CSC 8350 – Advanced Software Engineering

GitHub: https://tinyurl.com/mv7c568n

(Please do visit for better understanding)

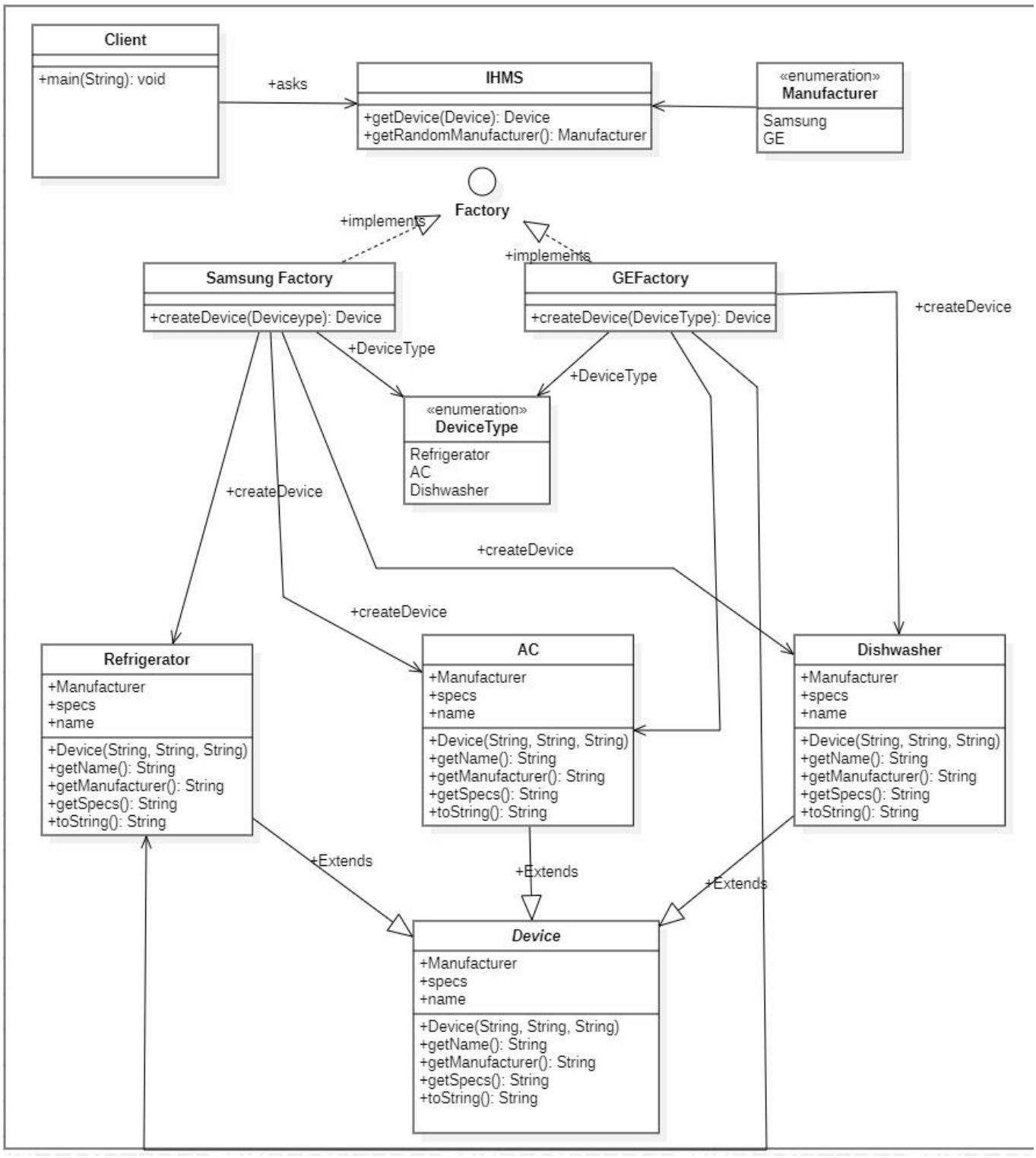
Sai Kowshik Ananthula

002638298

MIDTERM-1

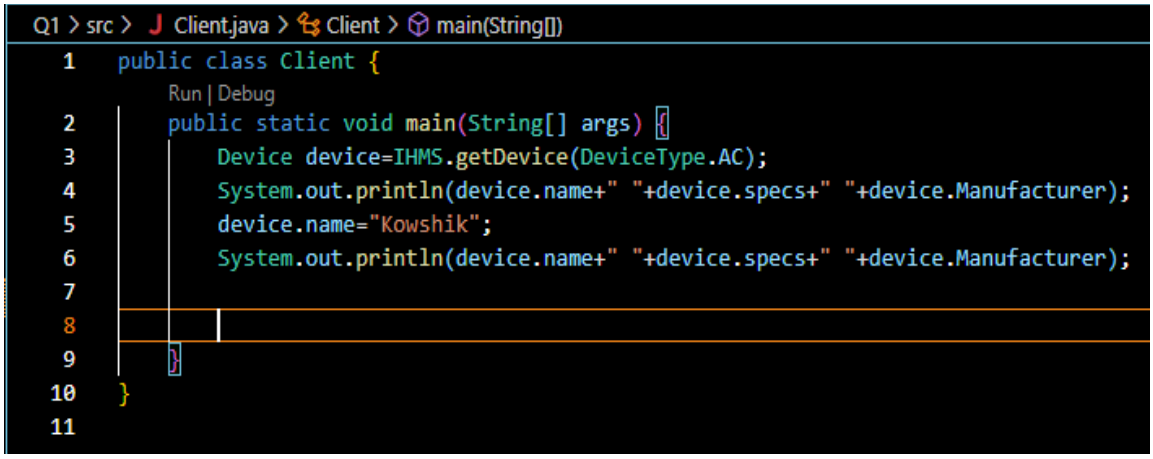
Q1.

Class Diagram:



Client.java

This method is used by the client to call Abstract factory for a device object.

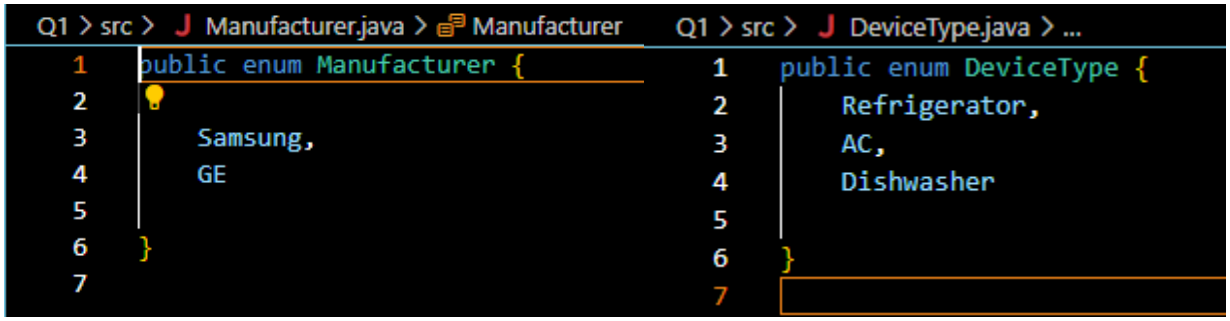


IHMS.java

This class ascts as a abstract factory and is to return newly created factory objects based on