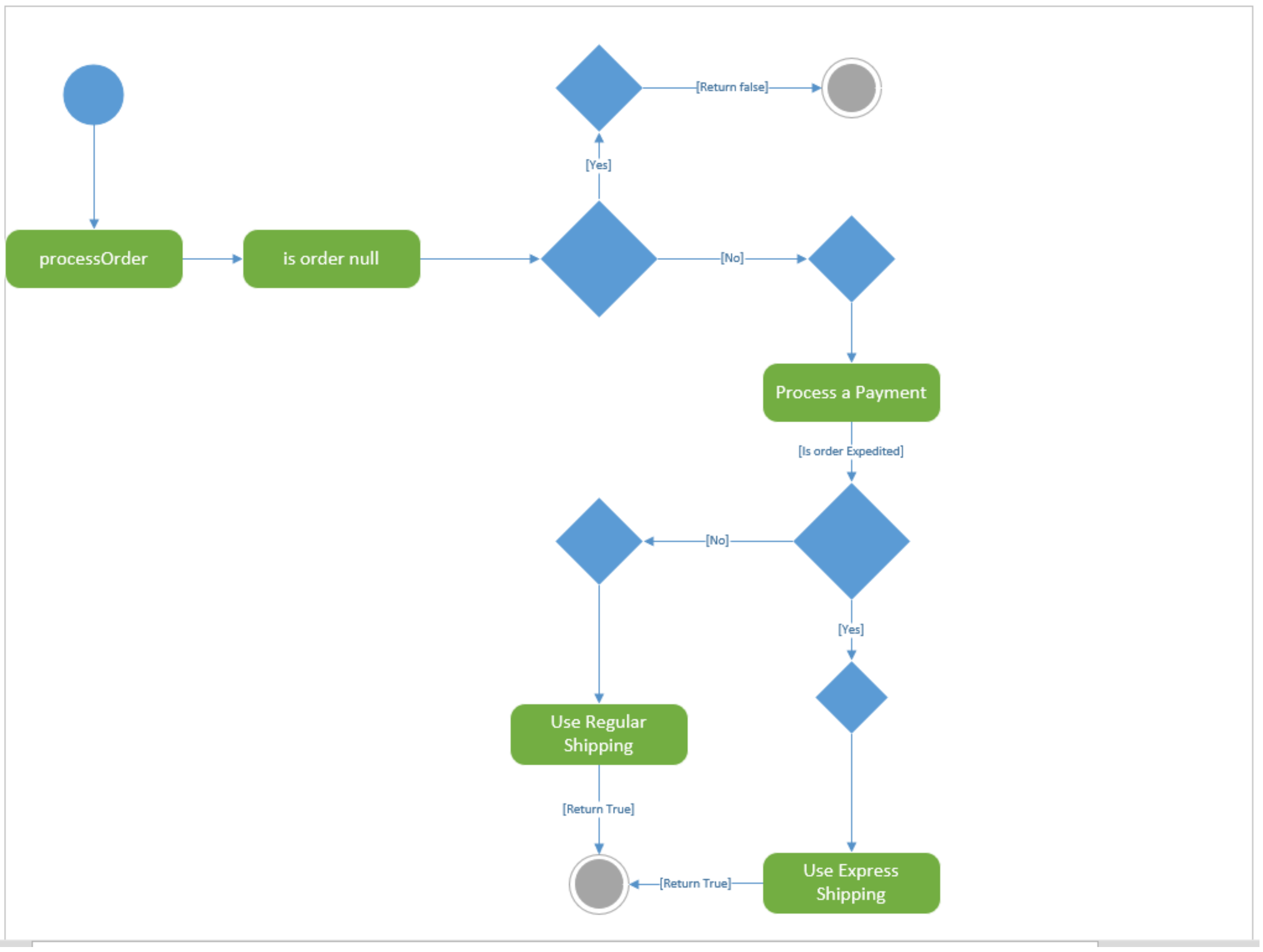
4)

