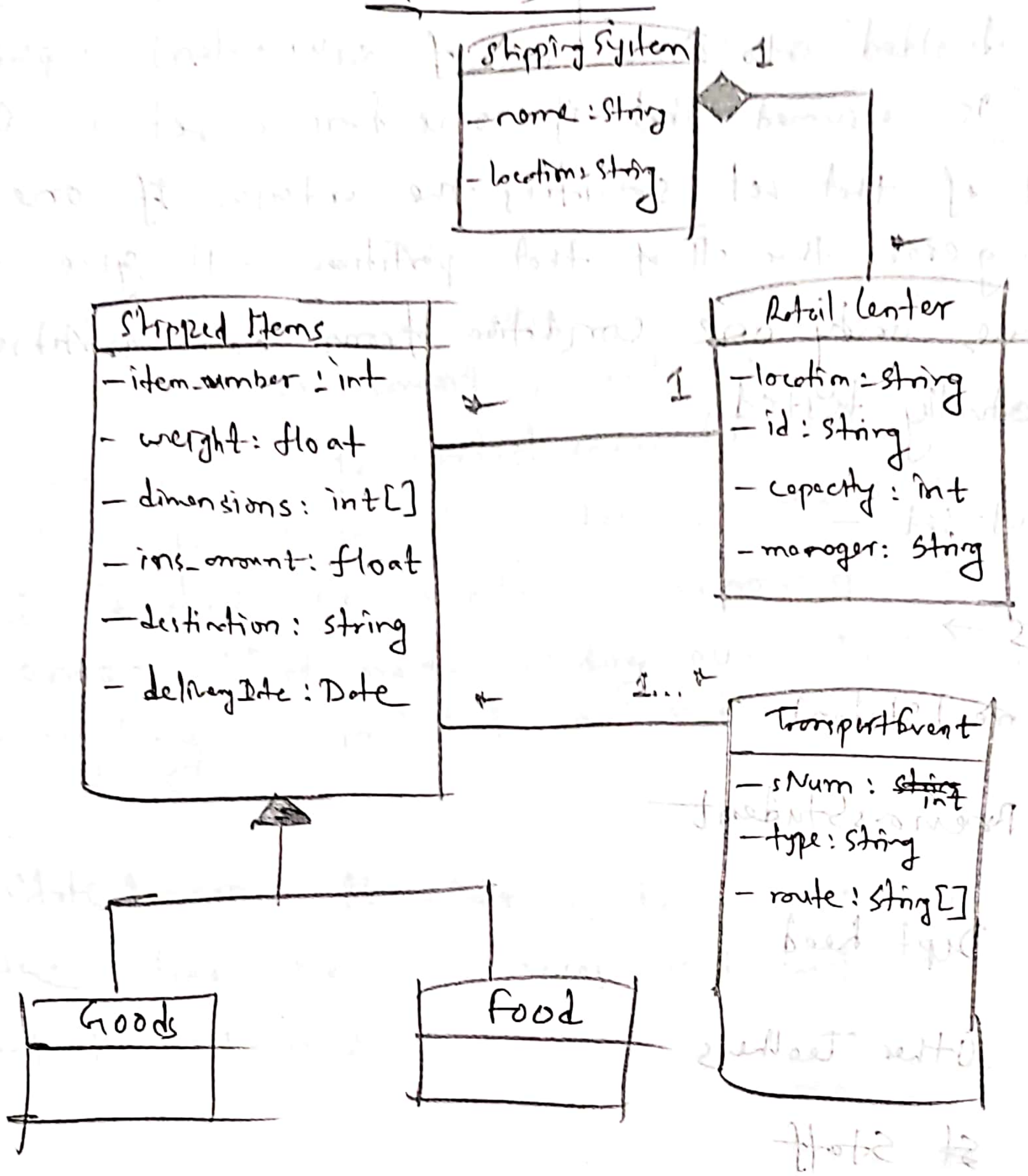
18 – 19

1. (a)



Shipping System
– name : String
– location : String
1
1
*
Shipped Items
– item_number : int
– weight : float
– dimensions : int[]
– ins_amount : float
– destination : string
– deliveryDate : Date
*
1
Retail Center
– location : String
– id : String
– capacity : int
– manager : String
*
1...*
TransportEvent
– sNum : ~~string~~ int
– type : String
– route : String[]
Goods
Food

b) Equivalence Partitioning → ~~Test~~ Test input data is divided into some set of equivalent input data. It is assumed that if one from a set is OK, then all of that set satisfies the criteria. If one is giving error, then all of that partition will give error. Thus, only one condition from each partition is actually tested.

