DESIGN PATTERNS

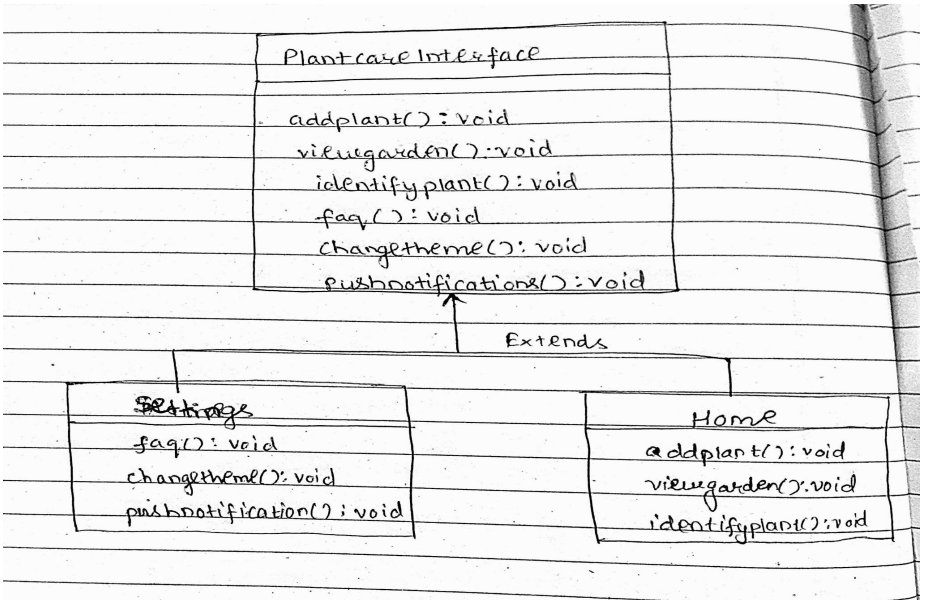
CREATIONAL PATTERN

1) FACTORY PATTERN:

Code:

```
interface PlantCareInterface {
  void addplant();
  void viewgarden();
  void identifyplant();
  void faq();
  void changetheme();
  void pushnotifications();
}


enum class PlantCareOption {
  HOME,
  SETTINGS
}


class HomePlantCareOption : PlantCareInterface {
  @override
  public void addplant();
  public void viewgarden(){
   viewinfo();
   removeplant();
   }
  public void identifyplant();
}


class SettingsPlantCareOption : PlantCareInterface {
  @override
  public void faq();
  public void changetheme();
  public void pushnotifications();
  }
}
```

```
object PlantCareFactory {

  fun getPlantCareFrom(type: PlantCareOption): PlantCareInterface {
    return when (option) {
      PlantCareOption.HOME -> {
        HomePlantCareOption()
      }
      PlantCareOption.SETTINGS -> {
        SettingsPlantCareOption()
      }
    }
  }
}
```



2) BUILDER PATTERN:

Code:
Plant.java:

```
public class Plant
{
  private string name;
  private boolean ispoisonous;
  private string kingdom;
  private string species;
  private float water;
```

```
  private float fertilizer;
  public plant(string name, boolean ispoisonous, string kingdom, string
species, float water, float fertilizer)
{
  super:
  this.name = name;
  this.ispoisonous;
  this.kingdom = kingdom;
  this.species = species;
  this.water = water;
  this.fertilizer = fertilizer;
}

@override
public String toString()
{
  return "Plant [name=" + name + ", Ispoisonous = "+ ispoisonous+ ",
kingdom=" +kingdom + ", species= " +species+ ", Amount of water
required per day =" +water + ", Amount of Fertilizer required per week
=" + fertilizer + "]";
}

}
```

