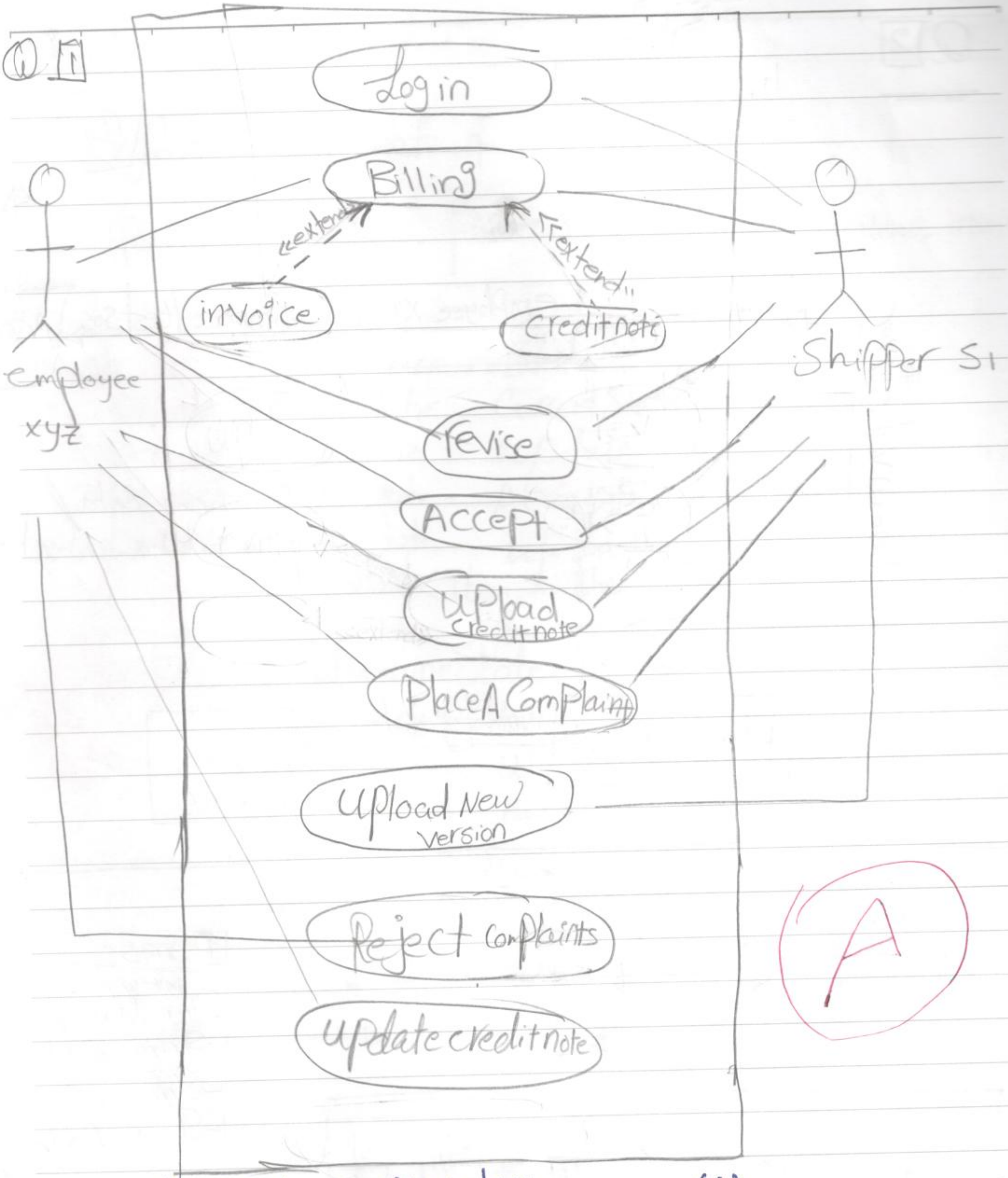