· Input set –

Sets →

i) CurrentStudent

ii) PreviousStudent

iii) Dept head

iv) Other Teachers

v) ~~St~~ Staff

vi) BITS Admin member

Input is for all, the login credentials etc and check whether they can login and perform their tasks.

c) Defect rate → number of defects in a project.
Defect density → the average number of defects in a unit of code length or size, say KLOC

Checklist → Checklists are a way to maintain that all essential components are functional and every requirement is satisfied.
Omissions are the hardest errors to find as ~~it~~ if there is not something, how can we find?
Checklists are helpful in removing omission errors. That means finding out whether everything is included or not.

2.(a) State Pattern, IBehaviour is the interface and the others are the classes that will implement this interface.

```
public interface IBehaviour{
    ~~public~~ void action(Robot r, Position p, Boolean obstacle);
}

public class Aggretive implements IBehaviour{
    public void action(Robot r, Position p, Boolean obstacle){
        r.setState(this);
        if (obstacle == true){
            System.out.println(r.getName() + " is attacking ");
        }
    }
}

public class Defensive implements IBehaviour{
    public void action(Robot r, Position p, Boolen obs){
        r.setState(this);
        if (obs == true){
            System.out.println( r.getName()+ "running away");
        }
    }
}
```

```
public class Normal implements IBehaviour {
    public void action (Robot r, Position p, Boolean isOn) {
        r.setMode(Collect);
        if (obs == true) {
            // do nothing
            System.out.println(r.getName() + " stays calm");
        }
    }
}

public class Robot {
    private String name;
    private IBehaviour b; ~~initialb~~;
    Robot (String n, ~~IBehaviour b~~) {
        this.name = n;
        ~~this.initialb = b~~;
    }
    ~~private~~ public void setState( IBehaviour b) {
        this.b = b;
    }
}
```

```
public class Demo {
    public static void main (String [] args) {

        Robot r1 = new Robot ("Big Robot", ~~new Aggressive()~~);
        Robot r2 = new Robot ("George v.2.1");
        Robot r3 = new Robot ("R2");

        ~~r1.~~ ~~A~~
        IBehaviour a = new Aggressive();
        a.doAction(r1, (2,3), True);
        :
        :
    }
}
```

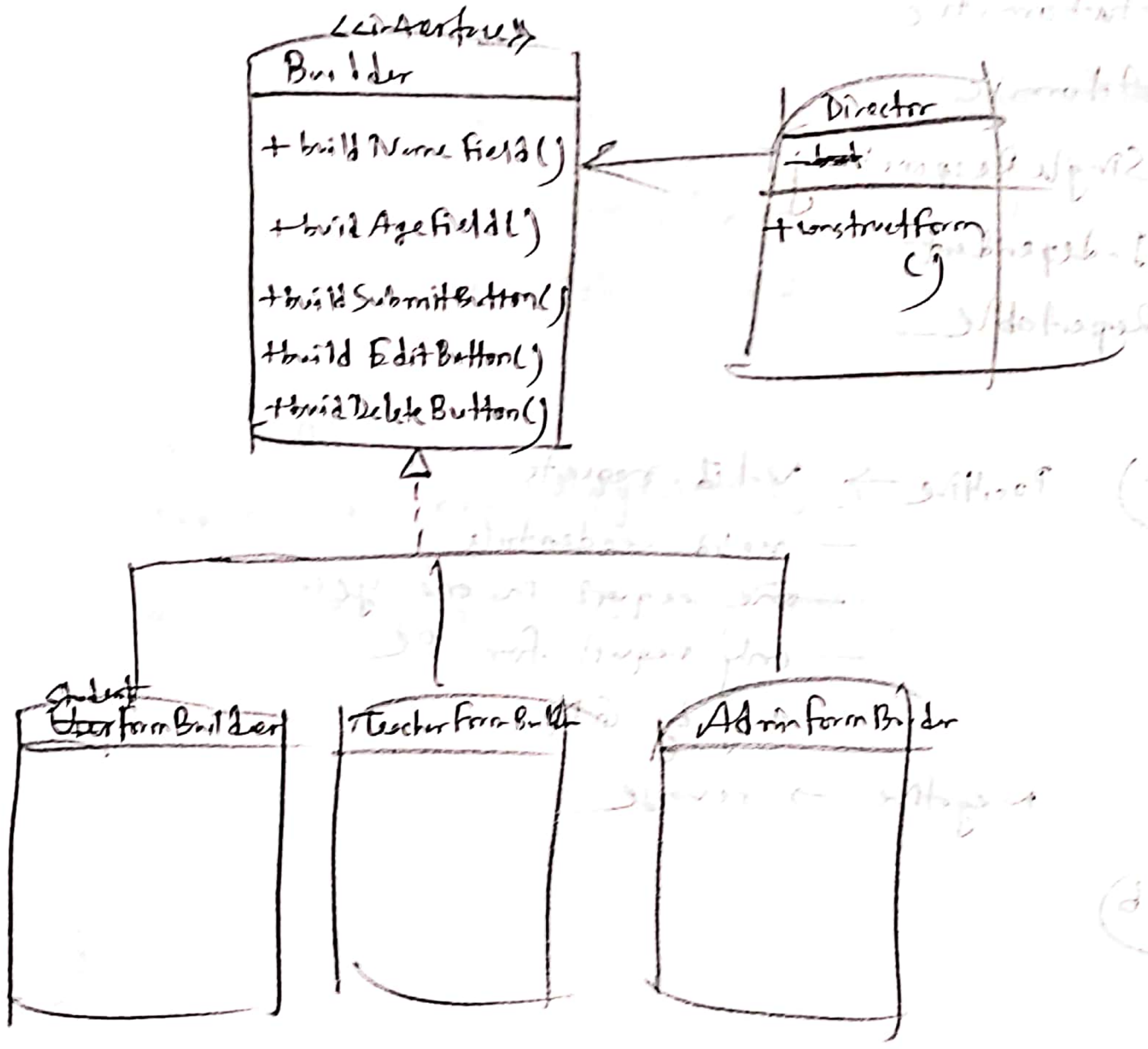
2. b) SLL - Triangles . . .