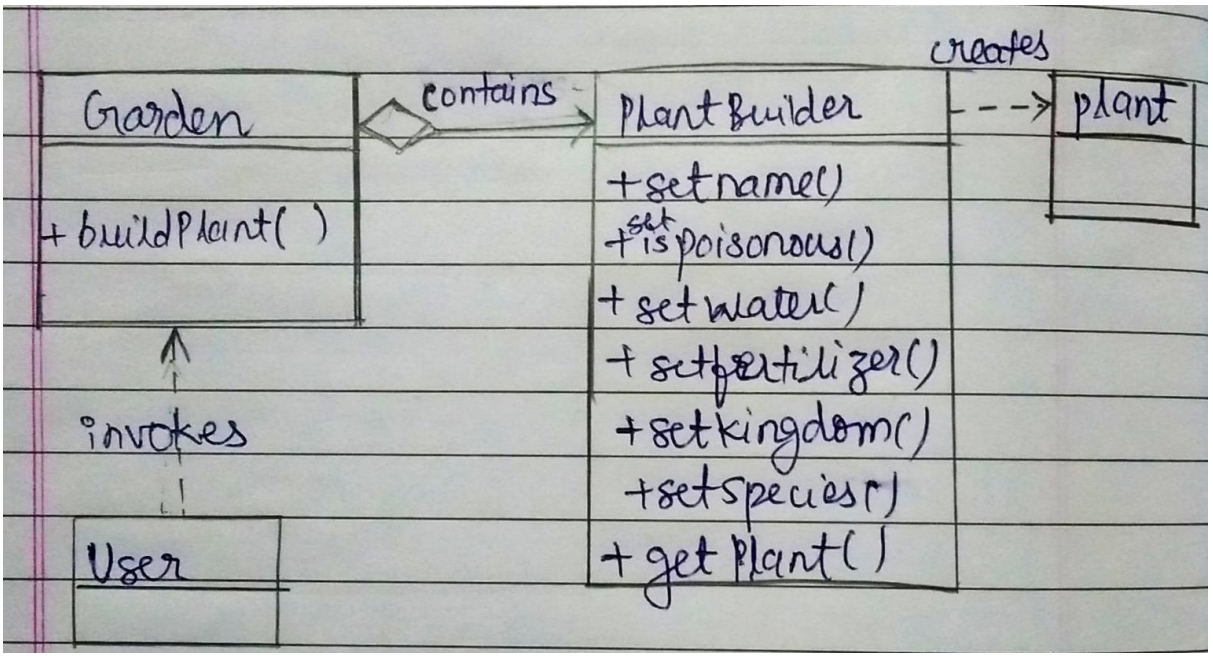
garden.java:

```
public class garden
{
  public static void main(String a[])
  {
     Plant p = new PlantBuilder().setname(String
name).setispoisonous(boolean ispoisonous).setkingdom(String
kingdom).setspecies(String species).setwater(float
water).setfertilizer(float fertilizer).getPlant();
   System.out.println(p);
}
}
```

plantbuilder.java:

```
public class PlantBuilder
{
  private:
  string name;
  boolean ispoisonous;
  string kingdom;
  string species;
  float water;
  float fertilizer;

  public PlantBuilder setname(String name)
  {
    this.name = name;
     return this;
  }

  public PlantBuilder setispoisonous(boolean ispoisonous)
  {
  this.ispoisonous = ispoisonous;
  return this;
  }

  public PlantBuilder setkingdom(String kingdom)
  {
  this.kingdom = kingdom;
  return this;
  }

  public PlantBuilder setspecies(String species)
  {
    this.species = species;
    return this;
  }

  public PlantBuilder setwater(float water)
  {
    this.water = water;
    return this;
  }
```

```
   public PlantBuilder setfertilizer(float fertilizer)
  {
    this.fertilizer = fertilizer;
    return this;
   }

  public Plant getPlant()
  {
     return new Plant(name, ispoisonous, kingdom, species, water,
fertilizer);
  }
}
```



BEHAVIOURAL PATTERN

1) COMMAND PATTERN:

Code:

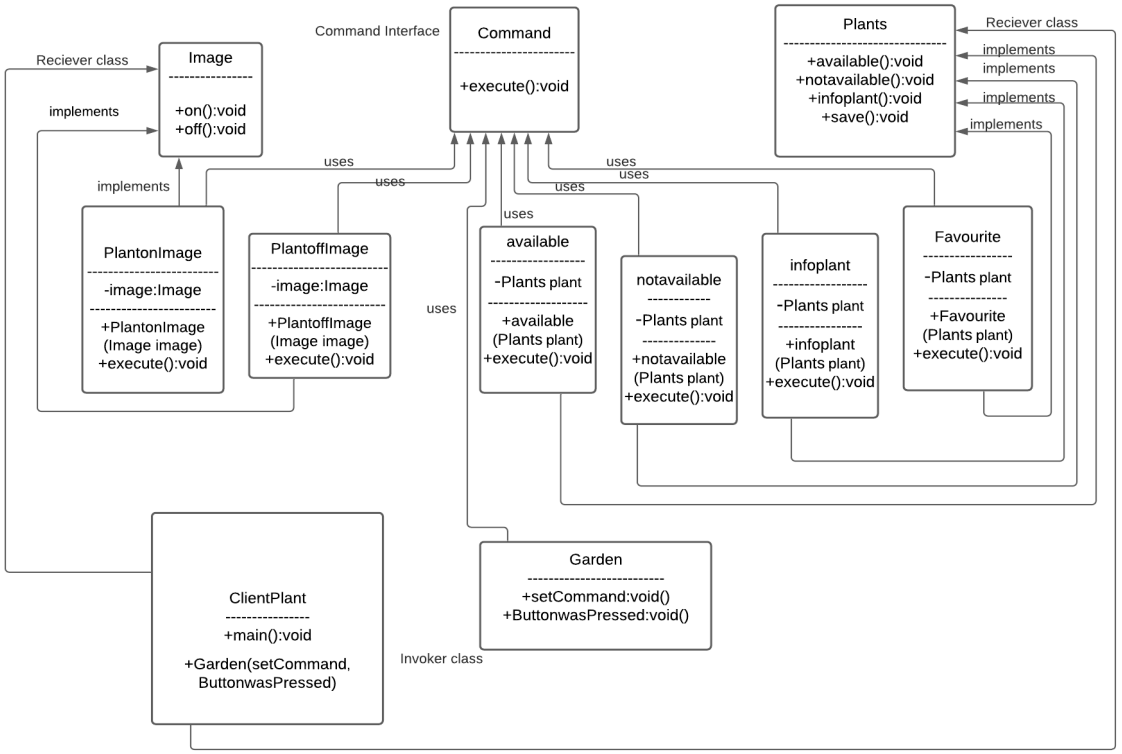
```
interface Command
{
    public void execute();
}

// Image Class with Google Lens and its corresponding command
// classes
class Image
```

```
{
    public void on(int Name)
    {
        System.out.println("Plant name is " name);
    }
    public void off()
    {
        System.out.println("Plant is not identified");
    }
}
class PlantonImage implements Command
{
    Image image;

    // The constructor is passed the plant to check
    // plant is available or not
    public PlantonImage (Image image)
    {
        this.image = image;
    }
    public void execute()
    {
    image.on();
    }
}
class PlantoffImage implements Command
{
    Image image;
    public PlantoffImage (Image image)
    {
        this.image = image;
    }
    public void execute()
    {
        image.off();
    }
}

// Searching Plants Class andand its corresponding command
// classes
class Plants
{
    public void available()
```

```
    {
        System.out.println("Plant is available ");
    }
    public void notavailable()
    {
        System.out.println("Plant is not available");
    }
    public void infoplant(int Name,Detail)
    {
        System.out.println("Plant"+ Name + "Detail:" + Detail );
    }
    public void save(){
            System.out.println("Plant Saved");
        }

}
class notavailable implements Command
{
    Plants plant;
    public notavailable (Plants plant)
    {
        this.plant = plant;
    }
    public void execute()
    {
        plant.notavailable();
    }
}


class available implements Command
{
    Plants plant;
    public available (Plants plant)
    {
        this.plant = plant;
    }
    public void execute()
    {
        plant.available();
    }
}
```

```
class infoplant implements Command
{
    Plants plant;
    public infoplant(Plants plant)
    {
        this.plant = plant;
    }
    public void execute()
    {
        plant.available();
        plant.infoplant();
    }
}

class Favourite implements Command
{
    Plants plant;
    public Favourite(Plants plant)
    {
        this.plant = plant;
    }

    @Override
    public void execute()
    {
        plant.save();
        plant.Favourite();
    }
}

// Garden with one button
class Garden
{
    Command slot; // only one button

    public Garden()
    {
    }

    public void setCommand(Command command)
    {
        // set the command the plant in garden
        // execute
```

```
        slot = command;
    }

    public void buttonWasPressed()
    {
        slot.execute();
    }
}


// Client Class
class ClientPlant
{
    public static void main(String[] args)
    {
        Garden fav= new Garden();
        Image image = new Image();
        Plants plant= new Plants();

        // we can change command dynamically
        fav.setCommand(new PlantonImage(image));
        fav.buttonWasPressed();
        fav.setCommand(new available(plant));
        fav.buttonWasPressed();
        fav.setCommand(new infoplant(plant));
        fav.buttonWasPressed();
        fav.setCommand(new Favourite(plant));
        fav.buttonWasPressed();
    }
}
```