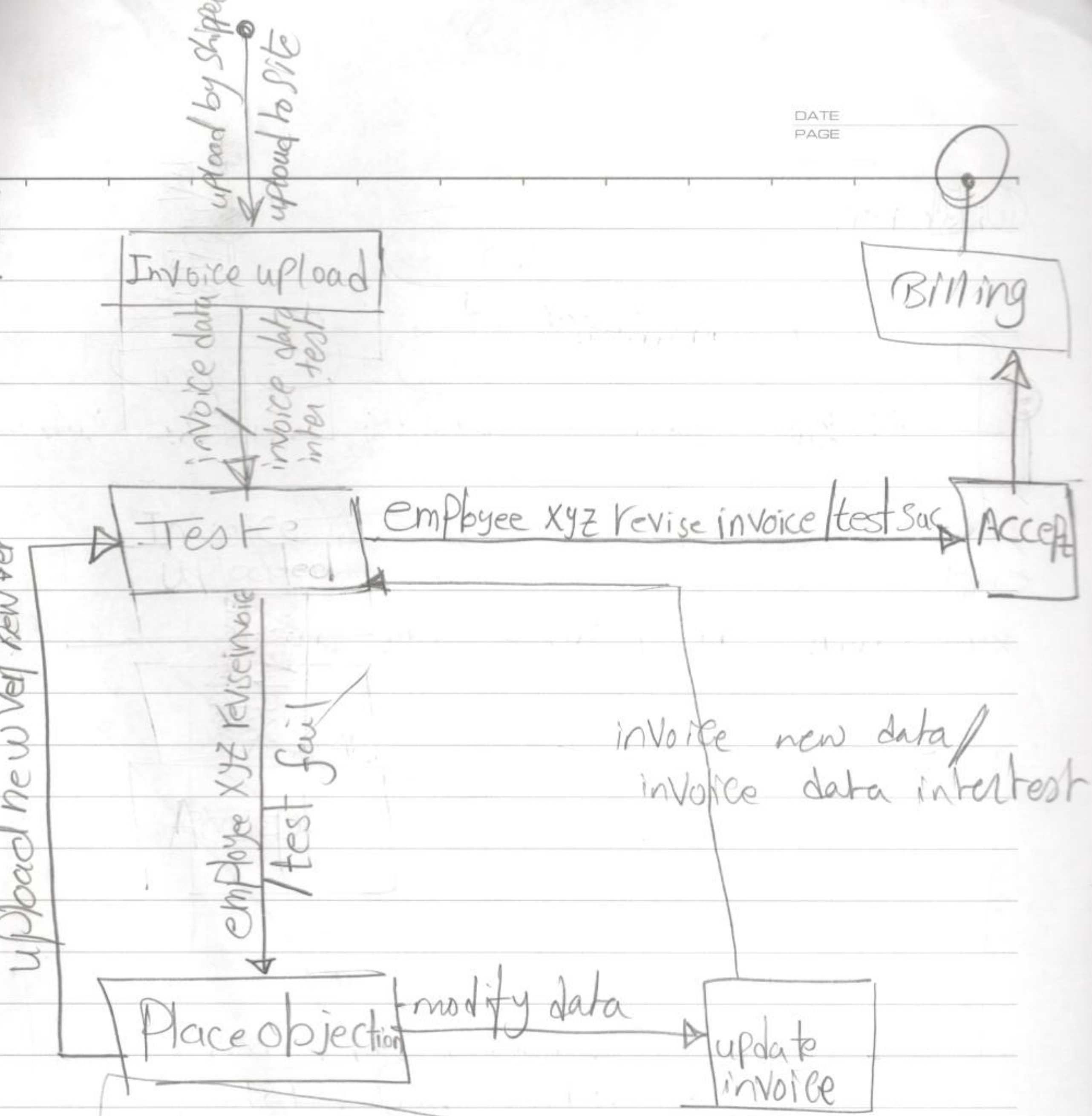
2009

