

## Learning Outcomes

Upon completing this assignment, students should be able to

- **Implement** an application using a constructor and `run()` architecture (standard 205 architecture).
- **Compile** and **run** an application that uses a `.jar` file containing a third-party Java package. (Use the class path argument.)
- Use the provided `Menu` class.
- Document a thorough test plan for a reusable object.

## Introduction

Think about the graphical applications you use every day: on a given platform, they tend to have very similar interfaces. Why is this? Because most modern applications are built using a graphical user interface (GUI) *toolkit* that provides window, icon, mouse, and pointer support (making it a WIMP interface).

In CS I, II, and III at SUNY Potsdam, to make programming about the computer science and not about using a particular library, students build *text-mode* or *terminal* programs without standard UI libraries. Here, in CS III, it would be convenient to have a reusable `Menu` that all programs could use, the actual items in the menu being loaded at run-time rather than hard-coded.

Initial use of the `Menu` interface spans two assignments: initially, a *provided* (but pre-compiled) implementation will be used to build a simple calculator program; then, with a working calculator client, students will implement the `Menu` interface themselves.

## Method

Taking a high-level look at the assignment: build a calculator program with one method for each operation where the user is prompted for two integers and then the operator is applied to the parameters. For example, the `add` handler should have the following interaction with the user:

```
First Number 123
Second Number 654
123 + 654 = 777
```

In response to each prompt the user gave a number. The program then calculated the sum.

The main program will create a menu that lists available operators and `quit`, running the appropriate method when the user selects the operator, all until the user quits the program.

The `Menu` interface and the `BCLMenu` implementation of that interface are *provided* inside a `jar` (Java ARchive) file. You will write the `driver.Calculator` application that uses the menu.

## Getting Started

Create a *new* directory for your solution. Initialize a `git` repository in the folder. Create a `.gitignore` file in the directory with the following contents:

```
# .gitignore for Java
# --- lines beginning with hash are comments
*.class
# ignore all compiled .class files (including in subdirectories)
```

Create a README file (full name depends on your chosen format). Put the assignment name, your name, and the due date for the assignment in the README.

To make sure you remember how to use **git**: *add* and *commit* the two new files. You can add/commit to the **git** database multiple times (and will as you write your assignment).

## Helper Files

Look in the assignment repository for the two directories: **data** and **lib**. You should copy (**recursively**) these directories and their files into your solution directory. Each has one file: **data/calculator.menu** describes the menu for the calculator in the format below. This is used to create an appropriate **Menu**.

**lib/bclMenu.jar** is the Java archive containing the **BCLMenu** class files. By telling **javac** and **java** to look *inside* the archive, you can compile and run a Java program with the already compiled class.

## Starting the Application

Your application is a Java file in the **driver** package. A Java *package* is a directory. The application is to be stored in **driver.Calculator.java**: create the directory and open the file.

The **first line** in a **.java** file *inside* a package is the **package** line. Add this to your file now:

```
package driver; // names the package (directory) where file is
```

The application is a Java class with a **main** method. For this class, almost all **main** methods will be about two lines long, for example:

```
public static void main(String[] args) {
    Calculator calc = new Calculator();
    calc.run();
}
```

The important thing to notice here is that **main** is **static**. This means it cannot access *data fields* or non-static methods of **Calculator**. But the **run** method, called on the **calc** object is **not static**. **run()** is where all the application program will go.

**All** programs in this course, assignment or lab, are expected to be written using this architecture: *construct* an application object and then call the **run()** method with the new object.

In order to compile the program you need to implement **run()**. Why not make it print "Hello, World!"?

## Compiling with bclMenu.jar

You can compile and run your application, so long as you tell **javac** and **java** where it lives:

```
pUsingMenu $ javac driver/Calculator.java
```

The command assumes you are in the directory *containing driver* (and *data* and *lib*, too). By default, `javac` includes the current directory in the list it searches for files so `driver/Calculator.java` is found in `./driver/Calculator.java` and the `.class` file is in the same folder. So, to run it, use:

```
pUsingMenu $ java driver.Calculator
Hello, World!
```

The directory separator becomes the Java package separator (dot) since you are naming the *class* to run for *java*.

