Lab 4c

CSC-121, Fall 2025

Group Exercises

The Partner Exercise is *very* long this week as it goes over skills you will need for Project #2. It will likely take the entire lab time to complete. If you finish the partner section exercise, go back to Lab 4b and cover any exercises with the TAs that you did not get to last time.

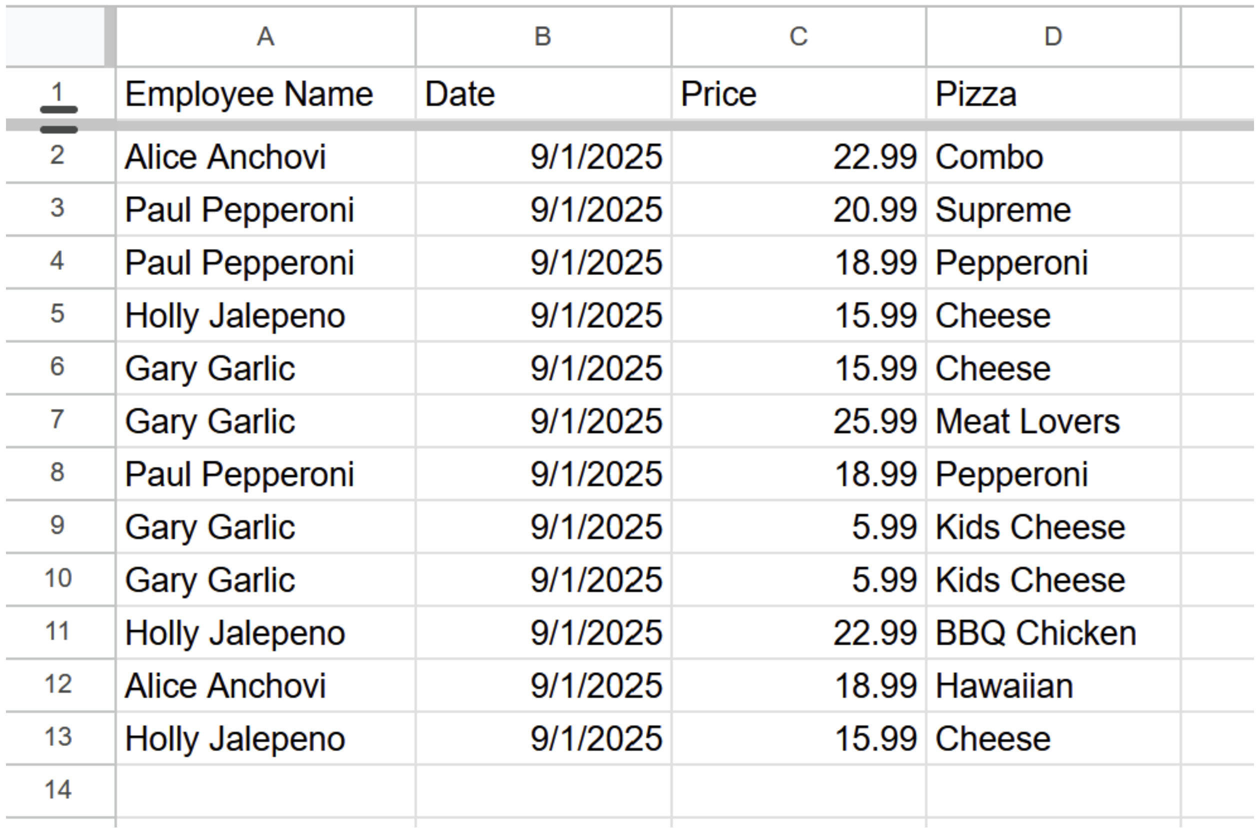
Partner Exercises

Work together with a partner to build the following programs.

Q1. Pizza Time

Today, we are going to learn how to read and parse data from a CSV file! The hip new pizza place, Pizzacute (Named after the geometric angle of pizza slices) tracks the sales made by each of its employees. The cafe has given you a CSV file containing one day of their lunch sales data. Pizzacute wants you to write a program that can tally and report the total sales for the day, and also report the total sales of an individual employee when asked via the console.