Automatic
Atomic
Single Responsibility
I-dependent
Repeatable

3 - a) Positive → valid requests
- valid credentials
- one request in one year
- only request for PC
- only by GM

negative → reverse

4 b)



Code → Main idea → in StudentFormBuilder the function for build ~~Sub~~ Edit and buildDeleteButton will be blank.

In TeacherForm, ~~Del~~ deletButton step will be blank.

In ~~other~~ Adminform, everything okay.

c) Cohesion → Connection among internal components
Coupling → Connection among different classes

We want high cohesion, low coupling

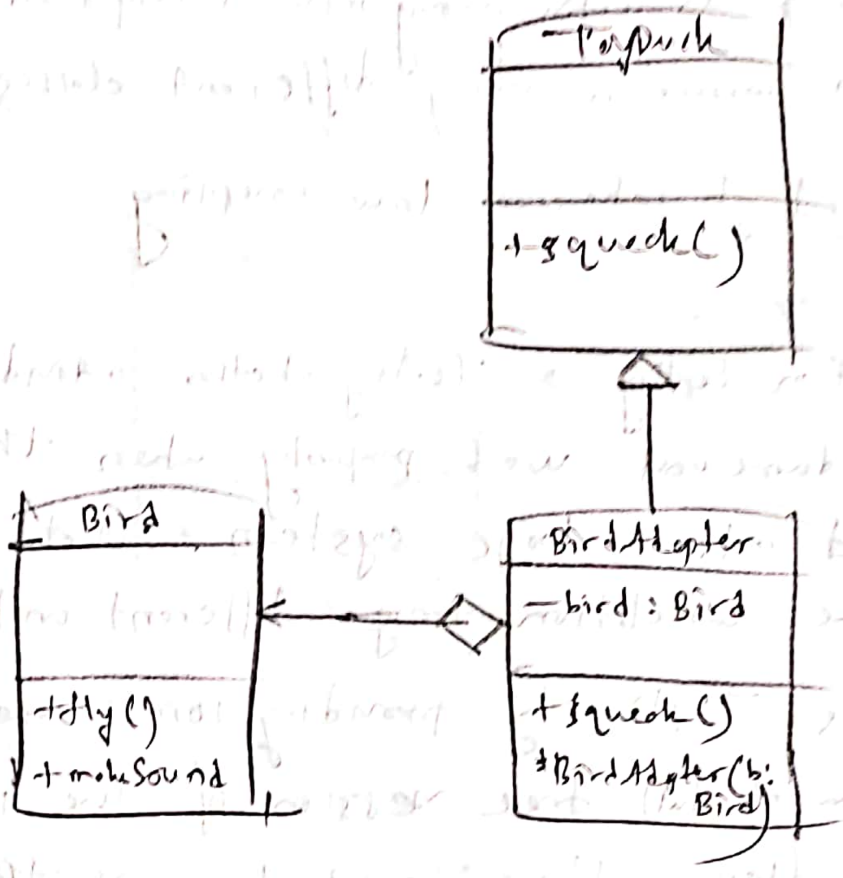
4. a) Integration Testing → Testing whether individual software / hardware unit work properly when they are integrated into a large system. That means it we tests the correlation among different units