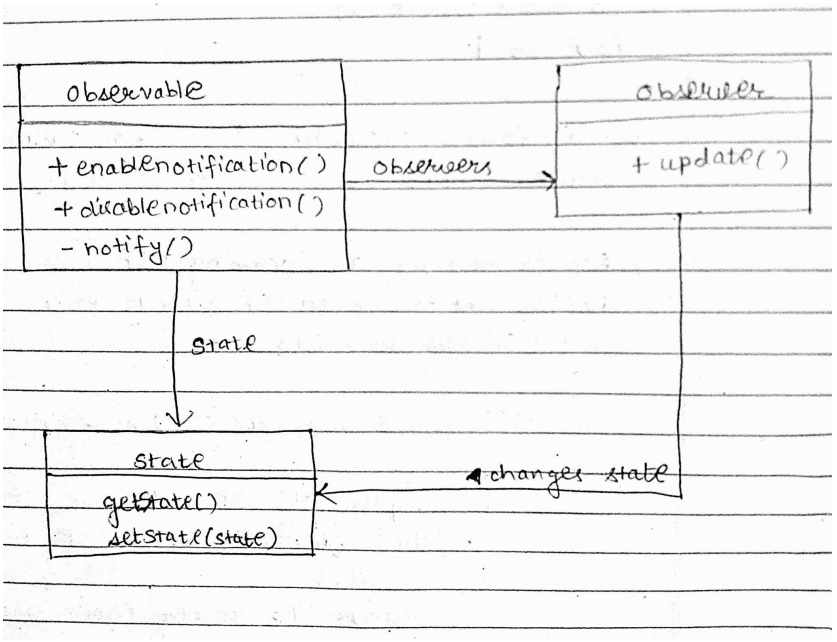


2) OBSERVER PATTERN:
code:

```
class PlantData
{
    string name;
    float water, fertilizer, time;
    Notification notification;

    // Constructor
    public PlantData(Notification notification)
    {
        this.notification = notification;
    }


    // Get timely updates regarding the amount of water to be watered
    private int getWater()
    {
        // return 1.5 litres for simplicity
        return 1.5;
    }

    // Get timely updates regarding amount of fertilizer to be given
```

```
    private int getFertilizer()
    {
        // return 0.4 kg for simplicity
        return 0.4;
    }

    // Get time interval after which water is to be watered
    private float gettime()
    {
        // return 10.2 for simplicity
        return (float)10.2;
    }

    // This method is used give notification when it is time to water
plants
    public void dataChanged()
    {
        // get latest data
        water = getWater();
        fertilizer = getFertilizer();
        time_interval1 = gettime();
        notification.update(water,fertilizer,time);
     }
}


// A class to display timely the amount of water and fertilizer to be
added
class Notification
{
    private float water, fertilizer;
    private float time_interval;

    public void update(float water,float fertilizer,float
time_interval)
    {
        this.water = water;
        this.fertilizer = fertilizer;
        this.time_interval = time_interval1;
        display();
    }

    public void display()
```

```
    {
        System.out.println("\nGive " +
                           water +"Litres of water and add"+ fertilizer
+ "grams of fertilizer to the soil \n" );
    }
}

// Driver class
class Main
{
    public static void main(String args[])
    {
        // Create objects for testing
        Notification notification = new Notification();


        // Pass the displays to Plant data
        PlantData plantData = new PlantData(notification);

        // In real app you would have some logic to call this
        // function when data changes
        plantData.dataChanged();
    }
}
```