client calls. It has a getRandomManufacturer() method to genetrate random manufactures and calls its getDevice() methods.

Enum classes: These classes are custom defined data type for both Manufacturer as well as Devices. The need for these classes is to add any number of devices and manufacturers in near future.

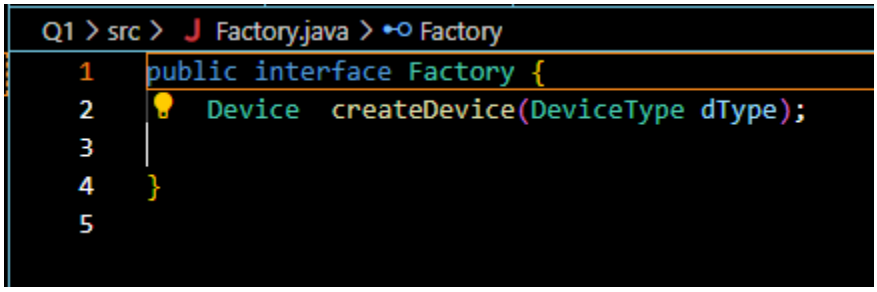
Manufacturer.java **DeviceType.java**



```
Q1 > src > Manufacturer.java > Manufacturer
1  public enum Manufacturer {
2
3      Samsung,
4      GE
5
6  }
7
```