1 [1]

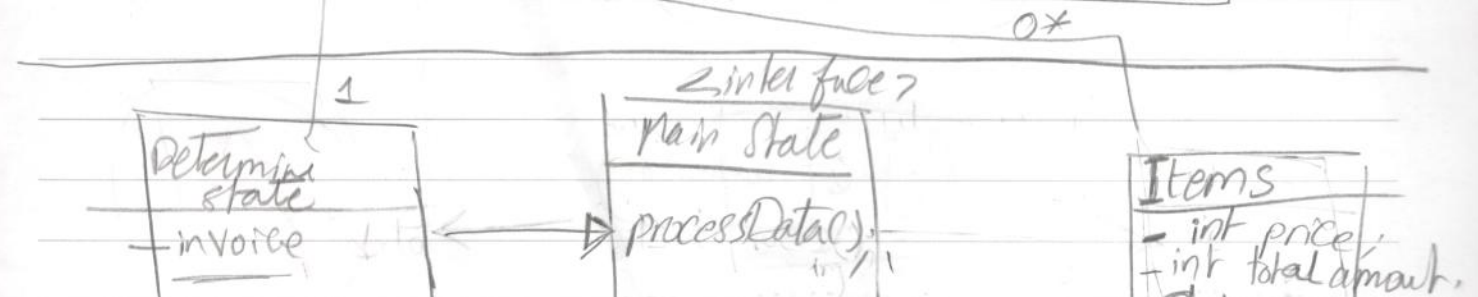


Ahmed Helmi AlWakil (6)
Basma Mahmoud Ali Taha (16)
Sherouk Hossam ElDien Khamis (29)
Abd ElRahman Hany (36)

Q 2



upload by shipper / upload to site
Invoice upload
invoice data / invoice data inter test
Test
employee xyz revise invoice / test suc
Accept
Billing
employee xyz revise invoice / test fail
upload new ver / new ver
invoice new data / invoice data intertest
Place objection
modify data
update invoice



1
0*
Determine state
- invoice
<interface>
main State
processData();
Items
- int price;
- int total amount;
- String name;
- String date.
- String address;
upload.
update
Accept

