1. The **`String s`** contains a ***valid*** (x,y) coordinate in the format **`'(22,154)'`**. There are no blanks in this String. Complete the single statement below that will find the x coordinate, as a **`String`** value (in this example, it should set **`x`** to **`"22"`**).

```
String x = _____________________________________________________;
```

2. What is printed to the console by the following code? (Your answer does not have to exactly match the format of the result from **`ArrayList.toString`**.)

```
ArrayList<String> myList = new ArrayList<String>();
myList.add("banana");
myList.add("kiwi");
myList.add(1, "orange");
myList.set(2, "lollipop");
myList.remove(1);
System.out.println(myList);
```

Console:

3. What output would be written to the console by the method below if called with **`m(1)`**?

```
public static void m(int n){
  if(n>10)
    System.out.println("large num");
  else {
    System.out.println(n);
    m(2*n+1);
  }
}
```

Console:

4. Draw all of the memory allocated by the following statements.

```
int[] a = new int[4];
String x = new String[3];
x[2] = "Hello";
```

`a` ☐

`x` ☐

5. How many array objects are created by **`new int[4][17]`**, and what types of data do they hold?

It creates ____ array(s) each containing ____ value(s) of type _____

It creates ____ array(s) each containing ____ value(s) of type _____

.

6. Complete the code below, which will set **`second`** to a reference to the 2nd node in a linked list, if there is one, and to **`null`** if there isn't. This code is *inside* the **`IntLinkedList`** class.

```
if(_____________________________)
   second = ____________________________________;
else
   second = null;
```

7. Which of the following algorithms is the most efficient in the worst case?

   a) Insertion sort
   b) Selection sort
   c) Merge sort
   d) Quicksort
   e) They are all equally efficient in the worst case

8. The **`doSomeIO`** method in **`methodX`**, below, might throw an **`IOException`**. You're getting the error "Error: unreported exception java.io.IOException". Show ***two*** ways to remove this error. (You don't have to do anything if the **`IOException`** occurs – just ignore it. But remove the error.)

```
public void methodX()
{

   doSomeIO();




}
```

```
public void methodX()
{

   doSomeIO();




}
```

9. A friend of yours likes robots, and built a `Robot` class as part of a project. Unfortunately, it is incomplete. Help your friend out by completing the `clone()` method.

```
class Robot {
    private int serialNum;
    private int servoCount;

    public Robot(int serialNum, int servoCount) {
        this.serialNum = serialNum;
        this.servoCount = servoCount;
    }

    public Robot clone() {


      }
```

10. Implement a superclass **`Art`**, and two subclasses representing different types of art: **`Sculpture`** and **`Painting`**.

    All types of art have an artist (**`String`**), and a value (**`double`**). Paintings also have a width and height (**`int`**).

    Write appropriate constructors for each class to initialize all instance variables.

    All types of art should also have a method **`void increaseValue()`** that will update the value of the art (for example, each time the art is sold). The value of sculptures increases by 5%, while the value of paintings and other types of art increases by 10%.

    You do not need to write **`toString()`** methods, or a main program or any other classes/methods not specified above. Use proper object-oriented design techniques, and avoid repeated or unnecessary code.

11. Define an ***abstract*** class **`Animal`** with one ***private*** instance variable (a **`String`**) giving the name of the type of animal (such as "Cat", "Moose", or "Butterfly"). Provide a constructor that accepts the name as a parameter. Provide a standard **`toString()`** method that returns a **`String`** of the form **`"Animal:Cat"`**, or **`"Animal:Butterfly"`**. Do not write any other methods in this class.

    Define a class **`Dog`** which is a subclass of the **`Animal`** class. There should be one private instance variable (a **`String`**) which specifies the breed of the dog (such as "Poodle" or "Beagle"). Provide a constructor that accepts the breed as its only parameter. Provide a standard **`tostring()`** method that returns a **`String`** of the form **`"Animal:Dog(Poodle)"`** or **`"Animal:Dog(Beagle)"`**. This **`tostring`** method ***must*** use the superclass **`tostring`** method to generate the first part of the result (the **`"Animal(Dog)"`** part).

12. The class **`NestedAL`** is a subclass of **`ArrayList`**. Every element stored in a **`NestedAL`** will be either 1) a **`String`** or 2) another **`NestedAL`**. For example, a **`NestedAL`** might contain:

    `["Test", ["one", "two", ["deep", "level"]], "Last"]`

    Complete the ***recursive*** method **`deepCopy()`** which will make a full deep copy of any **`NestedAL`**. It should return a copy of the **`NestedAL`**, which contains copies of any elements that are also **`NestedAL`** objects, to any number of levels. (You never need to make deep copies of **`String`** objects.)

```
public class NestedAL extends ArrayList {
public NestedAL deepCopy(){
```