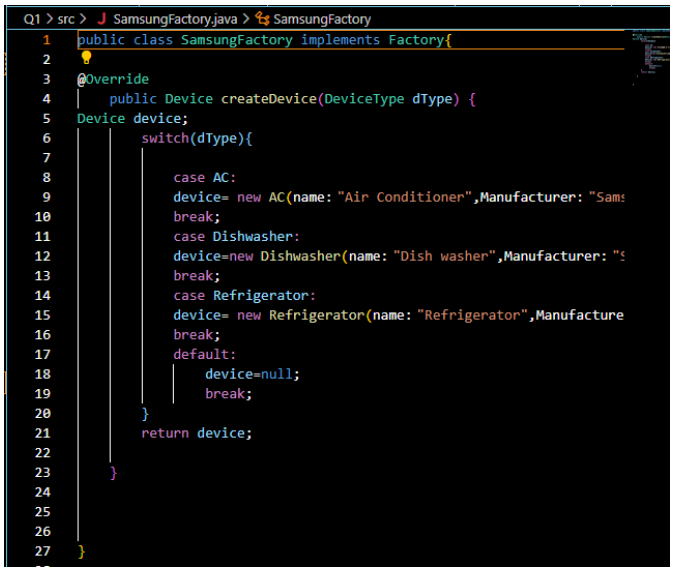
```
Q1 > src > DeviceType.java > ...
1  public enum DeviceType {
2      Refrigerator,
3      AC,
4      Dishwasher
5
6  }
7
```

Factory.java: This Interface provides the behaviour for the classes like how a factory should be and what are the methods and functions it should contain.



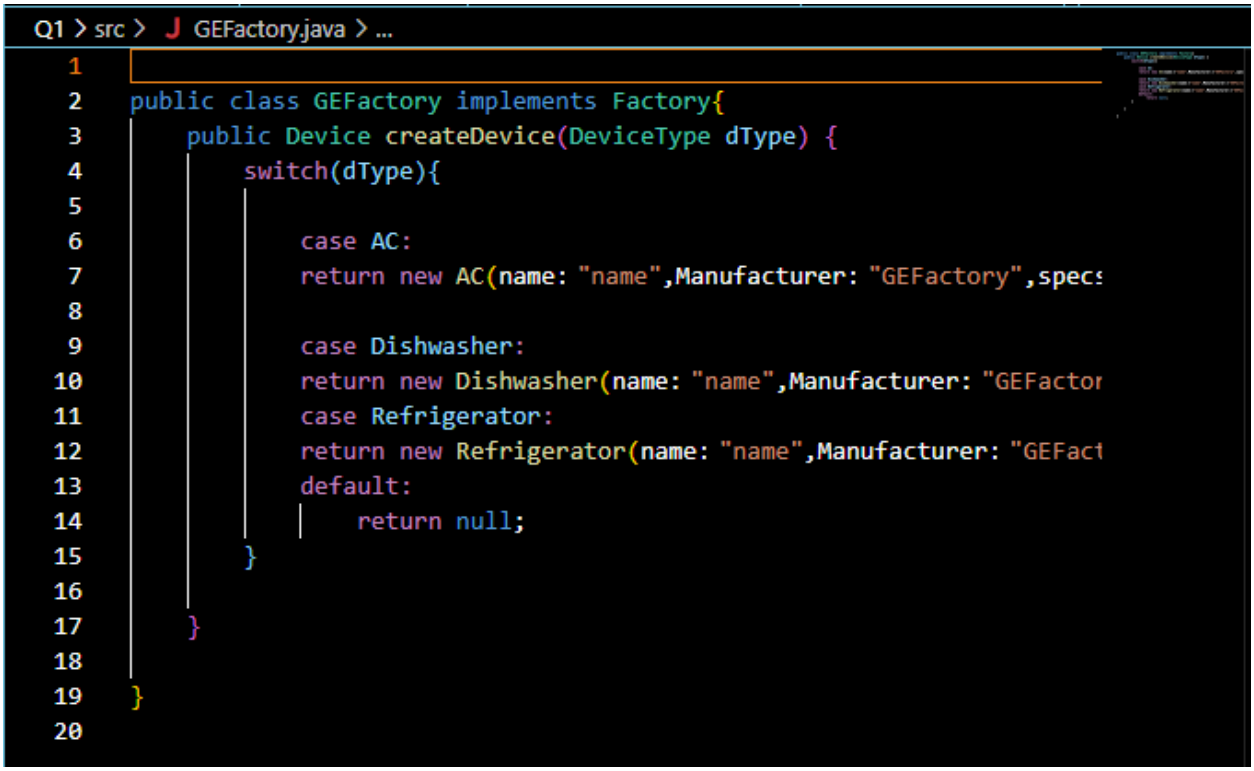
```
Q1 > src > Factory.java > Factory
1  public interface Factory {
2      Device  createDevice(DeviceType dType);
3
4  }
5
```