CSV stands for "Comma Separated Values" and is often used to represent data from a spreadsheet. The CSV sales file contains the following information:

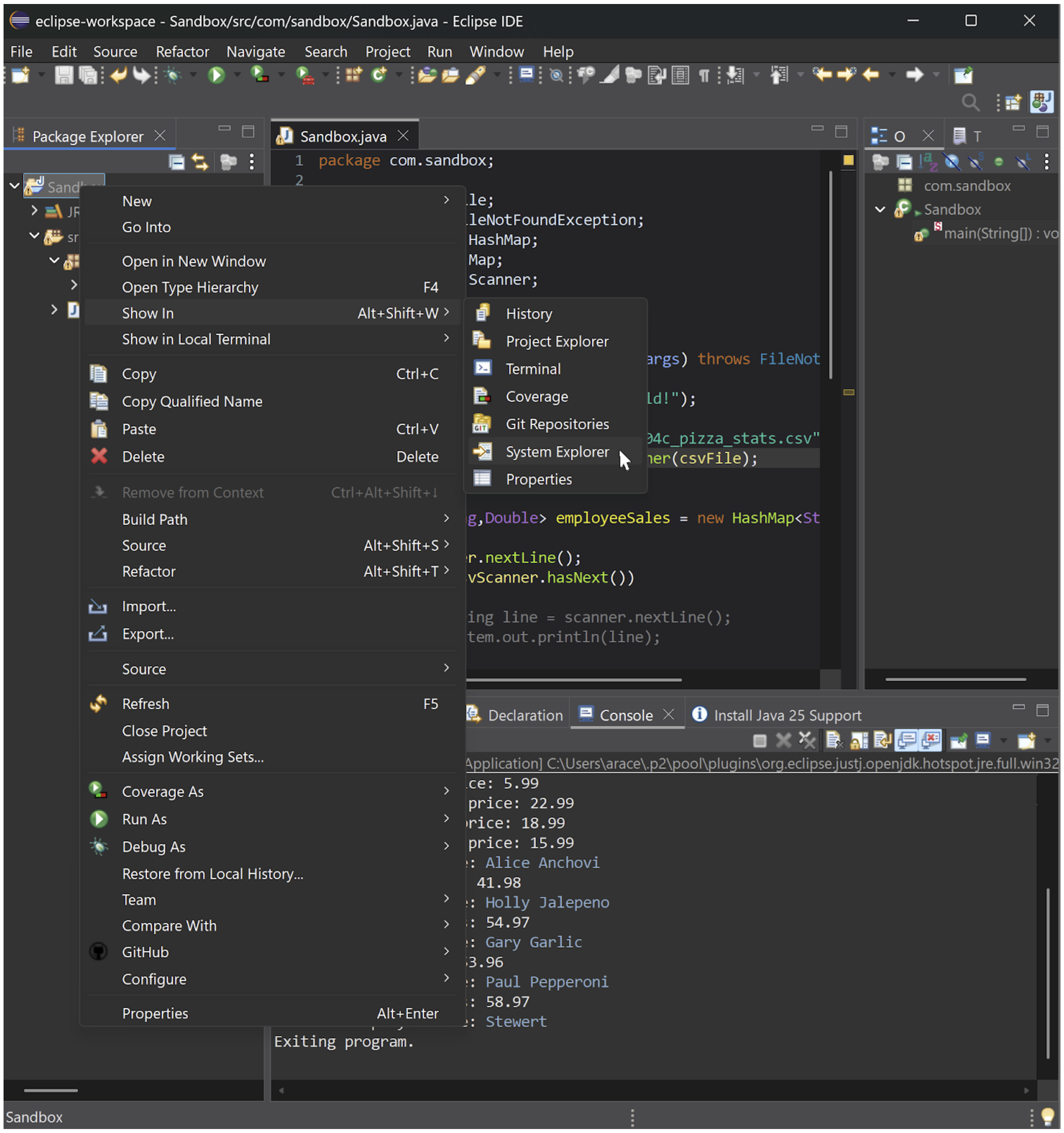


	A	B	C	D
1	Employee Name	Date	Price	Pizza
2	Alice Anchovi	9/1/2025	22.99	Combo
3	Paul Pepperoni	9/1/2025	20.99	Supreme
4	Paul Pepperoni	9/1/2025	18.99	Pepperoni
5	Holly Jalepeno	9/1/2025	15.99	Cheese
6	Gary Garlic	9/1/2025	15.99	Cheese
7	Gary Garlic	9/1/2025	25.99	Meat Lovers
8	Paul Pepperoni	9/1/2025	18.99	Pepperoni
9	Gary Garlic	9/1/2025	5.99	Kids Cheese
10	Gary Garlic	9/1/2025	5.99	Kids Cheese
11	Holly Jalepeno	9/1/2025	22.99	BBQ Chicken
12	Alice Anchovi	9/1/2025	18.99	Hawaiian
13	Holly Jalepeno	9/1/2025	15.99	Cheese
14				

