

This lab has **six (6)** checkpoints.

Learning Outcomes

Upon completing this lab, students should be able to

- Use `gradle init` to initialize a Java application project using Groovy as the build language.
- Use Java *text blocks* for multi-line literal strings within a program.

Introduction

1. Dr. Ladd found a printout of a Java file from one of his projects. From the root of the project (the default (no-name) package directory), what is the **path** of the *file* that produced the following printout?

```
package bcl.utiltiy.containers;

public class Cupboard {
    public String contents() {
        return "bare";
    }
}
```

✓ Show and explain your answer to this question to the lab instructor.

2. What is the output of the following code? Note that you are looking at a **Java Text Block** (and you should read about it to know what it means).

```
String elements = """
    hydrogen
    helium
    lithium
    beryllium
    boron
    carbon
    """;

Scanner scanIt = new Scanner(elements);
int count = 1;
while (scanIt.hasNextLine()) {
    String item = scanIt.nextLine();
    System.out.println "[" + (count++) + " ] - " + item);
}
```

✓ Show and explain your answer to the lab instructor.

3. Create a new folder for your lab project. Go into that folder and run `gradle init` to initialize the project. The question answers are, in order:

```

2 application    [type]
3 Java           [implementation language]
no              [multiple subprojects]
2 Groovy        [build script language]
4 JUnit Jupiter [testing framework]
greeter        [project name]
greeter        [package where App.java will be]
17             [Java version]
no             [use new API]

```

In the same directory, run `git init` to create an empty git database. Run `ls -A1` to see all folders/files in the root directory¹. Compare this list to the results of running `git status`.

Use `gradle build` to compile your project (the `App.java` somewhere down there) to make sure it compiles. Do an `ls -A1 app/build`.

Use `cat .gitignore` to dump the contents of `.gitignore` to the terminal. Which lines in there are *comments*? What happens to the files you listed just above when you add the whole project to git?

Find the file `App.java` that was just compiled (it will be somewhere below `app/`). What is the *relative* path from the directory with `.gitignore` to the file `App.java`?

✓ Show the lab instructor

- The `ls` and `git status` file lists.
- Your explanation of what, in `.gitignore` is comment.
- The relative path to `App.java`

4. Edit `app/build.gradle` to change the name of the main class started with `gradle run` and add a line to permit the running program to receive keyboard input.

Open the file `app/build.gradle`. This is a Groovy *program* that tells Gradle how to compile and run your whole project. We want to change the `mainClass` value and set a value for `run.standardInput` variable.

Find the line setting `mainClass` in the application block. Change the name of the class to `greeter.Greet`. The class you are going to write below will be called `Greet` and be in the same *package*.

To test compiling, go down to `App.java` and rename it to `Greet.java` and change the public name of the class inside. Don't worry that it just says *Hello* for now.

This was a breaking change because the automatically generated testing code (`app/src/test` and below) uses the `greeter.App` name still. Quickest fix: *remove* `app/src/test/*` (everything in and below `test`).

With the new `mainClass` and renamed class file, make sure `gradle build` works.

✓ Show the lab instructor that your changed/renamed files build.

5. Connect standard input to `System.in` of your *running* program and change `Greet` to ask for a name and print a personalized greeting.

¹`man ls` will show the manual page for `ls`. The `-A` argument is almost all or everything except `.` and `..`; the `-1` (that is the digit one) tells `ls` to print one entry per line.

In `build.gradle` add a new section describing settings for the `run` task. This can go at the end of the file:

```
run {
    standardInput = System.in
}
```

This tells gradle to connect gradle's `System.in` to the `standardInput` of the running executable. So the keyboard is passed in to your program.

Now modify `greeter.Greet` to *prompt* the user for their name, read a whole line in (that can include spaces), and then say, **Hello, <name>, and the rest of the World!**.

Use `gradle run -q --console=plain` to run your program. The `-q` argument stands for *quiet*, limiting the amount of output that Gradle adds while running. The `--console` argument determines how Gradle decorates the input you provide; `plain` tells Gradle to just connect the keyboard to `Standard.in` (which is then passed in to your program).

✓ Show the lab instructor your changed/renamed files, your program building, and then your program running.

6. When you use `gradle build`, the directory structure under `build/classes` mirrors that under `src`. That is, `build/classes/java/main` is the *class-path* to the root of your classes. Using what you know about java and the `--class-path` argument, run your program from the `app` directory with `java` and the appropriate class path. Remember that your main program is in the `greeter` package.

If you were told that `build/libs/app.jar` was the base of `build/classes/java/main`, could you use the `.jar` file in the `--class-path` argument to `java` to run your program w/o gradle?

✓ Run your program w/o gradle for the lab instructor.

7. Add and commit your full gradle project (if you `git add` just the root directory, `git` will add all subdirectories recursively).

Create a repository in your Gitea turn-in organization, associate it with your lab work, and push the repo. This will be done at the end of every lab.