### Finding BCLMenu

At the top of your program, add two **import** lines (remember: the **first** line is the **package** line; you will add any necessary **import** lines *after* that):

```
import menu.Menu;
import menu.BCLMenu;
```

Add a *Menu field* to your calculator. Initialize its value in the constructor to a `BCLMenu` constructed with a `FileInputStream` wrapped around `data/calculator.menu` (the description in the *data* directory you copied over from the assignment repo).

In *run*, replace printing "Hello, World!" with a call to `Menu.display()` on your newly constructed menu.

To compile and run with the library, use the following:

```
javac --class-path ../lib/bclMenu.jar driver/Calculator.java

java --class-path ../lib/bclMenu.jar driver.Calculator
```

Assuming the data file was found, it should print the following:

```
add - add two numbers
subtract - subtract two numbers
multiply - multiply two numbers
divide - divide two numbers
quit - quit the program
```

It will then terminate because `display` just prints the menu entries.

**Suggestion:** Add and commit your updated, working code to `git`. This makes sure you can roll back to this point if things break.

### Using `Menu.match()`

You can see the **interface** that `BCLMenu` implements in `src/Menu.java` in the assignment repo. You can see all the methods it implements.

`Menu.match` is the secret sauce. It is called with a prompt (to print for the user at the bottom of the menu) and a `Scanner` for input from the user. The prompt is up to you. You should make a "keyboard" `Scanner` using `System.in`:

```
Scanner k = new Scanner(System.in);
```

and pass that scanner into `match`.

What does `match` do? It lets the user type in commands. If the command *matches* one of the command parts of a menu entry, the command is returned. If it does not match any entry, the method loops, prompting the user again.

This means that when you call `match`, it will always return one of the commands in the menu. For `calculator.menu`,

```
add
add two numbers
prompts for two INTEGERS, first and second, and displays their sum
subtract
subtract two numbers
prompts for two INTEGERS, first and second, and displays their difference
multiply
multiply two numbers
prompts for two INTEGERS, first and second, and displays their product
divide
divide two numbers
prompts for two INTEGERS, first and second, and displays their quotient
quit
quit the program
halts the running program
```

the valid commands are `add`, `subtract`, `multiply`, `divide`, `quit` .

`match` does accept one other “command”: if the user types `?` on a line, then the menu is displayed along with the help lines (the long third lines of each entry).

## `run()`

The final `run` method will do something like the following:

```
while (!done)
    String command = match

    if (command.equals("quit"))
        done = true
    else if (command.equals("add"))
        // call the add function
    else if (command.equals("subtract"))
        ...
```

## Input/Output

### File Formats

A Menu data file writes the three fields of a menu entry on three separate lines as in

```
<command>
<description>
<help text>
```

Such a file should have some multiple of three lines. `BCLMenu` has a constructor that takes a `Scanner` reading such a data source and a second constructor that takes an `InputStream` on such a data source.

The provided data file, `data/calculator.menu`, looks like the following:

```
add
add two numbers
prompts for two INTEGERS, first and second, and displays their sum
subtract
subtract two numbers
prompts for two INTEGERS, first and second, and displays their difference
multiply
multiply two numbers
prompts for two INTEGERS, first and second, and displays their product
divide
divide two numbers
prompts for two INTEGERS, first and second, and displays their quotient
quit
quit the program
halts the running program
```

## Menu Display

When `calculator.menu` is displayed, it looks like

```
add - add two numbers
subtract - subtract two numbers
multiply - multiply two numbers
divide - divide two numbers
quit - quit the program
```

## Help Display

When `calculator.menu` users ask for help, the menu looks like

```
    add - add two numbers
          prompts for two INTEGERS, first and second, and displays their sum
subtract - subtract two numbers
          prompts for two INTEGERS, first and second, and displays their difference
multiply - multiply two numbers
          prompts for two INTEGERS, first and second, and displays their product
    divide - divide two numbers
          prompts for two INTEGERS, first and second, and displays their quotient
    quit - quit the program
          halts the running program
```

## Documentation

### README

Must document how you tested. How do you know that it is right?

Must include instructions on how to **compile** and how to **run** the program as submitted. You can test these if you **clone** the repository you submitted into a **brand new location**. You can then follow the instructions and see if all the necessary files are in the repo.

## Deliverables

**Submission medium:** git to Gitea at `cs-devel.potsdam.edu`. The repo name is `p001-UsingMenu` and it goes in your organization for turning in assignments in CIS 205 this semester.