SamsungFactory.java: This class implements the Factory interface and override its methods and returns a device object based on client's requirements. This is a concrete class which provides a device form Samsung Factory.

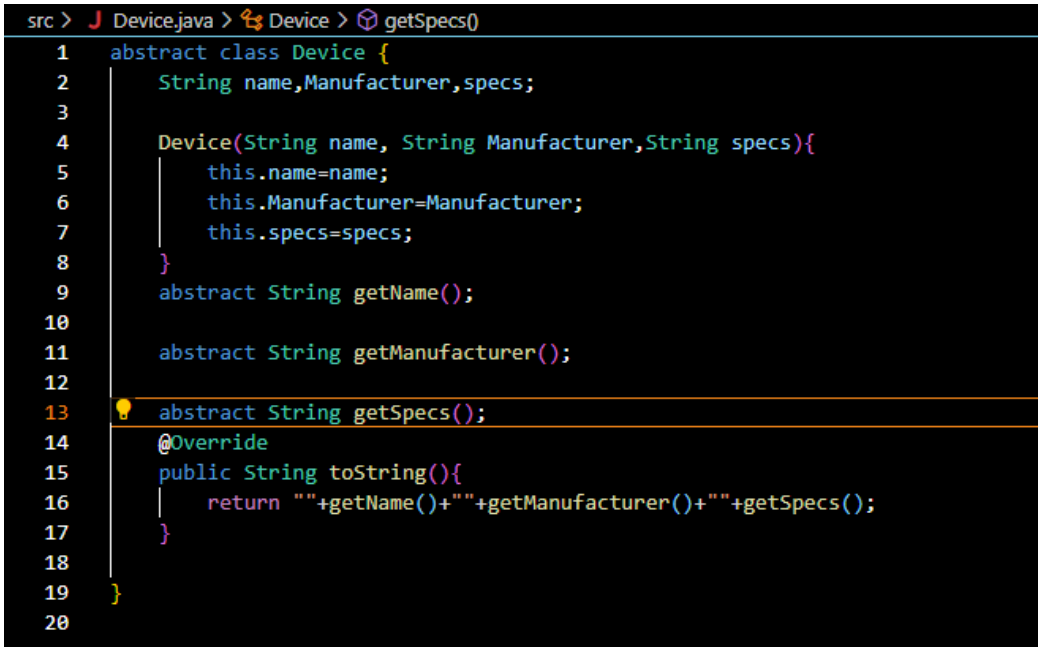


```
Q1 > src > SamsungFactory.java > SamsungFactory
1  public class SamsungFactory implements Factory{
2
3  @Override
4      public Device createDevice(DeviceType dType) {
5  Device device;
6          switch(dType){
7
8              case AC:
9              device= new AC(name: "Air Conditioner",Manufacturer: "Sams
10             break;
11             case Dishwasher:
12             device=new Dishwasher(name: "Dish washer",Manufacturer: "
13             break;
14             case Refrigerator:
15             device= new Refrigerator(name: "Refrigerator",Manufacture
16             break;
17             default:
18                 device=null;
19                 break;
20         }
21         return device;
22
23     }
24
25
26
27 }
```

GEFactory.java: This class implements the Factory interface and override its methods and returns a device object based on client's requirements. This is a concrete class which provides a device form GEFactory.



Device.java: This is an abstract class that provides the behaviour of the devices that are available in the factories. It contains both abstract methods as well as non-abstract methods.



AC.java: This class extends the abstract class Device. It acts as a container for both properties and operations that an Air Conditioner performs.

