Info: Wed December 8th 8:00-10:00PM

[redacted]

- ☐ [redacted]
    - ☐ [redacted]
    - ☐ [redacted]
- ☐ Make sure to check the times and locations of your final
- ☐ Study previous tests, quizzes, and homeworks

Variables

Question 1:

You can change the type of variable after it is declared:
Why or why not?

- ☐ True
- ☐ False

Question 2:

Variables labeled “final” can be changed:
Why or why not?

- ☐ True
- ☐ False

Question 3:

Give 2 examples of each of the following:

- strings: ___________ _____________
- booleans: ___________ _____________
- chars: ___________ ____________
- ints: ___________ ____________

Question 4:

Explaining the difference between a float and a double. Which is the default type?

Question 5:
Is this allowed? Why or why not?

```
double x = 10.0;
float y =  x;
System.out.println(y);
```

Expressions:

Question 6:
What type of incrementing is this? What will be printed?

```
int x = 10;
int y = x++;
System.out.println("x is: " + x);
System.out.println("y is: " + y);
```

Question 7:
What type of incrementing is this? What will be printed?

```
int x = 10;
int y = ++x;
System.out.println("x is: " + x);
System.out.println("y is: " + y);
```

Question 8:
Trace the following code. What will be printed?

```
int x = 20;
int y = 10;
while (x > y) {
   if (x % 2 == 0 || x % 5 == 0) {
       x -= 3;
       System.out.println("x is: " + x);
   } else if (y > 0) {
       y -= 2;
       System.out.println("y is: " + y);

   }

}
```

Question 9:
Write a program that takes in a string and capitalizes every even index character. The method should return the final String

Question 10:
Turn the following into a ternary expression:

```
public boolean ternaryEx(String x) {
   boolean isTrue;

   if (x.length() % 2 == 0) {
       isTrue = true;
   } else {
       isTrue = false;
   }
   return isTrue;
}
```

Question 11:

Override the toString method for the following class:

```
public class Parent {
   private int age;
   private String name;

//implement the override here


}
```

Question 12:
Based on problem 11, override the toString method for the following class:
Hint: is it possible to use the super keyword?

```
public class Child extends Parent {
   private String childName;

//implement the override here
}
```

Iterations and Math

Question 13:
Using printf formatting, print the following sentence:
my name is : maddy my age is 19 my GPA is   -4.000
A few notes:
The float should be printed with 3 decimal places
The minimum width of the float should be 10

Question 14:
What will the following code print out?

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };

intArray[3] = 9.1;
for(int x : intArray) {
   System.out.println(x);
}
```

Question 15:

Write two ways to iterate through an array and print each item:

Question 16:

What will the following code print?

```
float a = -14.9f;
float b = 199.3f;
System.out.printf("The absolute value " + "of %.3f is %.3f%n", a,
Math.abs(a));
System.out.printf("The ceiling of " + "%.2f is %.0f%n",b,
Math.ceil(b));
System.out.printf("The floor of " + "%.2f is %.0f%n",b,
Math.floor(b));
```

Question 17:

Write the code to generate an instance of the Random class and create variables int i, double d, and boolean b. Use the instance of Random to assign these variables random values:

Question 18:

Encapsulation does the following: (select all that are true)

- ☐ **Uses public visibility modifiers to ensure all classes can see data**
- ☐ **Uses public getters and setters**
- ☐ **Hides data from other classes'**
- ☐ **Ensures a class will have total control over what is stored in its fields**

OOP Principles

Question 19:
Use the OOP principle of polymorphism to answer the following questions:

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

A Deer IS-A Animal:

- ☐ True
- ☐ false

A Deer IS-A Vegetarian:

- ☐ True
- ☐ False

An Animal IS-A Deer:

- ☐ True
- ☐ False

A Deer IS-A Object:

- ☐ True
- ☐ False

Question 20:

Every primitive type has a corresponding wrapper class.

- ☐ True

- ☐ false

Question 21:

Java supports multiple inheritance:
Why or why not?

- ☐ **True**
- ☐ **False**

Question 22:

Write the default constructor and full constructor for the following class:
Note: make the default age to be 45 and the default name to be your own

```
public class Parent {
   private int age;
   private String name;

}
```

Question 23:
Write the default and full constructor for the class below:
Note: make the default child name your pet's name.
Indicate where the default call to super occurs.

```
public class Child extends Parent {
   private String childName;
}
```

Question 24:

What is wrong with the following code:

```
public class Parent {
//implementation
}

public final class Child extends Parent {
//implementation
}

public class Baby extends Child {
//implementation
}
```

Question 25:

List 3 differences between interfaces and abstract classes:

Question 26:

Create an interface with methods abstract methods display() and returnFinalNum() and static method returnOriginalValue()

Question 27:

Is this overriding, overloading, or neither

```
Public class Traveler {
      Public void explore(String place, String name) {
            //implementation
      }
}

Public class Hiker extends Traveler {
      Public void explore(String place, String name) {
            //implementation
      }
}
```

Question 28:

Parent is an abstract class.
True or false:
We can instantiate p. Why or why not? Could a subclass also be instantiated?

```
Parent p = new Parent();
```

Can we instantiate a child abstract class?

Question 29:

Trace the following code:

```
ArrayList<String> arr = new ArrayList<>();

arr.add("cs1301");
arr.add("cs1331");
arr.add(0,"cs1332");
arr.add(1,"cs2340");
arr.add(3, "cs1371");
System.out.println(arr);
```

Big-O

Question 30:

What is the time complexity of this:

```
for(Integer number : numbers) {
    if(number == comparisonNumber) {
      return true;
    }
  }
```

Question 31:

What is the time complexity of this method:

```
public static int returnLen(ArrayList<String> x) {
   return x.size();
}
```

Question 32:
What is the time complexity of this method?

```
private static void insertionSort(int[] elements) {
  for (int i = 1; i < elements.length; i++) {
    int elementToSort = elements[i];
    int j = i;
    while (j > 0 && elementToSort < elements[j - 1]) {
      elements[j] = elements[j - 1];
      j--;
    }
    elements[j] = elementToSort;
  }
}
```

JavaFX

Question 33:

Describe what the following code does:

```
TextField tf = new TextField();
button1.setOnAction(((ActionEvent event)-> {
   String name = tf.getText();
   System.out.println(name);
   tf.clear();
   nameLabel.setText(name);

           }));
```

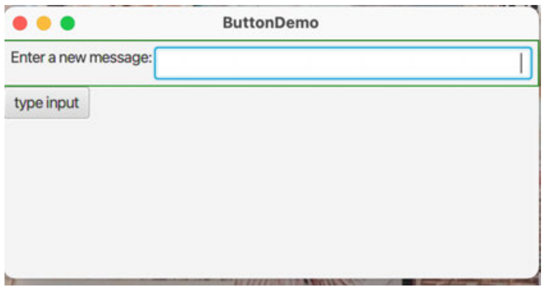
Question 34:
Which of the following is acceptable in event-driven programming:

- ☐
  ```
  button1.setOnAction(((e)-> {
          System.out.println(name);
          }));
  ```
- ☐
  ```
  button1.setOnAction(((Action Event)-> {
          System.out.println(name);
          }));
  ```
- ☐
  ```
  button1.setOnAction(e-> System.out.println("hi"));
  ```
- ☐
  ```
  button1.setOnAction(e-> {
     System.out.println("hi");
  });
  ```

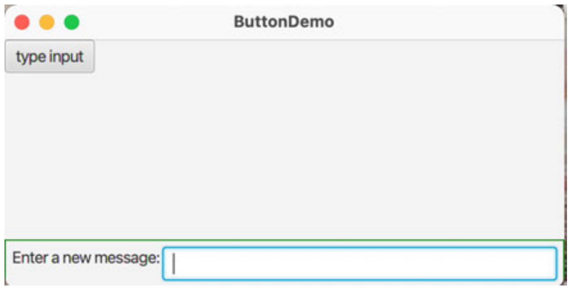
Question 35:
Trace the following code and decide what the JavaFX program will look like at completion: Assume the scene and stage have been properly set and no errors occur.

```
BorderPane pane = new BorderPane();
Text text = new Text("");
BorderPane paneForTextField = new BorderPane();
paneForTextField.setPadding(new Insets(5, 5, 5, 5));
paneForTextField.setStyle("-fx-border-color: green");
paneForTextField.setLeft(new Label("Enter a new message: "));

TextField tf = new TextField("Write name");
tf.setAlignment(Pos.TOP_LEFT);
paneForTextField.setCenter(tf);
pane.setTop(paneForTextField);
Label nameLabel= new Label("");

tf.setOnAction(e -> text.setText(tf.getText()));
Button button1 = new Button("type input");
button1.setOnAction(((ActionEvent event)-> {
   String name = tf.getText();
   System.out.println(name);
   tf.clear();
   nameLabel.setText(tf.getText());
           }));

button1.setOnAction(e-> {
   System.out.println("hi");
});

