Objectives
To continue practicing the use of GitHub to access the files for a practical assignment and the use of a “terminal window” to compile and run Java programs. To get a deeper understanding of how to create your own classes and methods in Java. To experiment with the structure of different classes by studying the given Java programs and modifying them by adding more functionality to the given methods.

Reading Assignment
You should review Sections 4.1 through 4.3 of your book to learn more about writing your own classes, constructors and methods. As you review this material, try to make a list of questions about concepts that you do not yet fully understand. In addition to discussing these questions with the teaching assistants and the course instructor, try to answer them as you complete this assignment.

Accessing and Studying the Example Programs
To access the practical assignment, you should go into the #practicals channel in our Slack team and find the announcement that provides a link for it. Unless you provide the instructor with documentation of the extenuating circumstances that you are facing, not accepting the assignment means that you automatically receive a failing grade for it.

Now, navigate the src/ directory to find files titled “Octopus.java”, “Utensil.java,” and “Practical9.java”. Study these programs by following through with the program execution flow manually. Compile and run these programs, and make sure you understand why and how the output is produced.

Modify the Example Programs
This practical assignment invites you to modify the given programs in the following manner.

1. Edit the file Octopus.java and find the constructor in this class. In the constructor, there is one parameter octopusName, which contains a String. Change this by adding one more parameter, octopusAge, of type int. This is the age of the octopus. Assign this to the appropriate instance variable (imitating what was done for the name).

   • Edit the file Practical9.java and look for the place where variable ocky is defined to be a new Octopus. Add an “age” to this so that we are specifying two things, not one, in the construction. Delete or comment out the “ocky.setAge(10)” method call in the next line.

   • Recompile and re-run the program and see if it correctly provides the age you specified.
2. • Edit the file Practical9.java. Declare a second Octopus variable (don’t just change the name of the one that’s there—create another one) and assign it any name and age that you want.
  • Create a second Utensil of any type you wish, imitating the declaration and initialization of spat. Assign a cost and a color to this utensil. Assign this utensil to the new Octopus you created.
  • Print out the name, age, weight, and favorite utensil of your new octopus. Print out the type, cost, and color of your new utensil.

3. If you finish with the tasks above you can also change values for your new octopus and its new utensil by using set methods on the new octopus object.

Testing your Program

Once you have written parts of your program, you are ready to use tools that build and run your program! If you are using Docker Desktop, you can use the following “docker run” command to start “gradle grade” as a containerized application, using the “DockaGator” Docker image available on DockerHub. You can run the following command to run the “gradle grade” on your project:

docker run --rm --name dockagator \ 
   -v "$(pwd)":/project \ 
   -v "$HOME/.dockagator":/root/.local/share \ 
   gatoreducator/dockagator

The aforementioned command will use "$(pwd)" (i.e., the current directory) as the project directory and "$HOME/.dockagator" as the cached GatorGrader directory. Please note that both of these directories must exist, although only the project directory must contain some content. Generally, the project directory should contain the source code and technical writing for this assignment, as provided to you through GitHub during the completion of a previous step. Additionally, the cache directory should not contain anything other than directories and programs created by DockaGator, thus ensuring that they are not otherwise overwritten during the completion of the assignment. To ensure that the previous command will work correctly, you should create the cache directory by running the command “mkdir $HOME/.dockagator”; you will only need to do this once. If the above “docker run” command does not work correctly on the Windows operating system, then you may need to instead run the following command to work around limitations in the terminal window:

docker run --rm --name dockagator \ 
   -v "$(pwd)":/project " \ 
   -v "$HOME/.dockagator:/root/.local/share" \ 
   gatoreducator/dockagator

To enter into an “interactive terminal” in the Docker container, you can instead use the following command

docker run -it --rm --name dockagator \ 
   -v "$(pwd)":/project \ 
   -v "$HOME/.dockagator:/root/.local/share" \ 
   gatoreducator/dockagator /bin/bash
Now, if you want to “build” your program you can type the command “gradle build” in your terminal, thereby causing the Java compiler to check your program for errors and get it ready to run. If you get any error messages, go back into your text editor and try to figure out what you mis-typed and fix it. Once you have solved the problem, make a note of the error and the solution for resolving it. Re-save your program and then build it again by re-running the “gradle build”. If you cannot build Practical9 correctly, then please talk with a technical leader or the instructor. When all of the errors are eliminated, you can run your program by typing “gradle run” in the terminal window—this is the “execute” step that will run your program and produce the designated output.