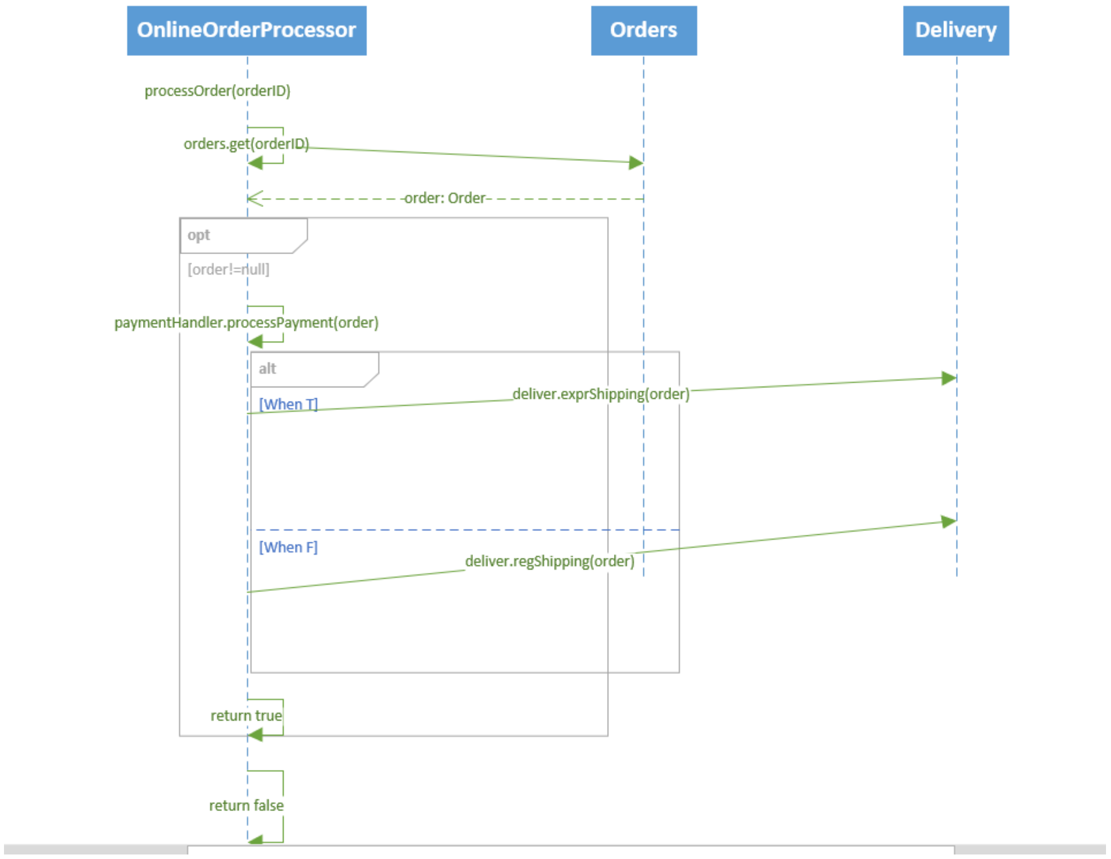
```
	@SuppressWarnings("unchecked")
	public static ArrayList Comparision(ArrayList oldList, Comparator c){

		Collections.sort(oldList, c);
		ArrayList newList = new ArrayList();
		newList = (ArrayList) oldList.get(0);//Gets max or min
		newList = (ArrayList) oldList.get(oldList.size() - 1);//Gets other max
or min

		return newList;
	}
```

8b)





9)

I implemented the classes as I saw fit and as I thought would be the most efficient for the program.

```
import java.util.*;

public class Lamp{
	protected boolean status;

	Lamp(){
		status = false;
	}

	public boolean getStatus() {
		return status;
	}

	public void turnOn(){
		status = true;
	}

	public void turnOff() {
		status = false;
	}
```

```
class Switch{
      protected Lamp lamp;
      Switch(){
            lamp = new Lamp();
      }

      public void changeStatus() {
            if(lamp.getStatus()) {
                  lamp.turnOff();
            }
            else {
                  lamp.turnOn();
            }
      }

      public String toString() {
            String str = "The status of the light bulb is ";

            if(lamp.getStatus()) {
                  str += "on";
            }
            else {
                  str += "off";
            }

            return str;

      }
}
```

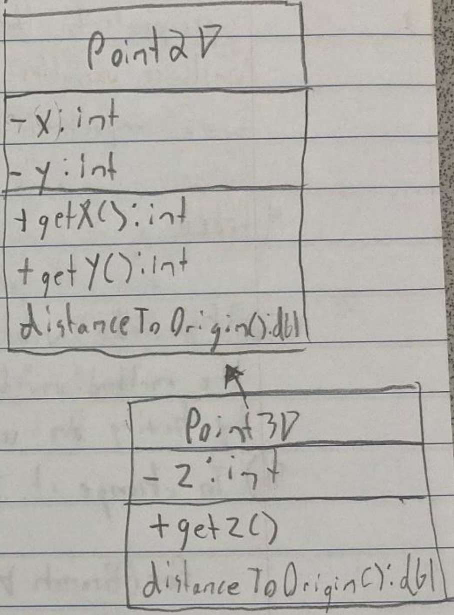
1-3,5-8a)