DATE
PAGE

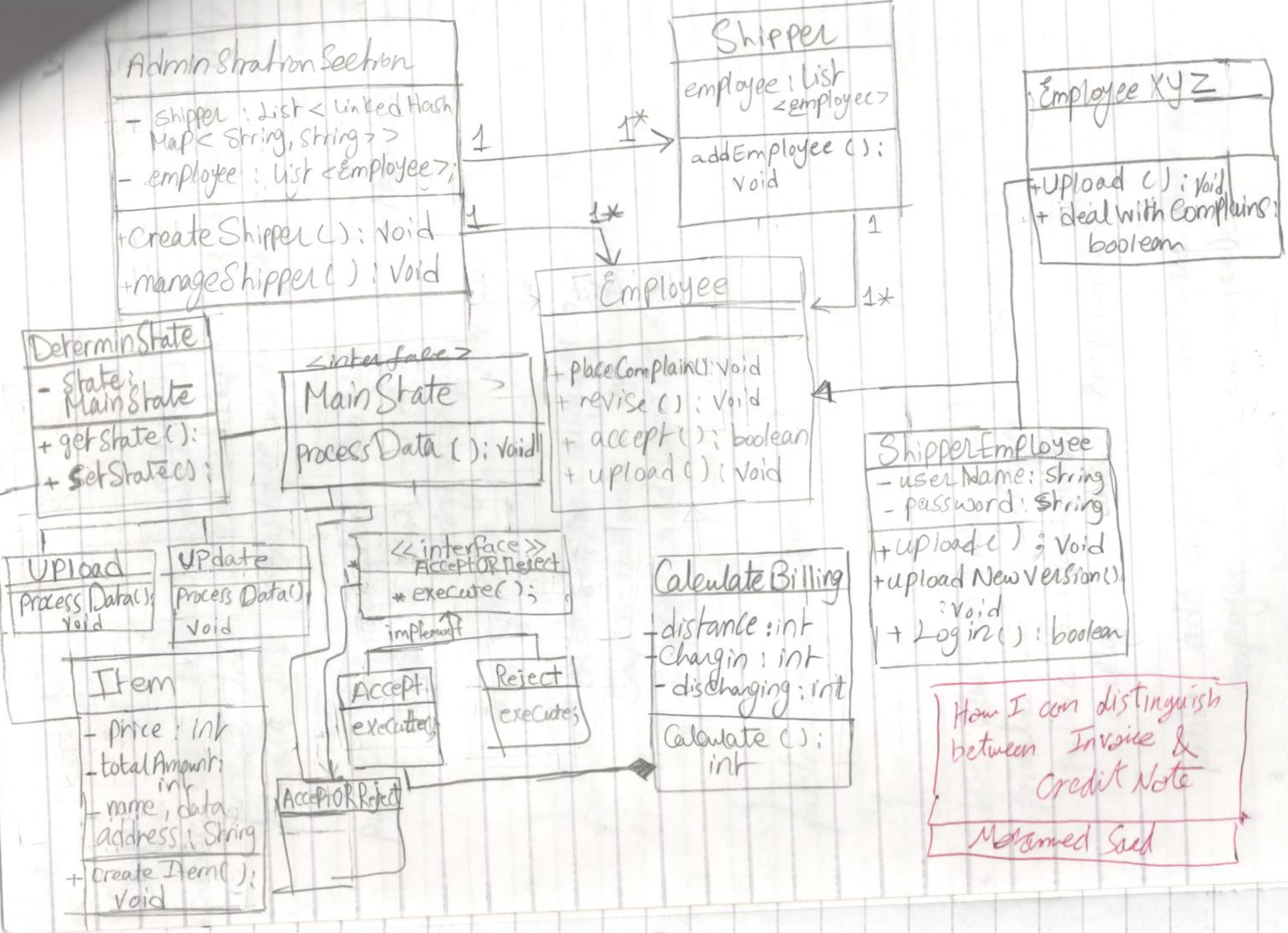
Skeleton

```
public interface MainState {
    Public boolean ProcessData (ArrayList<items> invoice);
}

Public class DetermineState {
    private MainState CurrentState;
    private MainState update;
    private MainState Accept;
    private MainState upload;
      //        ArrayList<Item> items;

    public DetermineState() {
            items = new ArrayList<>();
            update = new update(This);
            Accept = new Accept(this);
            upload = new Upload(This);
            Currentstate = Upload;
    }
    Setters ——> CurrentState
    getters ——> All the field.

    Public boolean ProcessData(ArrayList<items> invoice) {
        return CurrentState.ProcessData(invoice);
    }
```

DATE
PAGE

```
             Accept
             upload
Public class Update implements MainState{

    private DetermineState determineState ;
           accept
           update
    public Update (DetermineState state){
          determineState = state;

    }
    public boolean processData (List <items> items){

          Process data

    }
```

```
public class Items{

  private int price, total amount;
  private String date, name, addressOfCompany;

  Getters and Setters for all fields of the class

}
```



How I can distinguish between Invoice & Credit Note

Mohamed Saed

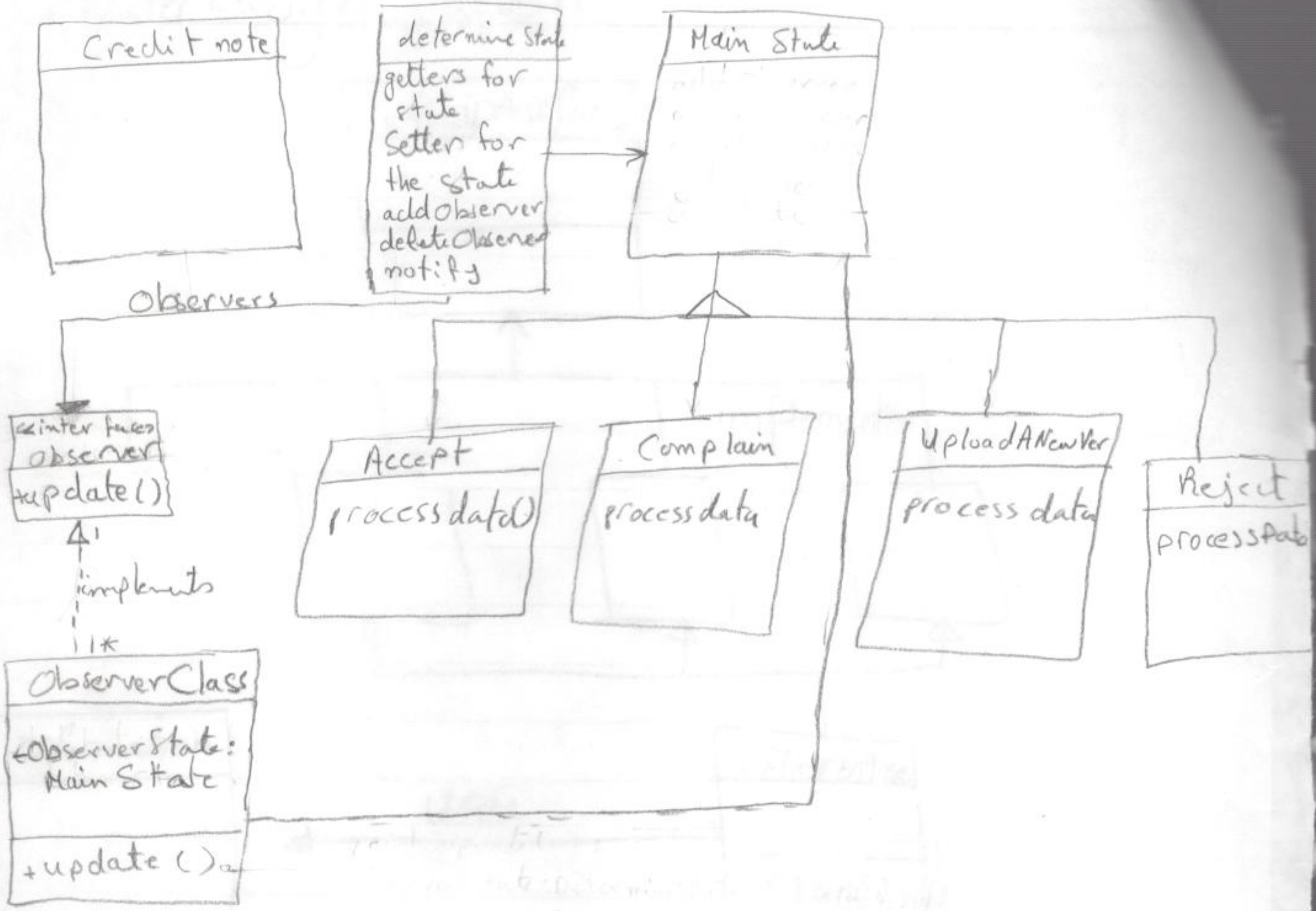
(Q4)

```
• public class AdminstrationSection () {
      private List <Linked HashMap <String, String> Shipper;
      private List <Employee> employer ;

   public AdminstrationSection () {
          shipper = new List <Linked Hash Map<String,
                        string> > {} ;
          employer = new Arraylist () ;
   }
   public createShipper ( String key , String value ) {
          Shipper customer = new Shipper () ;
          shipper . add (new Map < key, value >)


   }
   public manageShipper ( ) {


   }

• public class Shipper ( ) {
     private List <Employee> employer ;
       public Shipper ( ) {
          employer = new Employee () ;
       }
       public add employer (Employere employer) {
          employer . add ( employee) ;
       }.
```

```
public abstract class Employee {
    public Void plaleComplain () {
        // give the employer the ability to make
           Complain
    }
    public Void revise () {
        // revise shipper data
    }
    public boolean alleept () {
        // allept Shipperdata

    }
    abstract public Void upload () { }
}
public class ShipperEmployee extends Employee {
    private String userName;
    private String passward;
    public ShipperEmployee (String passward, String
                                    user Name) {
     This. user Name = user Name;
     This. pass ward = passward;
   }
  public Void upload () {
      // upload shipper's data.
  }
  public boolean Login (String name, String pass) {
      // Make sure of the passward and user
                                      Name
  }
```

```
public void uploadNewVersion () {
      // upload new version of Data.
  }

public class EmployeeXYZ extends Employee {
      public void upload () {
         // upload new data
    }
      public boolean dealWithComplains {
         // Some Strategy to deal with objections.
    }

public class CalculateBilling {
    private int distance;
    private int chagei
    private int discharge;
    public int calculate ( int distance, int charge) {
        // calculate the Bill with Some equations
    }

public class Item {
    private int price , totalAmount;
    private String address, name , data;
    public void create Item() {

    }
```