STRUCTURAL PATTERN

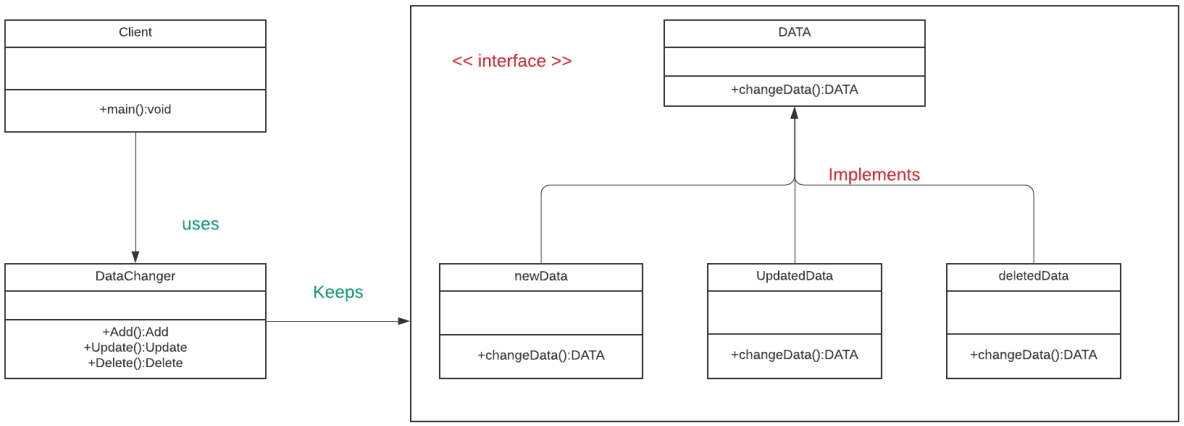
1) FACADE PATTERN

CODE:

```
package structural.facade;

public interface DATA
{
    public Data changeData();
}



package structural.facade;

public class newData implements DATA
{
    public DATA changeData()
    {
        Add adder = new Add();
        return adder;
    }
}


package structural.facade;

public class updatedData implements DATA
{
    public DATA changeData()
    {
        Update updater = new Update();
        return updater;
    }
}


package structural.facade;

public class deletedData implements DATA
{
    public DATA changeData()
```

```
    {
        Delete del = new Delete();
        return del;
    }
}



package structural.facade;

public class DataChanger
{
    public Update updateData()
    {
        updatedData updater = new updatedData();
        Update Update = (Update)updater.changeData();
        return Update;
    }

    public Add addData()
    {
        newData updater = new newData();
        Add Add = (Add)updater.changeData();
        return Add;
    }

    public Delete deleteData()
    {
        deletedData updater = new deletedData();
        Delete del = (Delete)updater.changeData();
        return del;
    }
}



package structural.facade;

public class Client
{
    public static void main (String[] args)
    {
        DataChanger changer = new DataChanger();
```

```
        Update updater = changer.updateData();
        Add adder = changer.addData();
        Delete del= changer.deleteData();

    }
}
```