Spreadsheets are grids that represent data as a set of rows and columns. Each row represents an individual pizza sale, and each column represents a particular feature of the sale. There are columns for the name of the employee that made the sale, the date of the sale, the price of the sale, and the type of pizza that was sold. The sales in this file represent lunch sales from September 1st, 2025.

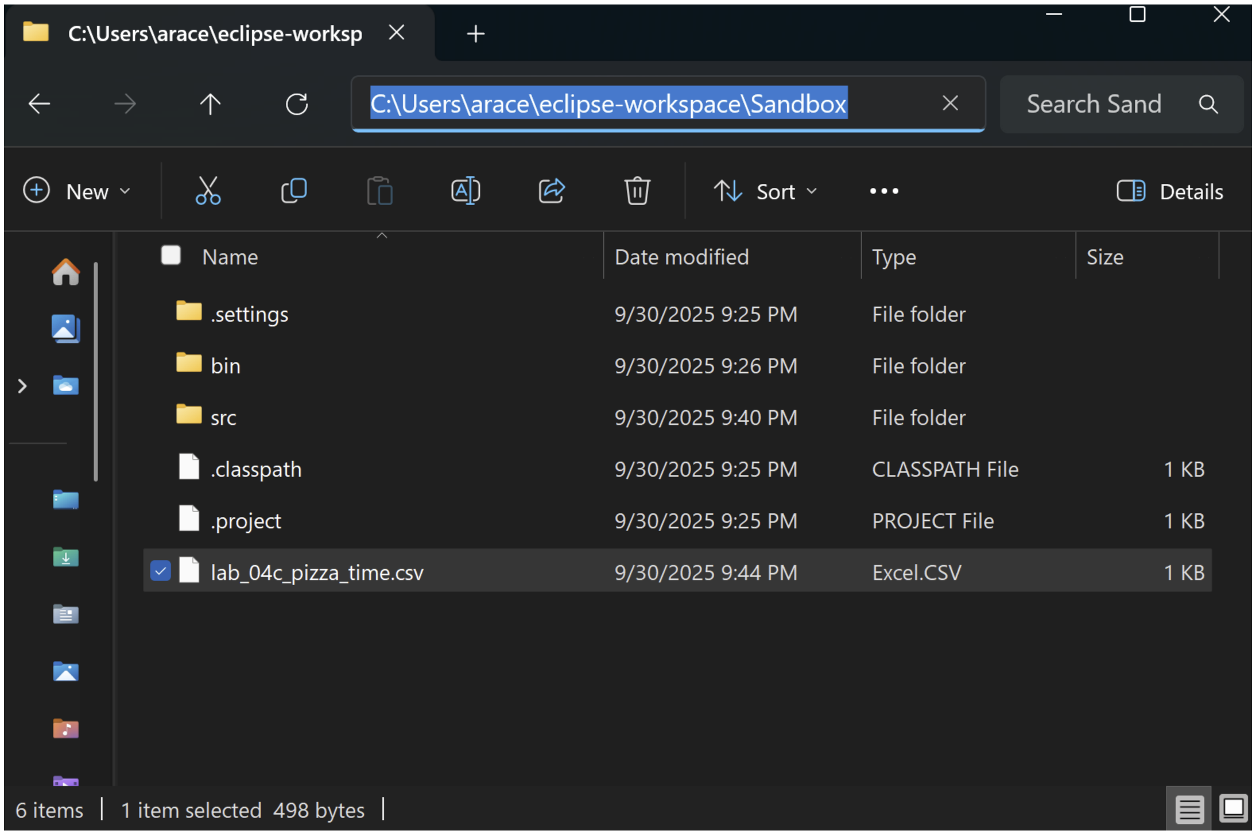
When viewed in a spreadsheet program like Google Sheets, Microsoft Excel, or LibreOffice Calc, the information will be displayed in a grid like the previous image with proper rows and columns. However, if you open the CSV file in a text editor like Notepad or TextEdit, it will look much different - Try it now. What do you observe? Does the name CSV seem appropriate?