Qs) Observer, Observable and State design pattern



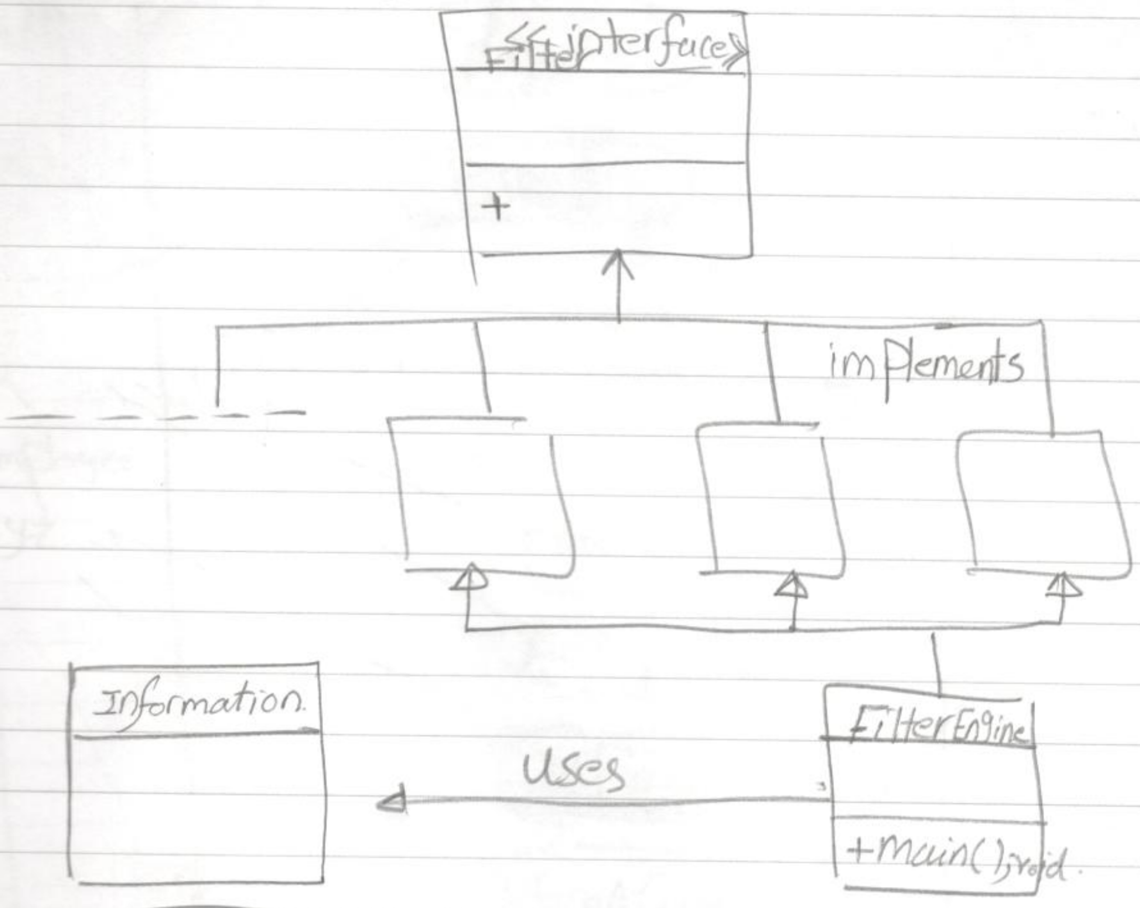
```
public interface Observer {
   public void update (Observable o, Object obj);
}

public class ObserverClass implements Observer {
  private Main State observerState;

  public void update (Observable o, Object obj) {
     observer State = o.get State;
     proccessing data according to State

  }
```