Checking the Correctness of Your Program and Writing

To check your Java source code you can started with the use of GatorGrader, type the command “gradle grade” in your terminal window. If you do have mistakes in your assignment, then you will need to review GatorGrader’s output, find the mistake, and try to fix it. Specifically, don’t forget to add in the required comments! If you are having trouble running GatorGrader locally, don’t forget to ensure that you still transfer all of your source code to GitHub. Please see the course instructor if you have questions about this step.

Once your program is building correctly, fulfilling at least some of the assignment’s requirements, you should transfer your files to GitHub using the “git commit” and “git push” commands. For example, if you want to signal that the src/main/java/practicalnine/Practical9.java file has been changed and is ready for transfer to GitHub you would first type “git commit src/main/java/practicalnine/DisplaySymbols.java -m "Your descriptive commit message"” in your terminal, followed by typing “git push” and checking to see that the transfer to GitHub is successful. If you notice that transferring your code to GitHub did not work correctly, then please try to determine why, asking a technical leader or the course instructor for help, if necessary.

When you use the “git push” command to transfer your source code to your GitHub repository, Travis CI will initialize a “build” of your assignment, checking to see if it meets all of the requirements. If both your source code meets all of the established requirements, then you will see a green ✓ in the listing of commits in GitHub after awhile. If your submission does not meet the requirements, a red X will appear instead. You should aim to finish practical assignments on the day that they are assigned; please see the instructor if you do not understand this policy.

Checking the Correctness of Your Program

As in the past assignments, you are provided with an automated tool for checking the quality of your source code. Please note that the practical assignments do not require you to produce a writing document as you do in the laboratory assignments. However, to check your Java source code you can started with the use of GatorGrader, type the command “gradle grade” in your terminal window. If you do have mistakes in your assignment, then you will need to review GatorGrader’s output, find the mistake, and try to fix it. Specifically, don’t forget to add in the required method!

Once your program is building correctly, fulfilling at least some of the assignment’s requirements, you should transfer your files to GitHub using the “git commit” and “git push” commands. For example, if you want to signal that the src/main/java/practicalnine/Practical9.java file has been changed and is ready for transfer to GitHub you would first type “git commit src/main/java/practicalnine/DisplaySymbols.java -m "Your descriptive commit message"” in your terminal, followed by typing “git push” and checking to see that the transfer to GitHub is successful. If you notice that transferring your code to GitHub did not work correctly, then please try to determine why, asking a teaching assistant or the course instructor for help, if necessary.
When you use the “git push” command to transfer your source code to your GitHub repository, Travis CI will initialize a “build” of your assignment, checking to see if it meets all of the requirements. Since this is another challenging practical assignment and you are continuing to learn how to repeat actions with while loops, don’t become frustrated if you make a mistake. Instead, use your mistakes as an opportunity for learning both about the necessary technology and the background and expertise of the other students in the class, the teaching assistants, and the course instructor.

Summary of Required Deliverable

This assignment invites you to submit, using GitHub, the following deliverable. Because this is a practical assignment, you are not required to complete any technical writing.

1. A correct version of src/main/java/practicalnine/Practical9.java, src/main/java/practicalnine/Octopus.java and src/main/java/practicalnine/Utensil.java that meet all of the established source code requirements and produces the desired text-based output.

Evaluation of Your Practical Assignment

Practical assignments are graded on a completion — or “checkmark” — basis. If your GitHub repository has a ✓ for the last commit before the deadline then you will receive the highest possible grade for the assignment. However, you will fail the assignment if you do not complete it correctly, as evidenced by either a red ✗ in your commit listing or the absence of a functioning GitHub repository for this practical assignment, after the set deadline for completing the project. Please see the course instructor if you do not understand how practical assignments are graded or you do not know how to complete one of the specific tasks in this assignment.

Adhering to the Honor Code

In adherence to the Honor Code, students should complete this practical assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else’s work. Deliverables (e.g., the Java source code) that are nearly identical to the work of others will be taken as evidence of violating the Honor Code. Please see the course instructor if you have any questions about this policy.