Beta Testing → Testing by providing some users a advance portion of full free version of the product to find out flaws. Users report bugs as they wish without a specific format. It is low cost.

Usability Testing → Testing how well a user can learn to use the interface.

b)

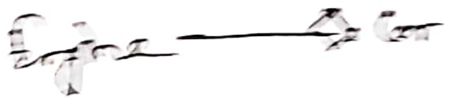
b)



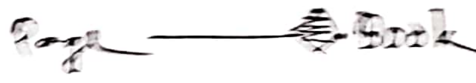
```
public class ToyDuck{
    public void squeak() {
        //
    }
}
public class BirdAdapter ~~implemen~~ extends ToyDuck{
    private Bird bird;
    public BirdAdapter (Bird b) {
      this. bird = b;
    }
    public void squeak() {
        bird. makeSound ();
    }
}
```

```
public class Bird {
    public void fly() {
        //
    }
    public void makeSound() {
        //
    }
}
```

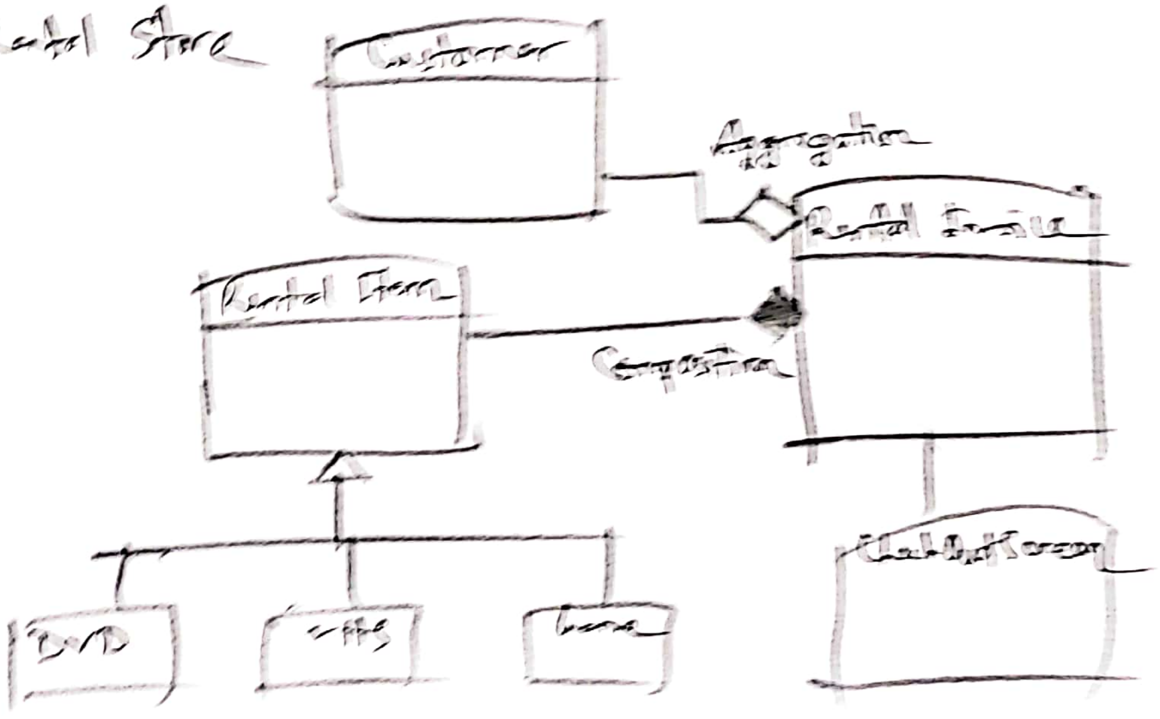
i) Aggregation → is part of
— white diamond



Composition → is entirely made of
— black diamonds
— stronger



Video Rental Store


Customer
Aggregation
Rental Invoice
Rental Item
Composition
DVD
VHS
Game
CheckOutScreen