state classes same as question 2

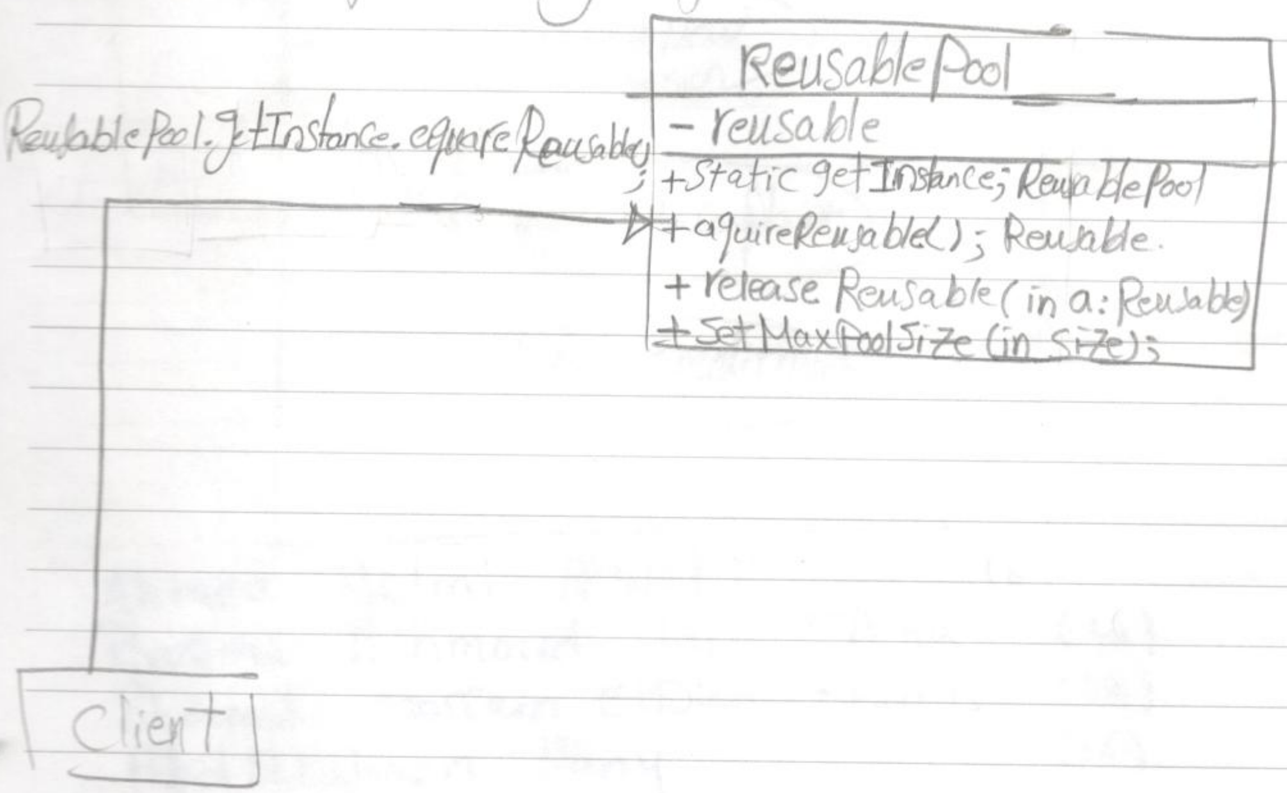
Q 6

DATE
PAGE

b) Filter design Pattern.



a) object Pool design Pattern



ReusablePool
- reusable
+Static getInstance; ReusablePool
+aquireReusable(); Reusable.
+ release Reusable(in a: Reusable)
+ SetMaxPoolSize (in Size);
ReusablePool.getInstance.equareReusable();
Client