Unit 2: Working with Objects

Topic 2 Lab 2: Partner Programming Challenges!

Name:	
Partner Name:	

This is a **pair programming** lab: you and your partner should code together in the **same** Replit project, just like if you were both typing in a shared Google Doc!

Setup

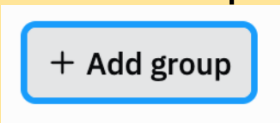
Follow the instructions below to create a collaborative Replit for you and your partner.

Choose **ONE** partner to:

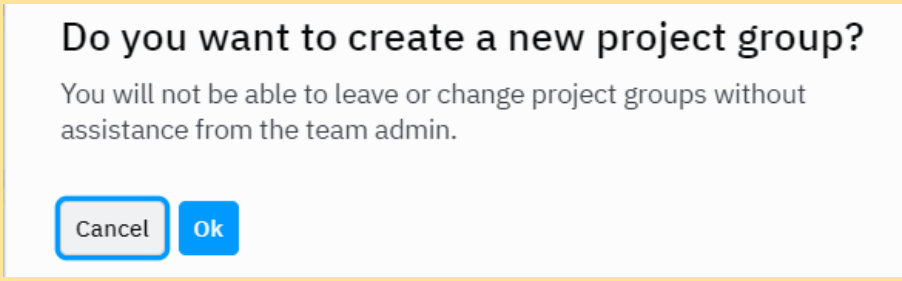
A. Open up the **U2T2 Lab 2: Partner Challenges** team project.

B. Click: Select project group on the right

C. Click **Add Group**:



D. Click **OK**:



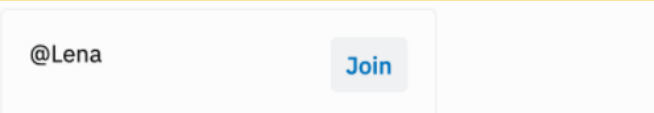
E. Click **Confirm** to allow notifications (if this pop up appears).

After the first partner completes the steps above, the **OTHER** Partner should:

A. Open up the **U2T2 Lab 2: Partner Challenges** team project.

B. Click: Select project group on the right

C. Locate and **Join** the group *started by your partner!* Make sure you select the right one (if you accidentally join the wrong group, let Mr. Miller know):



Once both partners are in, both partners will be coding alongside each other!

Partner Warm Up!

Create a Cat class (filename: Cat.java) and copy/paste this code:

```
public class Cat {
    private String name;

    public Cat(String name) {
        this.name = name;
    }

    public void feed(String food) {
        // write me!
    }

    public void introduce() {
        // write me!
    }
}
```

1. Write the `feed` method to print a statement like *"Yum! ___ loves ___!"* where the cat's name appears in the first blank and the food appears in the second blank.

Test your method by making a Cat object in the main method of your Main class, give the Cat a name of "**Fluffy**", then call the `feed` method, passing in "**tuna**" as the parameter.

The output should be:

```
Yum! Fluffy loves tuna!
```

2. Write the `introduce` method to print a statement like *"Hello, my name is ___!"* where the cat's name appears in the blank

Test your method by calling the `introduce` method on the same object you created above.

The output should be:

```
Hello, my name is Fluffy
```

3. Try modifying your introduce method to look like this, adding the food to the end of the printed string:

```java
public void introduce() {
    System.out.println("Hello, my name is " + name + " and I love " + food);
}
```

Run your code; what is the error and why does it occur?

Notice that the instance variable `name` can be used in *both* methods, whereas the parameter `food` can *only be used in the* `feed` method. What conclusion can you draw about instance variables vs. parameters in terms of which methods can use/access them?

Remove the broken code from step 3, then copy/paste your **Cat** code:

Game Time! Partner Challenge 1

1. Add a new Game.java file then copy/paste this incomplete Game class.

2. Take a look at the `Game` class constructor; notice that it has two parameters, and in the body of the constructor's code, one of the instance variables (`score`) is set to a default value of 0.