Q1 > src > AC.java > ...

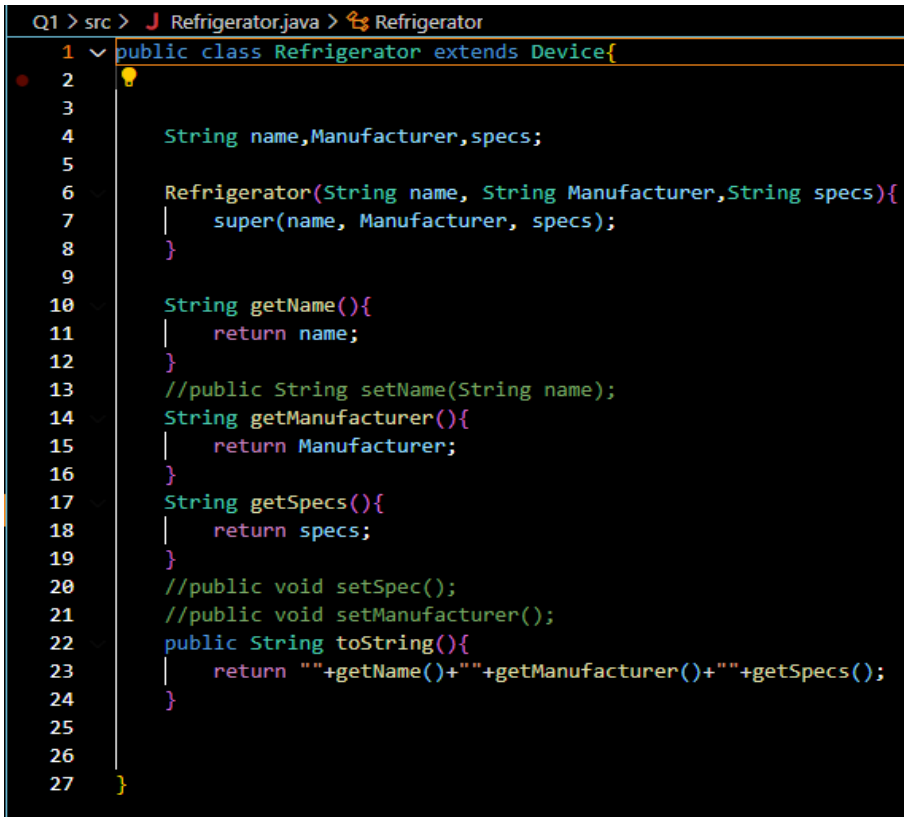
```java
public class AC extends Device{

    String name,Manufacturer,specs;

    AC(String name, String Manufacturer,String specs){
        super(name, Manufacturer, specs);
    }

    String getName(){
        return name;
    }
    //public String setName(String name);
    String getManufacturer(){
        return Manufacturer;
    }
    String getSpecs(){
        return specs;
    }
    //public void setSpec();
    //public void setManufacturer();
    public String toString(){
        return ""+getName()+""+getManufacturer()+""+getSpecs();
    }


}
```

Dishwasher.java: This class extends the abstract class Device. It acts as a container for both properties and operations that a Dish Washer performs.