VBox middleBox = new VBox();

middleBox.getChildren().addAll(button1, nameLabel);
pane.setCenter(middleBox);
```

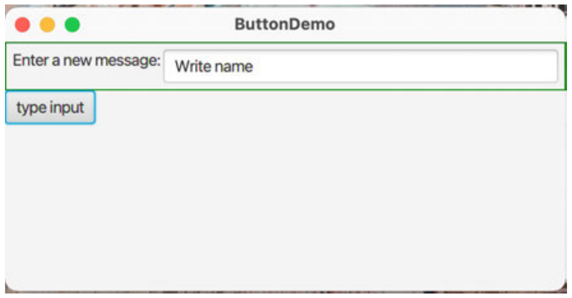
1. Option 1



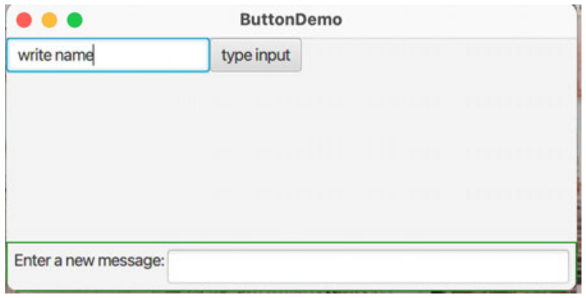
2. Option 2



3. Option 3



4. Option 4



Question 36:
Will this code run? If so, what will the ArrayList look like?

```
ArrayList<int> x = new ArrayList<>();
x.add(33);
x.add(2);
x.add(345);
x.add(45);
x.add(17);
x.remove(2);
System.out.println(x);
```

Question 37:
Will this code run? If so, diagram what the following linked list will look like:

```
LinkedList<Integer> x = new LinkedList<>();
x.add(10);
x.add(25);
x.add(-30);
x.addFirst(1);
x.addFirst(-2);
x.removeFirst();
System.out.println(x);
```

Question 38: Challenge:

Given an ArrayList of Integers, create a recursive method that iterates over each value and returns a total. If the Integer value is an even number, add 2x the value to the total. Otherwise, return the Integer's actual value:

Question 39:

Identify if the following snippets of code are autoboxing or autounboxing:

```
int num = 8;

Integer num2 = num;

-----------

Character no = new Character('u');

Character no_u = no;

-----------

Integer num = 10;

num.equals(12);

-----------

Integer num = 3;

num = num + 2;
```

Question 40:

T/F: If there is only one constructor for an object that takes in no parameters, then the object must have no instance values.

Question 41:

Check the variable names that would follow naming conventions, if it doesn't explain why:

```
__underTheFloorBoards

___IsLoveReal

___Am I Alive

__num_42

___12isTheBestNumber
```

Question 42:

What is the output of the following code?

```
int num = 4;

System.out.println(++num - num++);
```

Question 43:

Looking at the conversions below, check which ones can be done implicitly and explain why the conversion can or cannot be done implicitly:

```
___double -> int

___char -> boolean

_X_float -> double

___long -> byte

_X_int -> float
```

Question 44:

What is `val` equal to after the following line of code?

```
int val = (3 + 8 == 10) ? 1 : 0;
```

7. Convert the following `while` loop into a `for` loop:

```
int i = 0;

while (i < 15) {

     // loop body

     I++;

}
```

Question 45:
Looking at the following class hierarchy and code (you can assume that each class has an empty constructor):

```
                              Student
                             /       \
                            V         V
                      MEStudent   AEStudent

     Student s = new Student(); down

     Student ae = new AEStudent(); down

     MEStudent me = new MEStudent(); side
```

For each line of code below, state whether it compiles and/or executes:

```
     MEStudent s1 = (MEStudent) s;

     AEStudent ae1 = (AEStudent) ae;

     AEStudent ae2 = (AEStudent) me;

     Student s1 = (Student) me;
```

Question 46:

Using the same diagram above, these are the methods defined in each class (Note: remember that all classes extends the `Object` class):

```
Student: receiveGrades(), study()
```

```
MEStudent: buildBridge(), equals()
```

```
AEStudent: buildPlane(), study()
```

Imagine we define an object:

```
Student st = new AEStudent();
```

For each of the methods below, state whether the `st` object can execute it, if it can, state the specific implementation of which class its coming from:

```
buildPlane()
```

```
study()
```

```
equals()
```

Question 47:

Say we have 2 interfaces `Swimmable` and `Breathable`. Now we try to define another interface:

```
interface Huggable implements Swimmable, Breathable {

     \\Code

}
```

Is there anything wrong with this?

Question 48:

Create an `ArrayList` named `values` that holds `double` values.

Question 49:

T/F: Overriding is just another way of overloading.

Question 50:

If we had a `Button` named `henry` and we want it to do something when pressed, what are the 3 ways we can get it to do that?

Question 51:

What is the difference between `HBox` and `VBox`?

Question 52:

Coding: Create a class called `Tag`. `Tag` only has one private instance variable (`String tag`), one constructor, and one method. The constructor should take in only one parameter (`String tag`) and set it to the `tag` instance variable. The method called `modify` should return a `String`. `modify` will go through the first half of the characters in `tag` and increment the ASCII value of each by 1 and return this `String` (remember that `String` is immutable).