3. Your team's task is to complete several missing methods of the `Game` class, detailed below. Find the place in the `Game` class where you should complete each task by looking for the specified comment, such as: `/* TO BE IMPLEMENTED IN PART A */`

- **A.** Complete the missing code for the `getPlayers()` "getter method" that returns the value of the `players` instance variable.
- **B.** Add a `getScore()` "getter" method for the score instance variable.
- **C.** Complete the missing code for the `addPlayer()` method that increases `players` by 1; note that this method is `void` (it has no return value).
- **D.** Add an `increaseScore` method that has one `int` parameter (name it "`increase`") and increases `score` the amount of `increase`; the method should *not* return a value (i.e. make it `void`).
- **E.** Add an `averageScorePerPlayer` method that has no parameters and returns the *average score per player* as a `double`; for example, if `players` is 4 players and `score` is 11, this method should return 2.75.
- **F.** Complete the missing code for the `isGameOver` method so that `true` gets returned if `score` is greater than 9, otherwise `false` is returned.

4. TESTING!
Test your Game class by copying/pasting this code into your Main class' main method and running.

Expected output ***Make sure your program output matches the following exactly!***

```
Game name: Dodge Ball
Players: 5
Score: 0
Is game over? false
----- UPDATING STATE OF GAME -----
Game name: Dodge Ball
Players: 8
Score: 11
Avg score per player: 1.375
Is game over? true
```

5. Copy/paste your **completed and tested** Game class below:

6. Free Style! Remove and add some different test code in `Main;` create a new `Game` object (for example a game of Checkers with 2 players, but do anything you want!), add some players, increment the score a few times, print out the average score, etc. **Try some things out!**

Play around with the code of `Game` class too. Add or edit existing methods, change methods, test it out. See what you can come up with and what you figure out!

Copy/paste the test code you added to `Main`:

Copy/paste your freestyled Game class:

Briefly describe what you added, changed, and figured out!	

If you and your partner want to, feel free to compare your completed Game class to this sample

LAB CONTINUES ON NEXT PAGE

Letter Writer! Partner Challenge 2

1. Add a new `Letter` class to your Replit project, and **copy/paste this code**.

2. Clear out the code in your Main class' main method.

3. In the main method, create a `Letter` object (name the variable whatever you want), and call *each of the three methods* to print a letter:

- start with the greeting method
- then the special message method
- then the closing method

Note: the `Letter` class only has a *no-parameter* constructor, so to create a `Letter` object, you need to do this: `new Letter()`

Expected output:

```
Hello, friend!
Java is pretty cool, wouldn't you say?
See you later!
```

Let us double check our code

4. In the `Letter` class, locate `/* TO BE IMPLEMENTED IN PART 4 */` and add a **new** `writeLetter` method that, when called, prints the *same* output as calling the three separate methods. It should not return a value. ***Do not*** *just type three identical print statements! Instead, you should* ***call the three methods that already exist in the `Letter` class****!* (Never create duplicate functionality -- use what already exists!)

5. TEST your solution by ***replacing*** the following lines in your client class:

```
Letter myLetter = new Letter();
myLetter.greeting();
myLetter.specialMessage();
myLetter.closing();
```

With *these*:

```
Letter myLetter = new Letter();
myLetter.writeLetter();
```

Run it and check that you get the *same* output as calling the three methods separately:

```
Hello, friend!
Java is pretty cool, wouldn't you say?
See you later!
```

Copy/paste your new `writeMethod` method below:

Compare our solution

6. Let's update the `Letter` class so that the person who is writing the letter can store their name and that name gets printed as part of the closing; here's how:

A. Add a **new instance variable** of type `String` named "`from` to the `Letter` class.

B. Next, add a single `String` parameter to the constructor (name the parameter whatever you want) which allows the client to provide the name of the person writing the letter.

C. Lastly, update the `closing()` method to print a *second* line saying: **From, ____** with the `from` instance variable inserted. See test below for an example.

7. TEST your solution by ***replacing*** the following lines in your client class:

```
Letter myLetter = new Letter();
myLetter.writeLetter();
```

With *these* (feel free to use your name!):

```
Letter myLetter = new Letter("Mr. Miller");
myLetter.writeLetter();
```

Expected output:

```
Hello, friend!
Java is pretty cool, wouldn't you say?
See you later!
From, Mr. Miller
```

Copy/paste your updated Letter class below (you should have made three changes!):

confirm your changes here

8. Now, modify the `writeLetter()` method by adding a `String` parameter named `toName` to take in the name of the person the letter is going to.

9. Lastly, modify the `greeting()` method in order to print the `toName` name as part of the greeting; rather than "Hello, friend!" print "Hello, _____!"). Do this by *also* adding a String parameter to the `greeting` method and "passing through" the `toName` value *from* the `writeLetter` method *to* the `greeting` method.

10. TEST your solution by ***replacing*** the following lines in your client class:

```
Letter myLetter = new Letter("Mr. Miller");
myLetter.writeLetter();
```

With *these*:

```
// create Letter object
Letter myLetter = new Letter("Mr. Miller");

// write letters to various people
myLetter.writeLetter("Abdul");
System.out.println();

myLetter.writeLetter("Pritika");
System.out.println();

myLetter.writeLetter("Ethan");
System.out.println();

myLetter.writeLetter("Lucy");
```

Expected output:

CONSOLE SHELL

```
Hello, Abdul
Java is pretty cool, wouldn't you say?
See you later!
From, Mr. Miller

Hello, Pritika
Java is pretty cool, wouldn't you say?
See you later!
From, Mr. Miller

Hello, Ethan
Java is pretty cool, wouldn't you say?
See you later!
From, Mr. Miller

Hello, Lucy
Java is pretty cool, wouldn't you say?
See you later!
From, Mr. Miller
→
```

Copy/paste the code of your updated `Letter` class below:

Compare our implementation

In case you were wondering, we *could* have instead updated the `Letter` class to have *both* `from` *and* `toName` as *instance variables* and to accept both values in the constructor, like this:
 	
`Letter myLetter = new Letter("Mr. Miller", "Abdul");`
`myLetter.writeLetter();`
 	
In this case, we could have kept the `writeLetter` and `greeting` methods without parameters, and print the *instance variable's* toName value in the greeting method, rather than passing it as a parameter.
 	
Why might the way we did it (*with* the parameter) be a better approach for this situation?
	There is no right or wrong answer here! Just type what you think:

In case you are curious, here are Mr. Miller's thoughts (for what it's worth!)

Student Tracker! Partner Challenge 3

Add a new Student.java file then write the complete `Student` class as described below.

You are going to write the `Student` class such that a client program can store a student's full name (first and last) and graduation year (e.g. 2021), and add test scores, one at a time. At any time, the client program should be able to obtain a student's average test score, whether they are passing, or print basic info about the student, such as full name, graduation year, current test average, and number of accumulated tests.

Here are the instance variables, constructor, and methods you should include in your class:

```
public class Student

/* Instance Variables */
private String firstName
private String lastName
private int gradYear
private double accumulatedTestScores
private int testScoreCount

/* Constructor; see Note 1 below */
public Student(String firstName, String lastName, int gradYear)

/* Getter Methods */
// Returns firstName
public String getFirstName()

// Returns lastName
public String getLastName()

/* Setter Methods */
// Sets gradYear to newGradYear
public void setGradYear(int newGradYear)

/* All Other Methods */
// Adds newTestScore to accumulatedTestScores
// and increments testScoreCount by 1
public void addTestScore(double newTestScore)

// Returns true if the student's average test score is greater
// than or equal to 65; returns false otherwise (see Note 2 below)
public boolean isPassing()

// Returns the Student's average test score as the
// quotient of accumulatedTestScores and testScoreCount
public double averageTestScore()

// this method prints the students full name, grad year, test average,
// and whether they are passing (see Note 3 below)
```

```
public void printStudentInfo()
```

Note 1: The constructor has three parameters rather than five; this is because it makes sense to set the `accumulatedTestScores` instance variable to a default value of 0.0 and `testScoreCount` to 0, since both get added to over time, but should start at 0. So in the constructor, be sure to initialize these two values as well, using the appropriate default values. Also, since the parameter names match the instance variable names, don't forget to use `this` when initializing, e.g. `this.firstName = firstName;`

Note 2: Since this method requires the average test score, use your `averageTestScore` method to obtain this value -- do **NOT** use math to calculate the average again when you have a method that does it! Also, the Java comparison operator for **greater than or equal to** is: **>=**

Note 3: Similarly, this method also requires the average test score, so again, use your `averageTestScore` method to obtain this value. The same goes for `isPassing`; use the `isPassing` method's return value rather than writing code in the `printStudentInfo` method to redo that logic.

TESTING! HERE IS TEST CODE for you to copy/paste into your `Main` class:

```
Student student1 = new Student("Charles", "Smith", 2023);
student1.addTestScore(85.5);
student1.printStudentInfo();
System.out.println();

student1.addTestScore(94);
student1.printStudentInfo();
System.out.println();

student1.addTestScore(95);
student1.printStudentInfo();
System.out.println();

Student student2 = new Student("Amy", "Adams", 2022);
student2.addTestScore(68.2);
student2.printStudentInfo();
System.out.println();

student2.addTestScore(57.5);
student2.printStudentInfo();
System.out.println();

student2.setGradYear(2023);
student2.printStudentInfo();
System.out.println();

double student1avg = student1.averageTestScore();
double student2avg = student2.averageTestScore();
String student1name = student1.getFirstName() + " " + student1.getLastName();
String student2name = student2.getFirstName() + " " + student2.getLastName();

if (student1avg > student2avg)
{
    System.out.println(student1name + " has a higher average!");
}
else if (student2avg > student1avg)
{
```

```
    System.out.println(student2name + " has a higher average!");
}
else
{
    System.out.println(student1name + " and " + student2name + " have equal averages");
}
```

Expected output (compare yours carefully -- it should match this exactly!)

CONSOLE SHELL

```
Student Full Name: Charles Smith
Graduation Year: 2023
Number of tests: 1
Average Test Score: 85.5
Is passing: true

Student Full Name: Charles Smith
Graduation Year: 2023
Number of tests: 2
Average Test Score: 89.75
Is passing: true

Student Full Name: Charles Smith
Graduation Year: 2023
Number of tests: 3
Average Test Score: 91.5
Is passing: true

Student Full Name: Amy Adams
Graduation Year: 2022
Number of tests: 1
Average Test Score: 68.2
Is passing: true

Student Full Name: Amy Adams
Graduation Year: 2022
Number of tests: 2
Average Test Score: 62.85
Is passing: false

Student Full Name: Amy Adams
Graduation Year: 2023
Number of tests: 2
Average Test Score: 62.85
Is passing: false

Charles Smith has a higher average!
```

Copy/paste your complete and tested `Student` class below:

A sample solution for Problem 3 will be posted by the end of the period.

Sample Game implementation (back):

```java
public class Game {
    // instance variables (a.k.a. fields, properties, attributes, "state", data)
    private String gameName;
    private int players;
    private int score;

    // constructor
    public Game(String gameName, int players) {
        this.gameName = gameName;
        this.players = players;
        score = 0; // default value
    }

    // returns the name of the game
    public String getGameName() {
        return gameName;
    }

    // returns the number of players
    public int getPlayers() {
        return players;
    }

    // returns the current score
    public int getScore() {
        return score;
    }

    // increments the number of players by 1; this method has no return value (void)
    public void addPlayer() {
        players++;
    }

    // increments the game's score by the value of an int parameter named "increase";
    // this method has no return value (void)
    public void increaseScore(int increase) {
        score += increase;
    }

    // calcuates and returns the average score per player as a double
    public double averageScorePerPlayer() {
        double average = (double) score / players;
        return average;
    }
```

```java

    // returns true if score > 9, otherwise returns false
    public boolean isGameOver() {
        if (score > 9) {
            return true;
        } else {
            return false;
        }
    }
}
```

(back)

Compare (back)

Mr. Miller used “`myLetter`” as the variable name, but you can use any variable name you want:

Main.java

```java
public class Main {
    public static void main(String[] args) {
        // create Letter object
        Letter myLetter = new Letter();

        // call the three methods in correct order
        myLetter.greeting();
        myLetter.specialMessage();
        myLetter.closing();
    }
}
```

Sample solution (back)

Letter.java

```java
public class Letter {
    // instance variables
    /* none yet! */

    // constructor; since there are no instance variables to
    // initialize, this constructor has no parameters and is "empty".
    // to call this no-parameter constructor use: new Letter();
    public Letter() { }

    public void writeLetter() {
        greeting();
        specialMessage();
        closing();
    }

    public void greeting() {
        System.out.println("Hello, friend!");
    }

    public void specialMessage() {
        System.out.println("Java is pretty cool, wouldn't you say?");
    }

    public void closing() {
        System.out.println("See you later!");
    }
}
```

Confirm (back)

Three changes are needed, as outlined in steps A, B, and C:

Letter.java

```java
public class Letter {
    // instance variables
    private String from;

    // constructor
    public Letter(String from) {
        this.from = from;
    }

    public void writeLetter() {
        greeting();
        specialMessage();
        closing();
    }

    public void greeting() {
        System.out.println("Hello, friend!");
    }

    public void specialMessage() {
        System.out.println("Java is pretty cool, wouldn't you say?");
    }

    public void closing() {
        System.out.println("See you later!");
        System.out.println("From, " + from);
    }
```

A

B

C

Sample solution (back)

Letter.java

```java
public class Letter {
    // instance variables
    private String from;

    // constructor
    public Letter(String from) {
        this.from = from;
    }

    public void writeLetter(String toName) {
        greeting(toName);
        specialMessage();
        closing();
    }

    public void greeting(String name) {
        System.out.println("Hello, " + name + "!");
    }

    public void specialMessage() {
        System.out.println("Java is pretty cool, wouldn't you say?");
    }

    public void closing() {
        System.out.println("See you later!");
        System.out.println("From, " + from);
    }
}
```

passing the `toName` parameter through to the `greeting` method

Thoughts (back)

When implementing this with a parameter, it makes it possible to use **one single** `Letter` object (i.e. myLetter) to print multiple letters *to different people from the same person*:

```
// create Letter object
Letter myLetter = new Letter("Mr. Miller");

// write letters to various CSA students!
myLetter.writeLetter("Abdul");
System.out.println();

myLetter.writeLetter("Pritika");
System.out.println();

myLetter.writeLetter("Ethan");
System.out.println();

myLetter.writeLetter("Lucy");
```

If we had instead done it like this:

```
Letter myLetter = new Letter("Mr. Miller", "Abdul");
myLetter.writeLetter();
```

In order to print multiple letters to *different* people, we would need to:

A. Create a new `Letter` object for every person we want to write a letter to (this is less memory efficient since more objects created → more memory used):

```
Letter myLetter = new Letter("Mr. Miller", "Abdul");
myLetter.writeLetter();
Letter myLetter = new Letter("Mr. Miller", "Pritika");
myLetter.writeLetter();
etc.
```

OR

B. Add a setter method to the Letter class so that we could update the toName each time (more lines of code → more error prone!)

```
Letter myLetter = new Letter("Mr. Miller", "Abdul");
myLetter.writeLetter();
myLetter.setToName("Pritika");
myLetter.writeLetter();
etc.
```