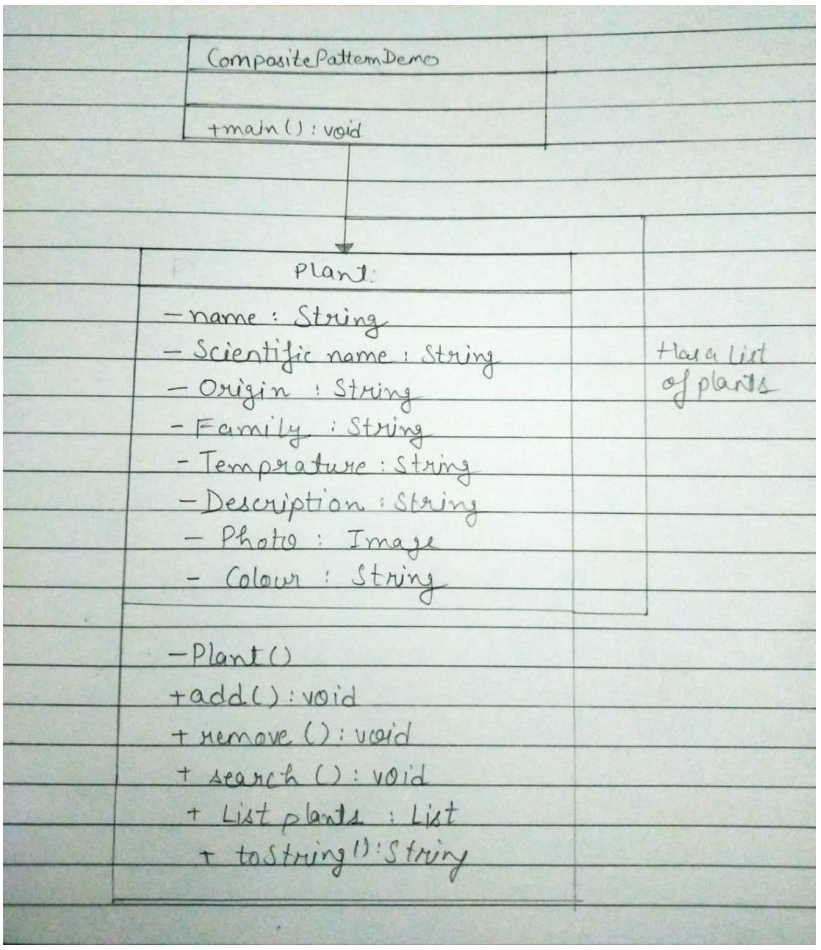


2) COMPOSITE PATTERN:

CODE:

```
import java.util.ArrayList;
import java.util.List;
public class Plant
{
  private String name;
  private String scientificname;
  private String origin;
  private String family;
  private String temprature;
  private String description;
  private String colour;
  public Plant (String name, String scientificname, String origin,
        String family, String temprature, String description,
        String colour)
  {
    this.name = name;
    this.scientificname = scientificname;
    this.origin = origin;
    this.family = family;
```

```
    this.temprature = temprature;
    this.description = description;
    this.colour = colour;
    public String toString ()
    {
      return ("Plant :[ Name : " + name + ", Scientific name : " +
          scientificname + ", origin :" + origin + ", family : " +
          family + ", Temprature : " + temprature + ", Description : "
+
          description + ", Colour : " + colour + "]");
    }
  }



  public class CompositePatternDemo
  {

    public static void main (String[]args)
    {

      Plant AfricanViolet =
    new Plant ("African Violet", "Saintpaulia", "East Africa",
          "Gesneriaceae", "24C",
          "African violets are one of the world's most popular
houseplants and for good reason. These compact, low-growing plants
flower several times a year, and they are available in a multitude of
leaf forms and colors",
          "Violet");

      Plant Viola =
    new Plant ("Viola", "Viola reichenbachiana", "United States",
          "Violaceae", "20C",
          "Viola typically have heart-shaped or reniform
(kidney-shaped), scalloped leaves, though a number have linear or
palmate leaves. The simple leaves of plants with either habit are
arranged alternately; the acaulescent species produce basal rosettes",
          "Violet");
        System.out.println (AfricanViolet);

    }
  }
```


CompositePatternDemo
+main() : void
Plant
-name : String
- Scientific name : String
- Origin : String
- Family : String
- Temprature : String
- Description : String
- Photo : Image
- Colour : String
-Plant()
+add() : void
+ remove() : void
+ search() : void
+ List plants : List
+ toString() : String
Has a List of plants