Q1 > src > Dishwasher.java > Dishwasher

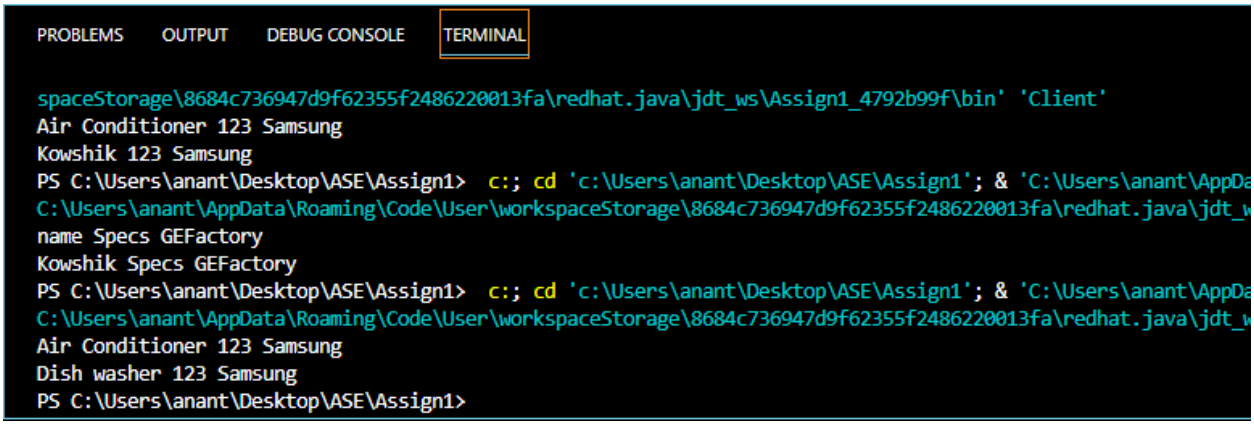
```java
public class Dishwasher extends Device{

    String name,Manufacturer,specs;

    Dishwasher(String name, String Manufacturer,String specs){
        super(name, Manufacturer, specs);
    }

    String getName(){
        return name;
    }
    //public String setName(String name);
    String getManufacturer(){
        return Manufacturer;
    }
    String getSpecs(){
        return specs;
    }
    //public void setSpec();
    //public void setManufacturer();
    public String toString(){
        return ""+getName()+""+getManufacturer()+""+getSpecs();
    }


}
```

Refrigerator.java: This class extends the abstract class Device. It acts as a container for both properties and operations that a Refrigerator performs.



```
Q1 > src > Refrigerator.java > Refrigerator
1  public class Refrigerator extends Device{
2
3
4      String name,Manufacturer,specs;
5
6      Refrigerator(String name, String Manufacturer,String specs){
7          super(name, Manufacturer, specs);
8      }
9
10     String getName(){
11         return name;
12     }
13     //public String setName(String name);
14     String getManufacturer(){
15         return Manufacturer;
16     }
17     String getSpecs(){
18         return specs;
19     }
20     //public void setSpec();
21     //public void setManufacturer();
22     public String toString(){
23         return ""+getName()+""+getManufacturer()+""+getSpecs();
24     }
25
26
27 }
```

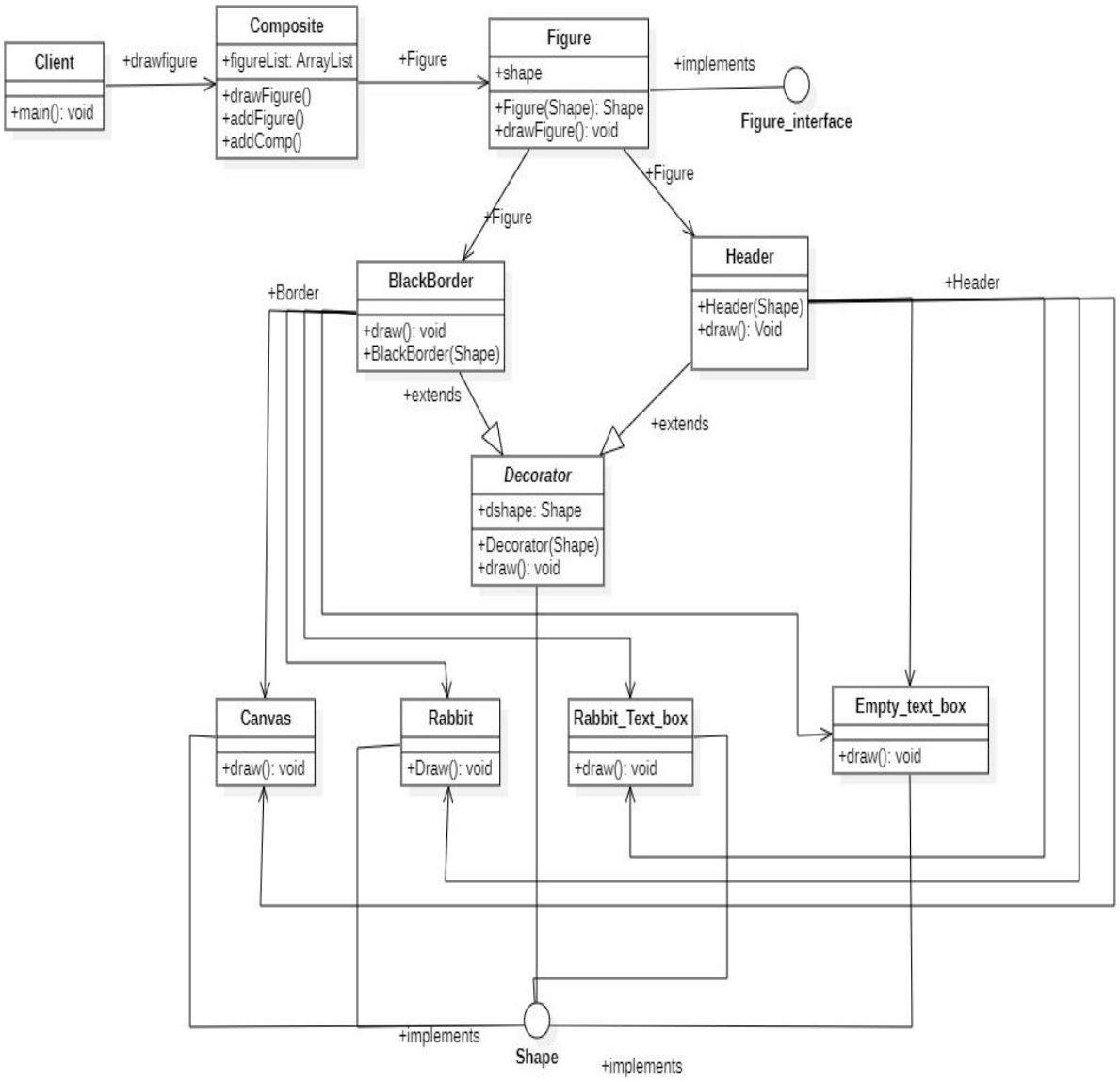
Output: When I asked IHMS.java for an AC, It returned me a Samsung AC. Later I tried changed the name of that device to 'Kowshik'. Without creating any object in the client class and without showing any concrete methods, This is a solution I came up with to produce a IHMS using Abstract Factory Design.