In order to make the sales program, we'll need to first load the CSV into our program. First, let's add the CSV file to our project. Right-click your code project in Eclipse and choose Show In -> System Explorer. You should see a file explorer window open with a number of folders corresponding to the names of your Eclipse projects. Copy the "lab_04c_pizza_time.csv" file into the folder matching your lab project's name.



```
package com.sandbox;

le;
leNotFoundException;
HashMap;
Map;
Scanner;

args) throws FileNot
ld!");
04c_pizza_stats.csv"
ner(csvFile);
g,Double> employeeSales = new HashMap<St
r.nextLine();
vScanner.hasNext())
ing line = scanner.nextLine();
tem.out.println(line);
```

```
Application] C:\Users\arace\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
ce: 5.99
price: 22.99
rice: 18.99
price: 15.99
: Alice Anchovi
 41.98
: Holly Jalepeno
: 54.97
: Gary Garlic
3.96
: Paul Pepperoni
: 58.97
: Stewert
Exiting program.
```


C:\Users\arace\eclipse-worksp
C:\Users\arace\eclipse-workspace\Sandbox
Search Sand
New
Sort
Details

Name	Date modified	Type	Size
.settings	9/30/2025 9:25 PM	File folder	
bin	9/30/2025 9:26 PM	File folder	
src	9/30/2025 9:40 PM	File folder	
.classpath	9/30/2025 9:25 PM	CLASSPATH File	1 KB
.project	9/30/2025 9:25 PM	PROJECT File	1 KB
lab_04c_pizza_time.csv	9/30/2025 9:44 PM	Excel.CSV	1 KB

6 items | 1 item selected 498 bytes |

Your CSV file should now be in the project directory along with the src and bin folders like in the image above.

Now we can get to coding. Like we saw in lecture, we'll use the File class to load the file.

```java
File csvFile = new File("lab_04c_pizza_time.csv");
```

We've loaded the file, but we still need some other tool to actually read the contents. We're familiar with the Scanner class, so let's use that.

```java
File csvFile = new File("lab_04c_pizza_time.csv");

Scanner csvScanner = new Scanner(csvFile);
```

Before we would create a Scanner by giving it System.in as an argument. That meant it would read from Standard Input, I.E. the console. This time, we gave it the file object containing our CSV as an argument. Our Scanner will behave exactly the same way as before, except now when we use the Scanner's "next" methods, we will be getting that information from the csv file instead of Standard Input! The Scanner object will read the file from top-to-bottom, left-to-right, just like if you were reading the text yourself. Let's test it by printing out the contents of the file with the Scanner's nextLine method.

```java
while (true)

{

      String line = csvScanner.nextLine();

      System.out.println(line);

}
```

Let's break this down. First, we have a while loop with the loop condition set to true. Since this loop condition is always true, our loop will run forever. Let's leave this as it is for now.

Inside the while loop, we ask the csvScanner to give us the next line from the file, and then we print the line out to the console on Standard Output. This loop will continue printing lines from the file until we have read the last line. Try running this code and see what happens.

You should see the entire contents of the CSV printed to the console, followed by an exception for "No line found". What do you think happened?

We need to tell our loop to stop when we have read the entire file. Conveniently, the Scanner has a method that can help us with that. Let's look at the Java Documentation for the Scanner class, and find the "hasNext" method. What does it say?

Let's try using the hasNext method in our code as the loop condition.

```java
while (csvScanner.hasNext())

{

      String line = csvScanner.nextLine();

      System.out.println(line);

}

csvScanner.close();
```

Let's break this down again. The condition for the while loop now asks the csvScanner object if it has any more input available. This will return true *until* we have read all the way to the bottom of our file, at which point it will return false. We now have a way to stop reading from the file when we reach the end! Since we can actually exit the loop now, we also add a line to close our csvScanner since we are done with it.

It's great that we can read the file now, but we can't really do much else with it yet. All of the columns are scrunched up together on the same line. We were asked to sum up the sales for each employee, but how can we get the sales info separated from everything else?

Rather than reading the whole line all at once, we need to read in the individual pieces of each line that are actually important to us. This is where the 'C' in CSV becomes very useful. Notice how each column feature of our data is separated with commas? We can use those commas as markers to tell us where each field starts and ends.

We're going to break each line down into individual "tokens" based on the positions of the commas. Then, we can read each token one-by-one. Since we know the order of the columns, we know what order the data is in and we can parse each token into individual variables.

To separate the input into tokens, we have to tell our csvScanner to use a "delimiter". From the Scanner documentation:

"*A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.*"

This sounds like exactly what we need! We're going to tell our csvScanner to use a comma as its delimiter, and we'll try reading in the next token instead of the next line. Let's try this.

```java
csvScanner.useDelimiter(",");

while (csvScanner.hasNext())

{

      String token = csvScanner.next();

      System.out.println(token);

}
```

Now, instead of printing the whole line of information, we are printing each individual token!

Now that we have tokenized the input, we need to actually do the tasks that the cafe asked from us. We need to print out the total sales for the day. To do this, we need to add up all of the sales data from the "Price" column. We can do this by picking out those column entries, parsing them as numeric values, and adding them. Let's try that, while also printing the employee name and price for each line as we go.

```java
double totalSales = 0.0;

csvScanner.useDelimiter(",");

while (csvScanner.hasNext())

{

    String name = csvScanner.next();

    String date = csvScanner.next();

    double price = Double.parseDouble(csvScanner.next());

    String pizzaType = csvScanner.next();


    totalSales += price;

    System.out.printf("Name: %s, price: $%.2f\n", name, price);

}

csvScanner.close();

System.out.printf("Total sales: $%.2f\n", totalSales);
```

It turns out there are a few minor problems with this code. Let's examine and fix them one-by-one. First, the very first row of the CSV file contains the names of the columns rather than actual data. We can fix this by adding a csvScanner.nextLine() call just before we enter our loop. Try that and see what happens.

Next, we can print the first line okay but then hit an error. The name of the first pizza, "Supreme," runs together with the name of the next employee, "Paul Pepperoni". Why do you think this is happening? Look at the CSV in a text editor and think about it before you turn to the next page.

The reason our data is running together is because there is no comma separator at the end of our row. There are two ways we can fix this. The first option is to open our CSV and add commas to the end of every row. This would fix the problem, but is a bit tedious. And what if we had 10,000 rows of data? The other option is to change our delimiter to look for commas *and* newlines. Let's see how that would work.

```java
csvScanner.useDelimiter(",|\\r?\\n");
```

We still have the comma delimiter, but now there are some other symbols. Let's break it down.

```
scanner.useDelimiter(",|\\r?\\n");
```

- `,` → Matches commas
- `|` → Means “or”
- `\\r?\\n` → Matches both Unix/Mac style newlines (`\n`) *and* Windows style (`\r\n`). Also, the `?` in the middle means "Zero or one occurrence" for the `\r` symbol, so `\r` will be used as part of the delimiter if it is present.
- Why the double slashes, like `\\n` instead of just `\n`?
  Without getting too technical, we are writing something called a "Regular Expression" rather than just a normal String. Regular expressions are used for matching patterns in String data, like finding our delimiters. We won't get much deeper into regular expressions in this class, just know that we need to treat regular expressions a little differently as a result.

Let's try our code again with these fixes.

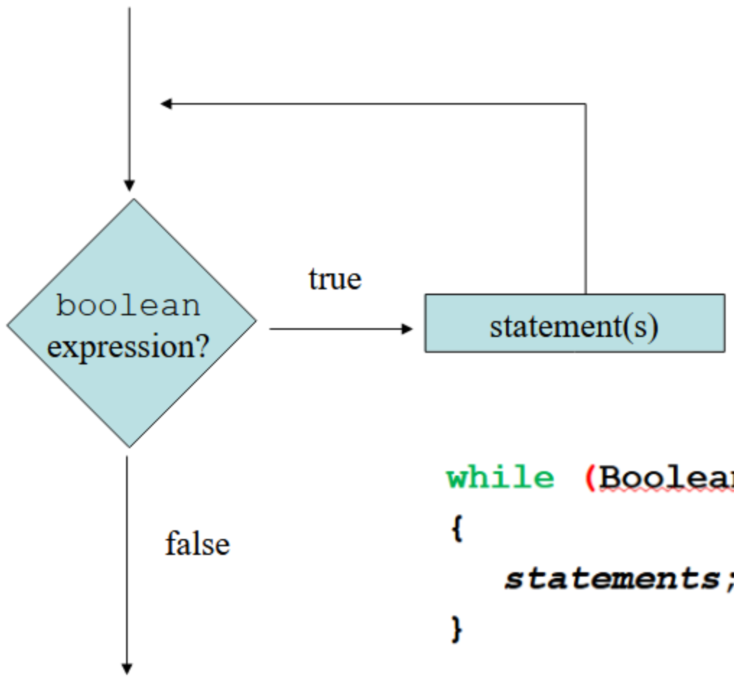
```java
double totalSales = 0.0;

csvScanner.useDelimiter(",|\\r?\\n");

csvScanner.nextLine(); // Skip the first row of column names.

while (csvScanner.hasNext())

{

    String name = csvScanner.next();

    String date = csvScanner.next();

    double price = Double.parseDouble(csvScanner.next());

    String pizzaType = csvScanner.next();


    totalSales += price;

    System.out.printf("Name: %s, price: $%.2f\n", name, price);

}

csvScanner.close();

System.out.printf("Total sales: $%.2f\n", totalSales);
```

If you got that the total sales was $209.88, then congratulations - You just correctly read in the sales data from the CSV and printed the total sales!

Now, the second task asks us to sum up the total sales for each employee. There are a number of ways we can go about this, like creating totalSales variables for each employee. What would be helpful is if we had a way to associate each employee with a value in a more natural way. It would be really great if we had something like our Dictionary data types from Python…

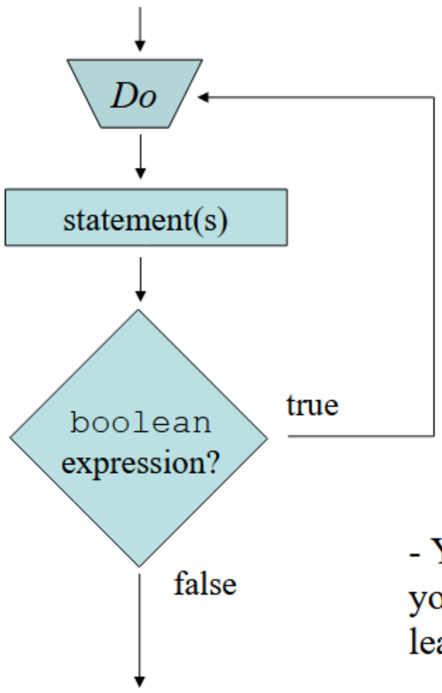
We've done A LOT in this lab already, so we'll tackle the second problem next week!

