Question 1

You have been asked to redesign the software architecture for an appliance emulator that models five different categories of stoves shown in Table 1. Each stove type reacts differently when a burner is turned on or turned off. Gas stoves use natural gas, ignited by a human user, to heat a burner whereas electric stoves use electricity to power a burner. Modular stoves use electricity to ignite gas when a burner is turned on, whereas downdraft stoves may not necessarily have this feature. Rather, they have an exhaust fan to pull fumes out of the air. Lastly a smart stove triggers an auto clean function when a burner is turned off and these tend to be electric.

Stove Type	Burner On	Burner Off
Gas	Turns on the gas. Ignition is handled by the user	Turns off the gas.
Electric	Turns on electricity. (Burner is heated electrically)	Turns off electricity.
Downdraft	Turns on the gas. Turns on the exhaust fan. Ignition may handled by the user if the stove is gas.	Turns off the gas. Turns off the exhaust fan.
Modular	Turns on the gas. Ignition is triggered electrically	Turns off the gas.
Smart	Turns on electricity.	Turns off electricity. Automatic cleaning of cooktop triggered.

Table 1. Stove Types and the Events for Turning a Burner On and Off

Suppose the approaches shown in Figures 1-4 were used by a former developer to model the functionality of the various types of stoves as cooktops (concrete manifestation of a stove type).

```
public interface Stove {
    public void turnOnBurner(int burnerNum);
    public void turnOffBurner(int burnerNum);
    public void turnOnGas(int burnerNum);
    public void turnOffGas(int burnerNum);
    public void cleanCooktop();
    public void turnOnElectricity(int burnerNum);
    public void turnOffElectricity(int burnerNum);
    public void turnOnExhaust();
    public void turnOffExhaust();
    public void igniteElectrically(int burnerNum);
}
```

Figure 1. Stove Interface Methods

```
public class Cooktop implements Stove {
    private String primaryStoveType;
    private String secondaryStoveType;
    private String stoveName;
    public Cooktop(String stoveName, String primaryStoveType){
        this.stoveName = stoveName;
        this.primaryStoveType = primaryStoveType;
        secondaryStoveType = "";
    }
    public Cooktop(String stoveName, String primaryStoveType, String secondaryStoveType){
        this.stoveName = stoveName;
        this.primaryStoveType = primaryStoveType;
        this.secondaryStoveType = secondaryStoveType;
    }
```

Figure 2. Cooktop Implementation - Class Definition, State, Constructors

```
    public void turnOnBurner(int burnerNum){
        switch(primaryStoveType){
            case "GasStove":
                turnOnGas(burnerNum);
                if(secondaryStoveType.equals("ModularStove"))
                    igniteElectrically(burnerNum);
                break;
            case "DowndraftStove":
                turnOnGas(burnerNum);
                if(secondaryStoveType.equals("ModularStove"))
                    igniteElectrically(burnerNum);
                turnOnExhaust();
                break;
            case "ElectricStove":
                turnOnElectricity(burnerNum);
                break;
            case "ModularStove":
                igniteElectrically(burnerNum);
                break;
        }
        System.out.println(stoveName+ " Burner: " + burnerNum + " on");
    }

    public void turnOffBurner(int burnerNum){
        switch(primaryStoveType){
            case "GasStove":
                turnOffGas(burnerNum);
                break;
            case "DowndraftStove":
                turnOffGas(burnerNum);
                turnOffExhaust();
                break;
            case "ElectricStove":
                turnOffElectricity(burnerNum);
                if(secondaryStoveType.equals("SmartStove"))
                cleanCooktop();
                break;
        }
        System.out.println(stoveName+  " Burner: " + burnerNum + " off");
    }

    public void turnOnGas(int burnerNum){
        System.out.println(" Gas on for burner: " + burnerNum);
    }
    public void turnOffGas(int burnerNum){
        System.out.println(" Gas off for burner: " + burnerNum);
    }
    public void turnOnElectricity(int burnerNum){
        System.out.println(" Electricity on for burner: " + burnerNum);
    }
    public void turnOffElectricity(int burnerNum){
        System.out.println(" Electricity off for burner: " + burnerNum);
    }
    public void igniteElectrically(int burnerNum){
        System.out.println(" Igniting electrically burner: " + burnerNum);
    }
    public void turnOnExhaust(){
        System.out.println(" Exhaust on");
    }
    public void turnOffExhaust(){
        System.out.println(" Exhaust off");
    }
    public void cleanCooktop(){
        System.out.println(" Autocleaning cooktop");
    }
}
```

Figure 3. Cooktop Implementation - Methods

```java
public class App {
    public static void main(String[] args) throws Exception {
        //Types of stoves: GasStove DowndraftStove ElectricStove ModularStove SmartStove
        Cooktop gasCooktop = new Cooktop("GE","GasStove");
        Cooktop electricCoilCooktop = new Cooktop("KitchenAid", "ElectricStove");
        Cooktop hybridCooktop = new Cooktop("Miele","DowndraftStove", "ModularStove");
        Cooktop inductionCooktop = new Cooktop("Bosch","ElectricStove", "SmartStove");

        java.util.ArrayList<Cooktop> stoves = new java.util.ArrayList<>();
        stoves.add(gasCooktop);
        stoves.add(electricCoilCooktop);
        stoves.add(hybridCooktop);
        stoves.add(inductionCooktop);

        for(Stove s: stoves){
            System.out.println("--------------------------");
            s.turnOnBurner(1);
            System.out.println("----");
            s.turnOffBurner(1);
        }
    }
}
```

Figure 4. Runner Implementation showing Concrete Cooktop Objects and Method Invocation

Note: The code for these classes is available on the course page.

(a) Describe three problem(s) with the approach shown in Figures 1-3, to specify the objects' behaviours in the scenario. Support your answer with annotated code snippets. (6 marks)

(b) Identify three SOLID object-oriented design principles violated in Figures 1-3. (3 marks)

(c) Suggest how the implementation in Figures 1-3 can be refactored to adhere to the principles identified in part (b). (6 marks)

(d) Draw a UML class diagram to model a refactored solution for the domain in the scenario based on your answers in part (c). You may use the method signatures in Figure 1 and rename classes/ interfaces as necessary. Use Figure 4 as a guide for naming the concrete cooktop classes.

(20 marks)

(e) Produce refactored code that matches your solution in (d) but which still produces the output by the Runner implementation in Figure 4. Format and insert your answers neatly as shown in Figures 1-4 using the least space for easy readability. (15 marks)

(f) Suppose a user interacts with the GE GasCooktop object and the Miele HybridCooktop object in Figure 4. Draw a sequence diagram to illustrate the interactions when the user does the following in order:

- turns on burner 3 in the GE GasCooktop object
- turns on burner 1 in the Miele HybridCooktop object
- turns off burner 3 in the GE GasCooktop object
- turns off burner 1 in the Miele HybridCooktop object

(15 marks)

END OF EXAMINATION PAPER