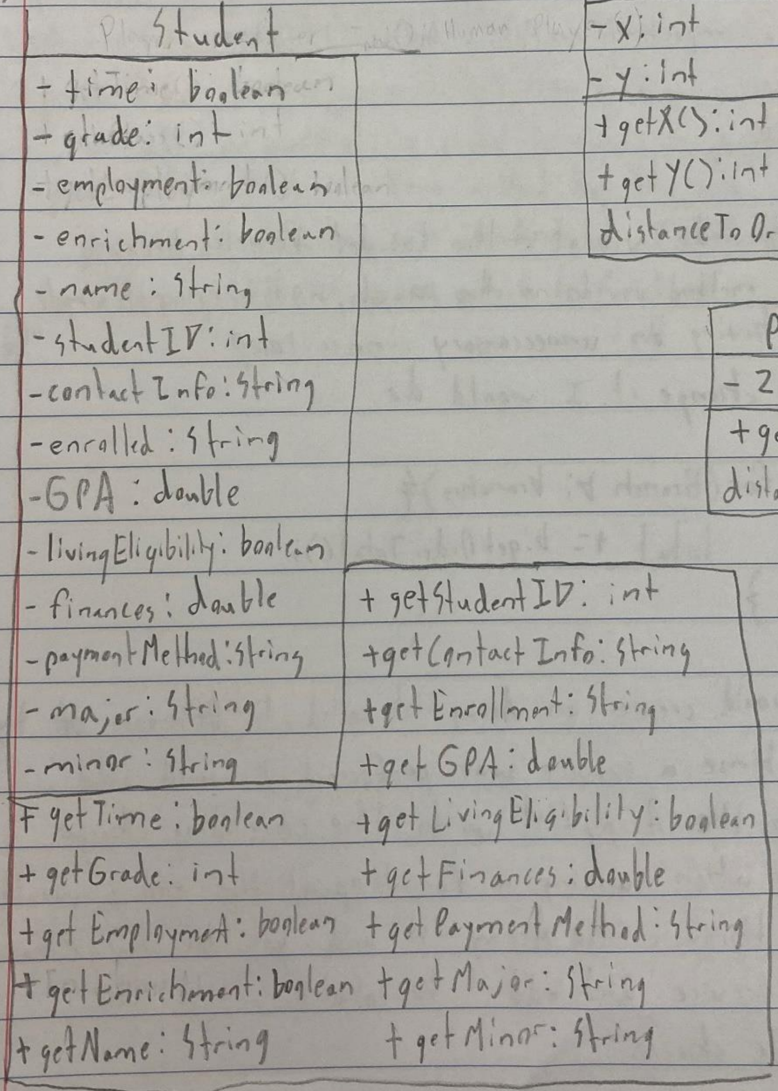
Noah Streveler Midterm

1) D obj = new C(); C obj = new C();
D obj = new D(); this.super();

2) The only issue that I see is there should be a method overriding distanceToOrigin()

3





3) I used this design because I wasn't sure if we were allowed to use subclasses, as the instructions said just to use instance variables and methods. For these methods though, I used boolean when I felt it was one or the other, and used other primitives when appropriate to the variable I was getting. I also added instance variables for major and minor as I felt they were important information.

4 Coded

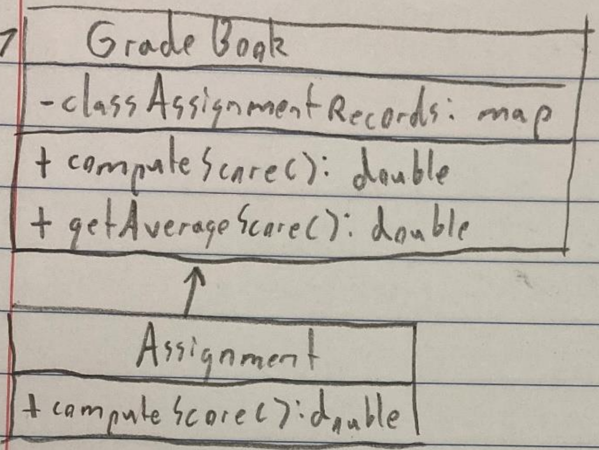
5a) This code violates the Law of Demeter because the method is doing too much, not keeping it simple by having an unneccassary inner loop

5b) To change it I would do

```
for (Branch b: branches){
    total += b.getOrderTotal();
}
```

6 I would create an ArrayList called listOfServices of type service and every time a service was performed I would add it to the ArrayList with the correlating charge. Then when running a for loop at the end I would run through the ArrayList and get the charge for each service and add it to totalBill, like the example shows.

7



8 a) - OnlineOrderProcessor(), Temporal - It initializes variables

- processOrder, Communicational - Each line calls on another
  variable or method in order to function properly
- processReturn, Functional - This method has the sole
  purpose of figuring out which kind of refund the
  customer will be given, which is functional

b)