navigate to the `build/install/base/bin` directory. Run any of the programs by entering its name. You can also run it by double-clicking its icon in a window on your desktop. Each program has a `.bat` version for running on Windows. *(Note: Do not try to use `gradlew run` to run programs. It is effectively just an alias to the `base` program.)*

I suggest keeping two command line windows open whenever you work on the code, one for building and one for running. That way, you can quickly rebuild and rerun between code edits without changing directories repeatedly. Editing itself can be in any text editor, including an IDE.

Using Gradle with IDEs and Other Coding Tools

You can use an IDE to *edit* code even if you build and run from the command line. If you'd like to also *build and run* inside the IDE, see the suggestions below for using Gradle in common IDEs. Refer to `Build/ou-cs-cg/about.txt` for Gradle commands specific to particular IDEs.

Eclipse: First, make sure you're using a version of Eclipse with the BuildShip plugin installed. Most recent Eclipse for Java distributions come with it. Second, run `gradlew eclipse` in the `ou-cs-cg` directory to prepare it for import into Eclipse. Third, import the `ou-cs-cg` directory as a *Gradle / Existing Gradle Project* using the Eclipse import wizard. Eclipse will add a *Gradle Tasks* pane to the editing window for running Gradle commands.

IntelliJ IDEA: See https://www.jetbrains.com/help/idea/gradle.html for information and help. *(Note: If you insist on coding directly in an IDE, I suggest trying out the free IDEA Community Edition. Many students who have tried it tell me they like it, especially coming from Eclipse.)*

Other IDEs: You mileage may vary. Try searching on "*open gradle project in <ide>*". Even if your IDE doesn't support Gradle directly, you can still use the IDE's editor to edit your code, but use Gradle on the command line to run build commands. If you prefer to use a standalone text/code editor like Atom or Sublime, you can use Gradle on the command line in exactly the same way.

Depending on your IDE, you may have to change its settings to use the Java 8 JDK that you installed for compiling and running. *(Note: You shouldn't have to download the JOGL library or add its location to the IDE settings. Gradle takes care of the JOGL dependency for you.)*

Some assignments require adding files (like images) to the codebase. If you use an IDE, be careful how you do that. Some IDEs import files to locations visible only to the IDE. Always add files directly to the desired location inside `ou-cs-cg/src` via the regular file system using a command line or your desktop. Doing that will keep the code self-contained and transportable.

Regardless of which coding tools you use, always make sure your code **builds from scratch and runs as intended using Gradle on the command line**, since that's how we'll verify the build and launch the resulting programs when we grade the assignments.

Exploring an Example

Let's look at a simple point-drawing program written using JOGL. Run the `lorenz` program. It displays an animation of the Lorenz attractor (https://en.wikipedia.org/wiki/Lorenz_system). For differential equations like in the Lorenz system, one could integrate to calculate points precisely, such as with a high-order Runge-Kutta method. However, simple iteration and addition works for our purposes. (See http://www.algosome.com/articles/lorenz-attractor-programming-code.html.)

The corresponding code is in the `Lorenz.java` class. Study the code and how it's organized to learn more about how to update and render an animated scene with JOGL. In particular, look at how the number of points to draw, `m`, increases exponentially but resets each time it hits a cap. Then look at how those points are iteratively calculated and drawn as a set using `GL_POINTS`.

Creating a New Example

It's finally time to *create* some animated point and line art of your own using OpenGL. Start by reading about the Tinkerbell map (https://en.wikipedia.org/wiki/Tinkerbell_map). Then write code to animate an increasing number of points, like in the Lorenz example. For the number of points, set the cap and rate of increase to produce an interesting and pleasant visual result. Choose the four Tinkerbell map constants likewise. Adjust the point locations to show the whole scene; there are several ways to do this.

When the number of points resets after passing the cap, switch to drawing line segments using `GL_LINE_STRIP`. After the second reset, use coordinates (y_n, y_{n+1}) instead of (x_{n+1}, y_{n+1}) as the point locations to draw the line segments. After the third reset, switch back to drawing just the points with `GL_POINTS`. Cycle between the four "modes" on subsequent resets, indefinitely.

For this assignment, navigate to the `edu.ou.cs.cg.assignment.homework01` package and modify `Application.java` to implement your Tinkerbell example. Don't create any new Java files or directories for subpackages. (You'll be allowed to create both in future assignments.) You may add private methods if it helps to modularize your code. Document your code appropriately. The corresponding program to run is `hw01` in the `build/install/base/bin` directory.

Turning It In

Turn in a complete, cleaned, renamed, zipped **COPY** of your **ENTIRE** `homework01` directory:

- Never delete `About` or `Results`! Preserve all file structure and contents of the assignment's original zip download, except for any modifications and additions specified in the instructions.
- Take a screenshot of your application window when it's in an interesting graphical state.
- Put the screenshot in the `Results` directory as `snapshot.png` or `snapshot.jpg`.
- Go into the `ou-cs-cg` directory.
  - Make sure it contains all of the code modifications and additions that you wish to submit.

    - Run `gradlew clean` (on the command line). This should remove the `build` directory, reducing the size of your submission. We will clean and rebuild when we grade regardless.
    - If you used an IDE, remove any IDE-specific leftovers (such as the Eclipse `bin` directory).
- Append your 4x4 to the `homework01` directory; mine would be `homework01-weav8417`.
- Zip your entire renamed `homework01-xxxx####` directory.
- Submit your zip file to the Homework01 assignment in Canvas.

These steps will make your submissions smaller and neater, which speeds up grading a lot.

The actual coding in this assignment is meant to be fast and easy. We'll score the assignment on how well you followed the turn-in instructions, how accurately you reproduced a recognizable Tinkerbell map, how well the animation works as described above, and...how cool it looks!