The while loop Flowchart



```
while (BooleanExpression)
{
   statements;
}
```

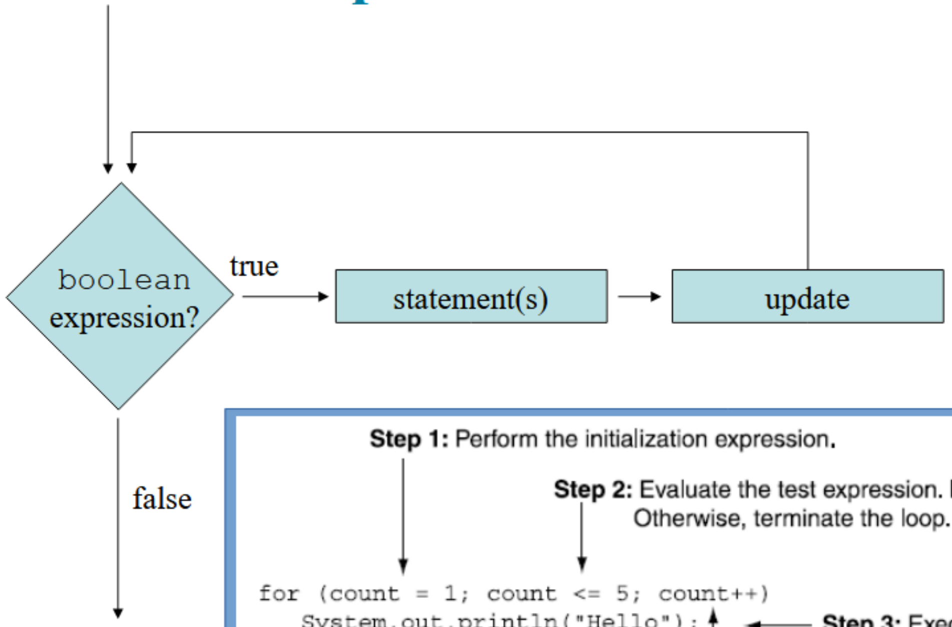
The do-while Loop Flowchart

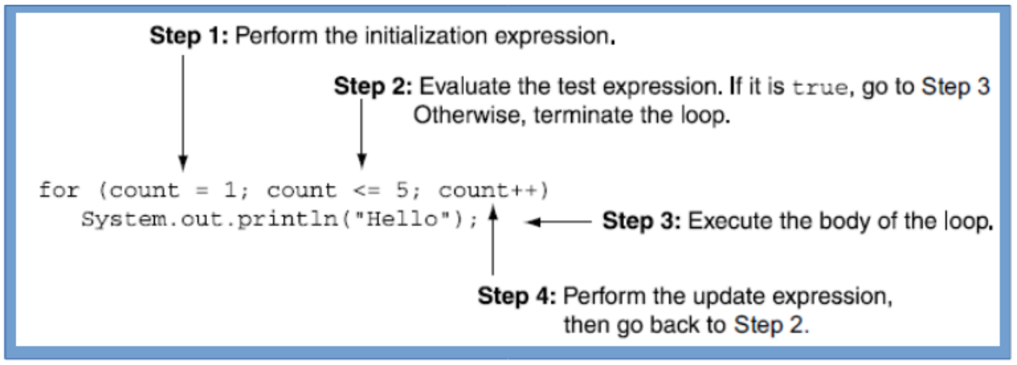


```
int x = 1;
do
{
     System.out.println(x);
} while (x < 0);
```

- You should use the do-while loop when you want to make sure the loop executes at least once.

The `for` Loop Flowchart




Step 1: Perform the initialization expression.
Step 2: Evaluate the test expression. If it is true, go to Step 3 Otherwise, terminate the loop.
Step 3: Execute the body of the loop.
Step 4: Perform the update expression, then go back to Step 2.

```
for (count = 1; count <= 5; count++)
   System.out.println("Hello");
```

Individual Tasks

1. Project Check-In

Have you started your Project **#2**, yet? ***Don't wait until the last minute!*** You should present your plan to one of the Tech Fellows and get feedback. Remember, part of your grade is based on doing the check-in.

2. CodePath Signup

(*NOTE: If you completed the CodePath signup in the previous lab, skip this block.*) As stated on the first day of class, this course is presented in cooperation with CodePath.org. You may have already gotten an email from CodePath to take their pre-assessment test. Below is a link to sign up for the CodePath in Residence perks, such as career counseling, resume and portfolio reviews, and other services. These are *free* services provided to any CodePath course student or alumni. Signing up is not required, but strongly encouraged. You will also need a free GitHub account in order to sign up. You can use the remaining time in lab to sign up, or talk to a TA about your project or the homework.

https://go.codepath.org/CiRFA25

3. CodePath Pre-Assessment

Check your email! CodePath has sent out a pre-assessment Computer Science test to all enrolled students. Don't worry, you are not graded on this! CodePath uses these assessments to track student progress and improvement throughout the course. Just answer the questions as best as you can, and it is okay if you don't know everything. You'll need to set aside about 30 minutes for the assessment, either here in lab time or at home.