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

spaceStorage\8684c736947d9f62355f2486220013fa\redhat.java\jdt_ws\Assign1_4792b99f\bin' 'Client'
Air Conditioner 123 Samsung
Kowshik 123 Samsung
PS C:\Users\anant\Desktop\ASE\Assign1>  c:; cd 'c:\Users\anant\Desktop\ASE\Assign1'; & 'C:\Users\anant\AppDa
C:\Users\anant\AppData\Roaming\Code\User\workspaceStorage\8684c736947d9f62355f2486220013fa\redhat.java\jdt_w
name Specs GEFactory
Kowshik Specs GEFactory
PS C:\Users\anant\Desktop\ASE\Assign1>  c:; cd 'c:\Users\anant\Desktop\ASE\Assign1'; & 'C:\Users\anant\AppDa
C:\Users\anant\AppData\Roaming\Code\User\workspaceStorage\8684c736947d9f62355f2486220013fa\redhat.java\jdt_w
Air Conditioner 123 Samsung
Dish washer 123 Samsung
PS C:\Users\anant\Desktop\ASE\Assign1>
```

Q2.

Class Diagram:



Pseudocode:

Client.java: This class is used to create all the atomic decorative and composite objects and glue them together.

```
3   public class Client {
        Run | Debug
4       public static void main(String [] args){
5           //Drawing Rabbit with header and border
6           Shape border= new BlackBorder(new Rabbit());
7           Shape header= new Header(border);
8           Figure rabbit_header_border= new Figure(header);
9
10          // Drawing Rabbit_Text_box with border
11          Shape rabbit_text_box= new Rabbit_Text_box();
12          Shape border2= new BlackBorder(rabbit_text_box);
13          Figure rabbit_textBox_border= new Figure(border2);
14
15          //Merging Both the figures
16          Composite comp= new Composite();
17          Composite comp1= new Composite();
18          comp1.addFigure(rabbit_textBox_border);
19          comp1.addFigure(rabbit_header_border);
20
21          //Drawing Empty_text_box with header and border
22          Shape border3= new BlackBorder(new Empty_text_box());
23          Shape header3=new Header(border3);
24          Figure empty_box_with_border_and_header= new Figure(header3);
25
26
27          //Drawing header and border
28          Shape border4= new BlackBorder(new Canvas());
29          Shape header4= new Header(border4);
30          Figure canvas_withborder= new Figure(header4);
31
32
33          comp.addFigure(canvas_withborder);
34          comp.addComp(comp1);
35          comp.addFigure(empty_box_with_border_and_header);
36          comp.drawFigure();
37      }
```

Canvas.java: This class provides the initial object to draw on.

