Info:

- ☐ [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

Java is strongly typed

Question 2:

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

- ☐ True
- ☐ False

They are constants and therefore cannot be changed

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?

Doubles are more accurate than floats.
(Double is the default type)
Doubles cannot be turned into floats

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

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

This is not allowed because it is a lossy conversion. Floats can be turned into doubles

---

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);
```

x is: 11
y is: 10

Postincrement

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);
```

x is: 11
y is: 11
Preincrement

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);

   }

}
```

x is: 17
y is: 8
y is: 6
y is: 4
y is: 2
y is: 0
Continues to run

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

```
public String toUpperMethod(String x) {
   String x = "georgia tech";
   String y = "";
   for (int i = 0; i < x.length(); i++) {
       if (i % 2 == 0) {
           y += Character.toUpperCase(x.charAt(i));
       } else {
           y += x.charAt(i);
       }

   }
   System.out.println(y);
   return y;
}
```

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;
}
```

```
boolean isTrue = x.length() % 2 == 0 ? true : false;
```

Question 11:

Override the toString method for the following class:

```
public class Parent {
   private int age;
   private String name;
//implement the override here

public String toString() {
   return "name : " + name + " age: " + age;
}


}
```

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

public String toString() {
   return super.toString() + " child's name:  " + childName;
}

}
```

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

```
System.out.printf("my name is : %s my age is %d my GPA is %10.3f", "maddy",
19, -4.0);
```

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);
}
```

Error bc 9.1 is not int

Question 15:

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

```
for (int y: x) {
   System.out.println(y);
}

OR

for (int i = 0; i < x.length; i++) {
   System.out.println(x[i]);
}
```

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));
```

```
The absolute value of -14.900 is 14.900
The ceiling of 199.30 is 200
The floor of 199.30 is 199
```

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:

```
Random random = new Random();

int i = random.nextInt();
double d = random.nextDouble();
boolean b = random.nextBoolean();

System.out.println(i);
System.out.println(d);
System.out.println(b);
```

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**]

<u>OOP Principles</u>

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

Every primitive data type has its corresponding object wrapper class: Character , Boolean and the subclasses of the abstract class Number for numeric values: Byte , Short , Integer , Long , Float and Double .

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;
public Parent(int age, String name) {
   this.age = age;
   this.name = name;

}


//default age = 45
public Parent() {
   this(45, "maddy");
}



}
```

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;

public Child(int age, String name, String childName) {
    super(age, name);
    this.childName = childName;

}

public Child(String childName) {
      super();
    this.childName = childName;
}


}
```

Question 24:

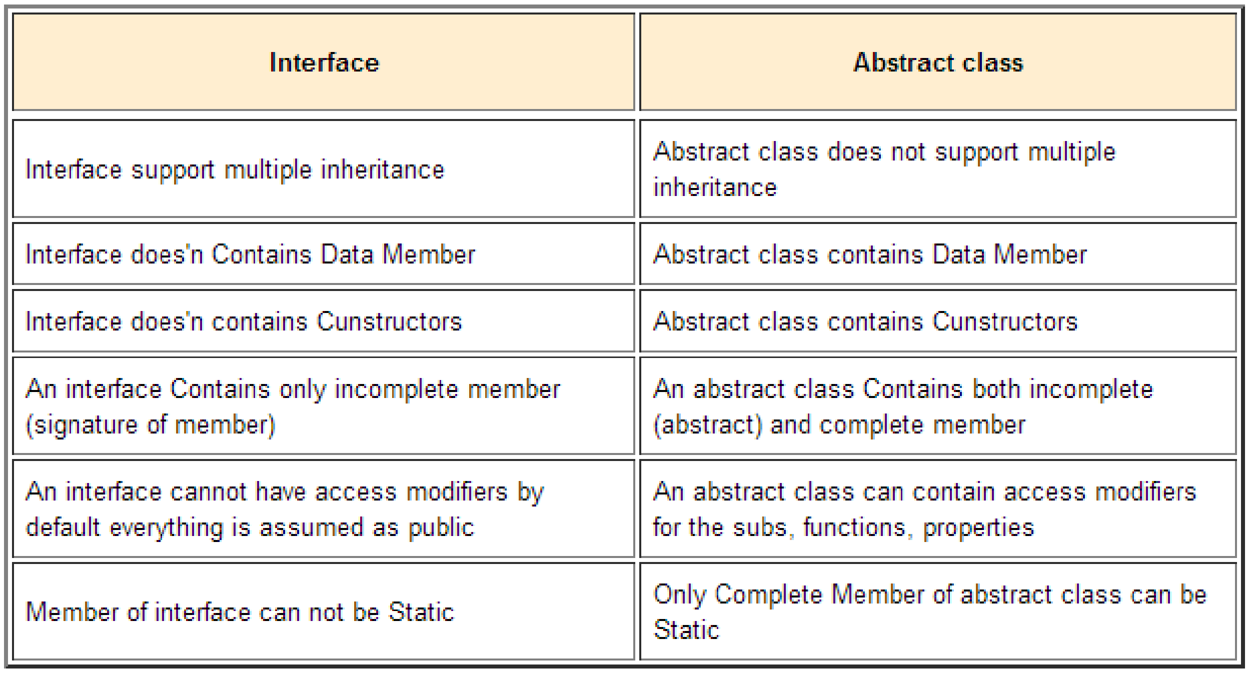
What is wrong with the following code:

```
public class Parent {
//implementation
}

public final class Child extends Parent {
//implementation
}

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

Child class cannot be extended bc it has been label as final

Question 25:

List 3 differences between interfaces and abstract classes:



Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

---

Question 26:

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

```
public interface InterfacePractice {

    void display();

    int returnFinalNum();

    static int returnFinalNum() {
        return FINAL_NUM;
    }

}
```

```java
public void display() {
    System.out.println("this is display()\n name: " + name + " age: " + age);
}
```

Question 27:

Is this overriding, overloading, or neither
Answer: overriding

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

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

Same name but different parameters, overloading

Method signature (name and parameters) are same, it's overriding

---

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();
```

```
Answer: false - only concrete subclasses can be instantiated
Can we instantiate a child abstract class?
Answer: no
```

```
Abstract classes cannot be instantiated
Subclasses usually provide implementation for methods
Can be subclasses
Can have static fields and static methods
```

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);
```

[cs1332, cs2340, cs1301, cs1371, cs1331]

Big-O

Question 30:

What is the time complexity of this:

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

O(n)

Question 31:

What is the time complexity of this method:

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

O(1)

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;
  }
}
```

O(n^2)

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);

           }));
```

- Gets text from TextField
- Prints out text in terminal

- Clears the TextField
- Sets label with name

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

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

These are all correct because they are all different ways to write lambda expressions

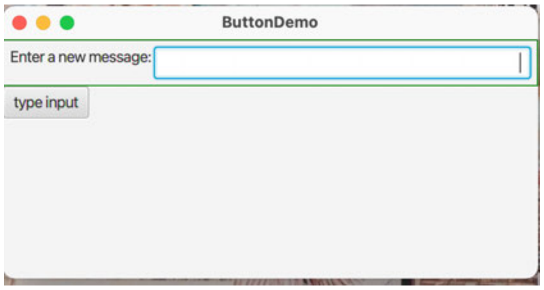
---

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.

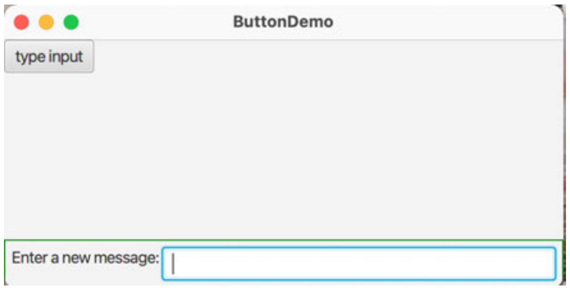
```
BorderPane pane = new BorderPane();
Text text = new Text("");
BorderPane paneForTextField = new BorderPane();
paneForTextField.setPadding(new Insets(5, 5, 5, 5));
paneForTextField.setStyle("-fx-border-color: green");
paneForTextField.setLeft(new Label("Enter a new message: "));

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);
```

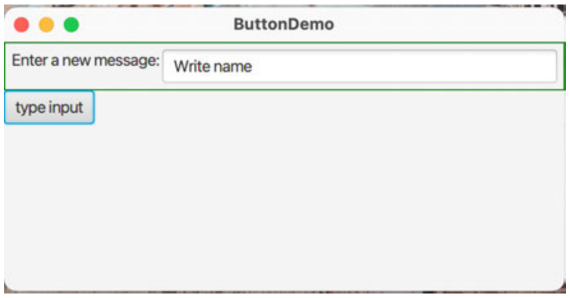
1. Option 1



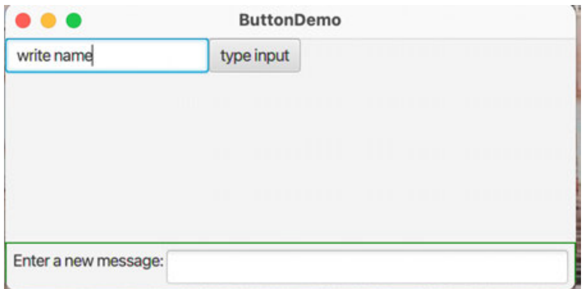
2. Option 2



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);
```

This is not allowed because ArrayLists need objects not primitives. (integer not ints)

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);
```

[1, 10, 25, -30]
^This should be written as nodes

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:

```
public static int returnLen(ArrayList<Integer> x) {
   if (x.size() == 0) {
       return 0;
   } else {
       Integer y = x.remove(0);
       if (y.intValue() % 2 == 0) {
           return 2 * y.intValue() + returnLen(x);
       } else {
           return y.intValue() + returnLen(x);
       }
   }
}
```

Question 39:

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

```
int num = 8;

Integer num2 = num; //autoboxing
```

```
-----------

Character no = new Character('u');

Character no_u = no; //autounboxing

-----------

Integer num = 10; //autoboxing

num.equals(12); //autoboxing

-----------

Integer num = 3; //autoboxing

num = num + 2; //autounboxing
```

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. **False, the object will still have the instance variables, they just won't be assigned a value when the object is created, unless it already had a predetermined value.**

Question 41:

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

`_X_underTheFloorBoards`

`___IsLoveReal` **`//Capitalized`**

`___Am I Alive` **`//Capitalized and spaces`**

`_X_num_42`

`___12isTheBestNumber` **`//Starts with a number`**

Question 42:

What is the output of the following code?

```
int num = 4;

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

0

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` **`//64 bits -> 32 bits, lossy conversions`**

`___char -> boolean` **`//16 bits -> 1 bit, lossy conversion`**

`_X_float -> double` **`//32 bits -> 64 bits, no info lost`**

`___long -> byte` **`//64 bits -> 8 bits, lossy conversion`**

`_X_int -> float` **`//32 bits -> 32 bits, no info lost`**

Question 44:

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

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

0

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

```
int i = 0;

while (i < 15) {

     // loop body

     i++;
}

    for (int i = 0; i < 15; i++) {
        // loop body
    }
```

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; //compiles, doesn't execute

    AEStudent ae1 = (AEStudent) ae; //compiles and executes

    AEStudent ae2 = (AEStudent) me; //doesn't compile or
    execute

    Student s1 = (Student) me; //compiles and executes
```

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() //No
```

```
study() //Yes, implementation from AEStudent
```

```
equals() //Yes, implementation from Object
```

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?

Yes, interfaces can implement more than one interface, but when an interface is trying to use another interface we use `extends` rather than `implements`.

Question 48:

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

```
ArrayList<Double> values = new ArrayList<Double>;

Or

ArrayList<Double> values = new ArrayList<>;
```

Question 49:

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

False, overriding can only happen through inheritance, while overloading can happen with or without inheritance.

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?

Inner class, anonymous inner class, or lambda expression

Question 51:

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

`HBox` holds nodes in a pane horizontally while `VBox` holds nodes in a pane vertically.

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).

```
public class Tag {
     private String tag;
```

```
    public Tag(String tag) {
        this.tag = tag;
    }

    public String modify() {
        String newTag = "";
        for (int i = 0; i < tag.length() / 2; i++) {
            newTag += (char) (tag.charAt(i) + 1);
        }
        tag = newTag;
        return tag;
    }
}
```