```java
public class Canvas implements Shape{

    @Override
    public void draw() {
        System.out.println(x: "Canvas has been created!!");

    }


}

```

BlackBorder.java: This class provides a BlackBorder that extends Decorator abstract class. It requires a shape object to draw on.

```java
public class BlackBorder extends Decorator {

    public BlackBorder(Shape dShape) {
        super(dShape);
    }
    @Override
    public void draw(){
        dShape.draw();
        setBlackBorder(dShape);
    }
    private void setBlackBorder(Shape decoratedShape)
        {
            System.out.println(x: "Border has been added and it's Color
        }
    }

```

Composite.java: This class can combine a composite with another composite object or a figure object with another composite object or two figure objects as a single composite object.

```java
import java.util.ArrayList;
import java.util.List;

public class Composite implements Figure_interface {
    private List<ArrayList<Figure>> figureList = new ArrayList<>();
    @Override
    public void drawFigure() {
        figureList.forEach((list)->{
        list.forEach((figure)-> figure.drawFigure());
        System.out.println(x: "End of Root!");
        }
        );

    }
    public void addFigure(Figure fig){
        ArrayList<Figure> figList= new ArrayList<Figure>();
        figList.add(fig);
        figureList.add(figList);
    }
    public void addComp(Composite comp){
        ArrayList <Figure> figList= new ArrayList<>();
        for (ArrayList<Figure> fig:comp.figureList){
          figList.addAll(fig);
        }
        figureList.add(figList);
    }


}

```

Decorator.java: This is an abstract class which provides behaviour of both header and BlackBorder class.

```java
public abstract class Decorator implements Shape{
    protected Shape dShape;
    public Decorator(Shape decoratedShape)
    {
        this.dShape = decoratedShape;
    }

    public void draw() { dShape.draw(); }
}

```

Empty_text_box.java: This is an atomic class that provides empty text box object.

```java
public class Empty_text_box implements Shape{

    @Override
    public void draw() {
        System.out.println(x: "Drawing Empty_Text_Box....");

    }


}
```

Figure_Interface.java: This class provides structure for figure class.

```java
public interface Figure_interface {
    public void drawFigure();
}

```

Figure.java: This class implements figure interface and can fit a shape object and its properties.

```
1
2   public class Figure implements Figure_interface {
3       private Shape shape;
4       Figure(Shape shape){
5           this.shape=shape;
6
7       }
8
9       @Override
10      public void drawFigure() {
11          shape.draw();
12      }
13  }
```

Header.java: This class extends shape object and holds properties of rabbit figure.

```
1   public class Header extends Decorator {
2
3           public Header(Shape dShape) {
4               super(dShape);
5           }
6           @Override
7           public void draw(){
8               dShape.draw();
9               setBlackBorder(dShape);
10          }
11          private void setBlackBorder(Shape decoratedShape)
12              {
13                  System.out.println(x: "Header has been added to the fig
14              }
15  }
16
```

Rabbit_Text_box.java: This class implements shape interface and holds properties of rabbit text box figure.

```java
public class Rabbit_Text_box implements Shape{

    @Override
    public void draw() {
        System.out.println(x: "Drawing Rabbit_text_box....");

    }


}
```

Rabbit.java: This class implements shape interface and holds properties of rabbit figure.

```java
public class Rabbit implements Shape{

    @Override
    public void draw() {
        System.out.println(x: "Drawing Rabbit....");

    }


}
```

Shape.java: This interface provides structure for atomic objects like rabbit, empty text box etc.

```java
public interface Shape{

    public void draw();
}

```

Output:

```
Canvas has been created!!
Border has been added and it's Color is Black!
Header has been added to the figure!
End of Root!
Drawing Rabbit_text_box....
Border has been added and it's Color is Black!
Drawing Rabbit....
Border has been added and it's Color is Black!
Header has been added to the figure!
End of Root!
Drawing Empty_Text_Box....
Border has been added and it's Color is Black!
Header has been added to the figure!
End of Root!
```

Object Diagram: