SDA

LAB MIDTERM

USMAN ALI

FA21-BSE-159

HOSTEL SYSTEM

Hostel management system using pipe and filter architecture,

- eligibility filter.
- services filter.
- :payment filter.

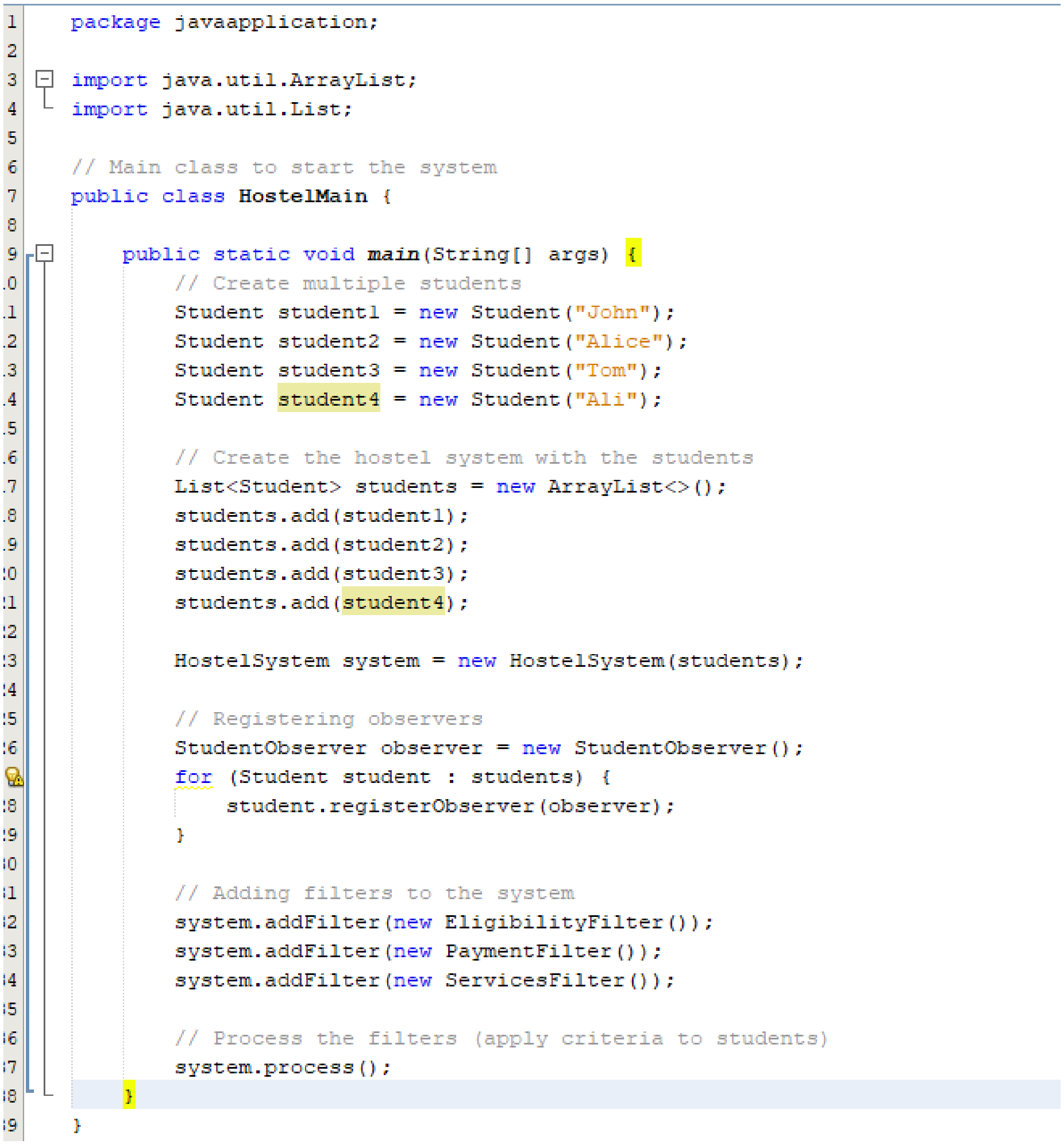
Plus observer pattern to send notification to all students required.

3 Layer architecture.

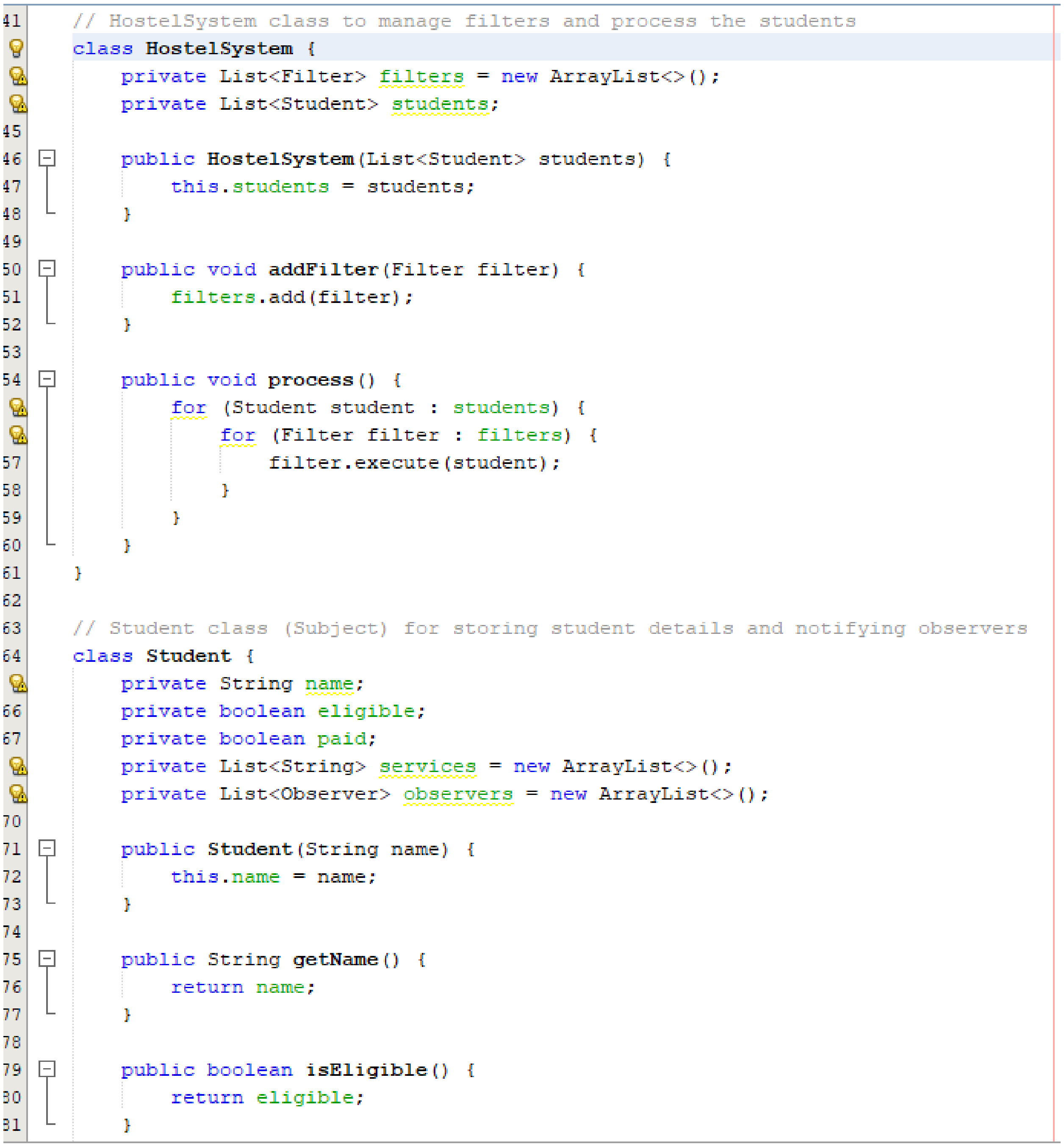
DATE: 19/11/2024

SUBMITTED TO : SIR MUKHTIYAR ZAMIN

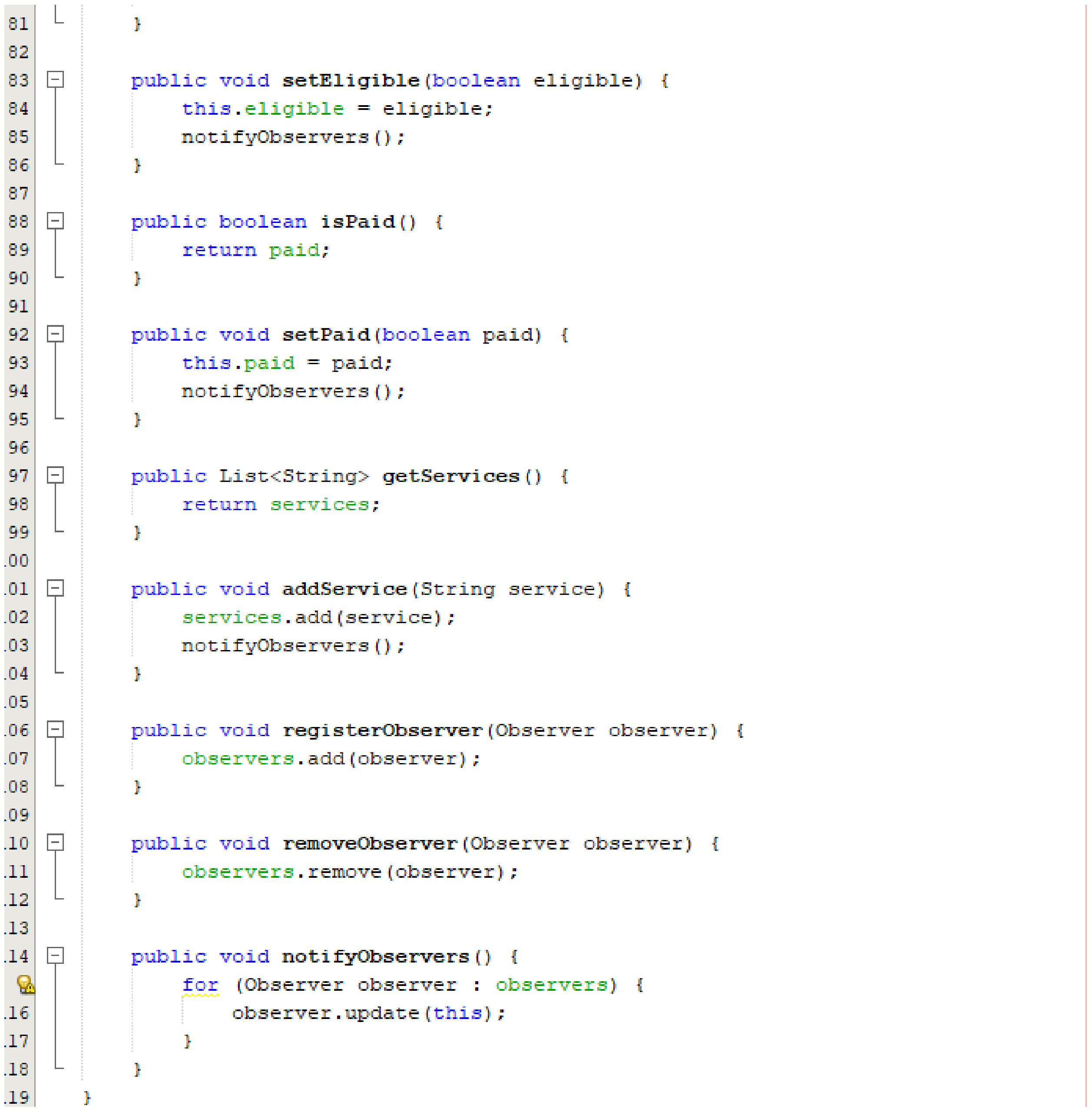
CODE:



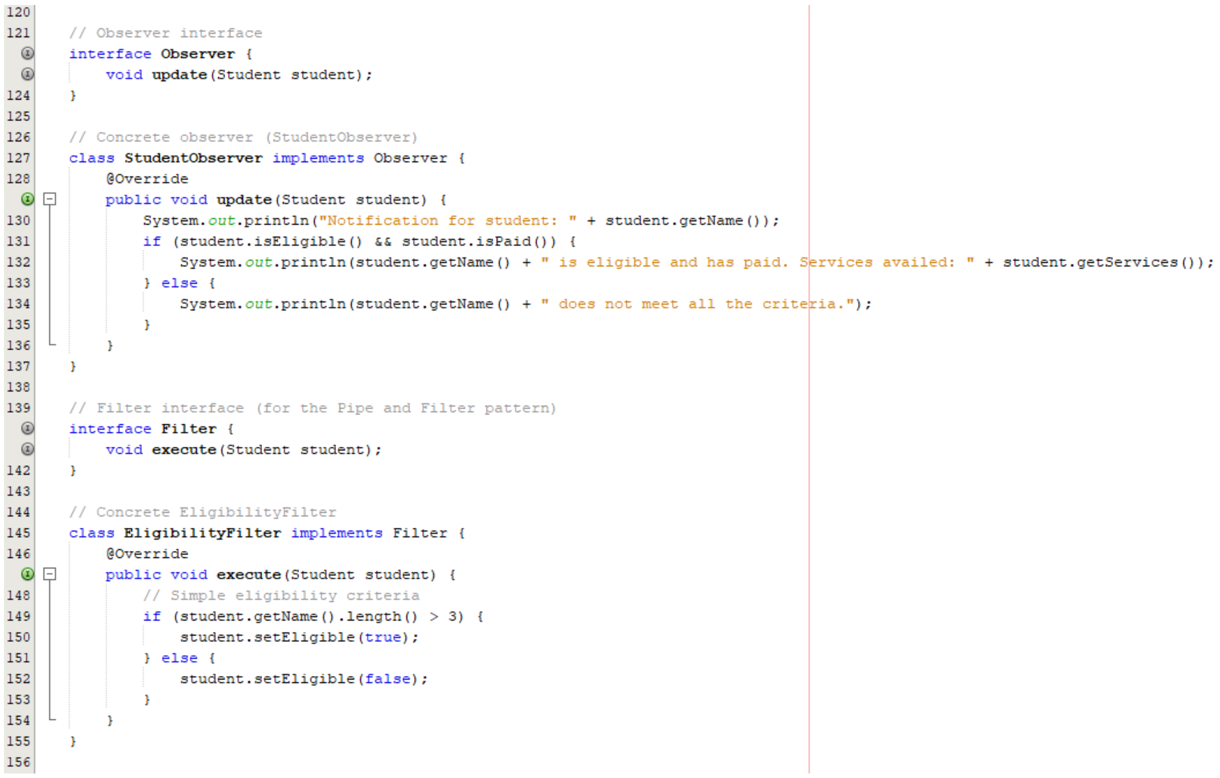
```java
package javaapplication;

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

// Main class to start the system
public class HostelMain {

    public static void main(String[] args) {
        // Create multiple students
        Student student1 = new Student("John");
        Student student2 = new Student("Alice");
        Student student3 = new Student("Tom");
        Student student4 = new Student("Ali");

        // Create the hostel system with the students
        List<Student> students = new ArrayList<>();
        students.add(student1);
        students.add(student2);
        students.add(student3);
        students.add(student4);

        HostelSystem system = new HostelSystem(students);

        // Registering observers
        StudentObserver observer = new StudentObserver();
        for (Student student : students) {
            student.registerObserver(observer);
        }

        // Adding filters to the system
        system.addFilter(new EligibilityFilter());
        system.addFilter(new PaymentFilter());
        system.addFilter(new ServicesFilter());

        // Process the filters (apply criteria to students)
        system.process();
    }
}
```



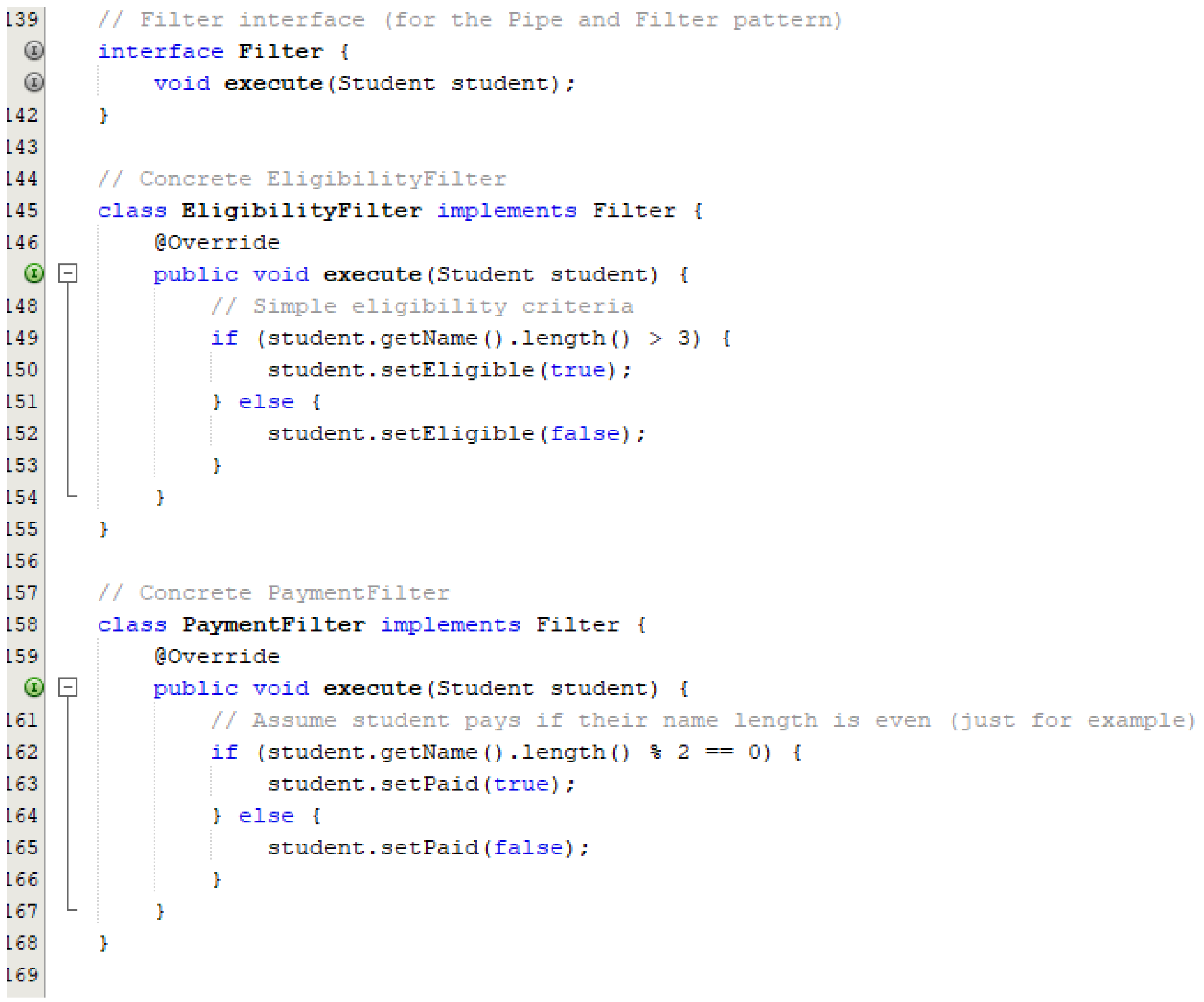
```
// HostelSystem class to manage filters and process the students
class HostelSystem {
    private List<Filter> filters = new ArrayList<>();
    private List<Student> students;

    public HostelSystem(List<Student> students) {
        this.students = students;
    }

    public void addFilter(Filter filter) {
        filters.add(filter);
    }

    public void process() {
        for (Student student : students) {
            for (Filter filter : filters) {
                filter.execute(student);
            }
        }
    }
}

// Student class (Subject) for storing student details and notifying observers
class Student {
    private String name;
    private boolean eligible;
    private boolean paid;
    private List<String> services = new ArrayList<>();
    private List<Observer> observers = new ArrayList<>();

    public Student(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public boolean isEligible() {
        return eligible;
    }
```



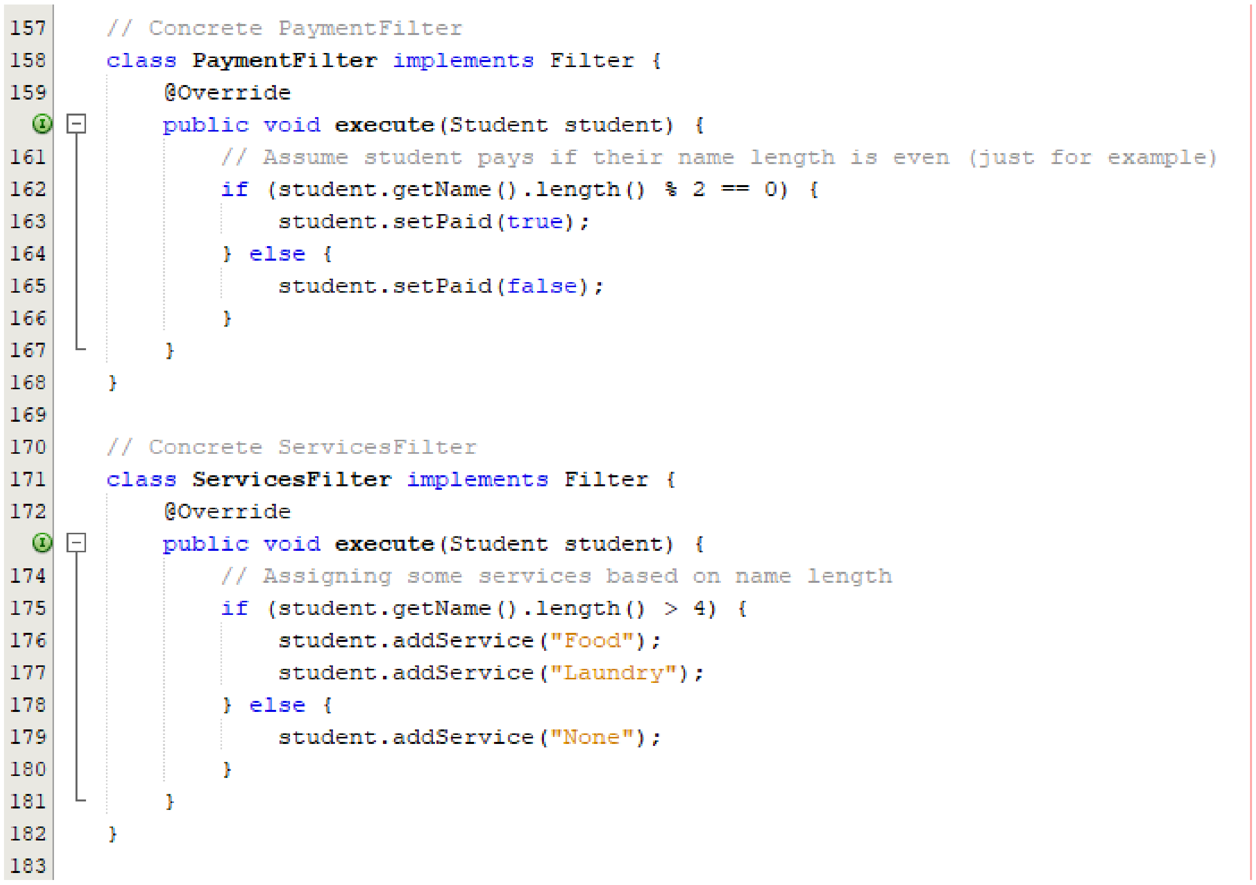
```java
    }

    public void setEligible(boolean eligible) {
        this.eligible = eligible;
        notifyObservers();
    }

    public boolean isPaid() {
        return paid;
    }

    public void setPaid(boolean paid) {
        this.paid = paid;
        notifyObservers();
    }

    public List<String> getServices() {
        return services;
    }

    public void addService(String service) {
        services.add(service);
        notifyObservers();
    }

    public void registerObserver(Observer observer) {
        observers.add(observer);
    }

    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(this);
        }
    }
}
```



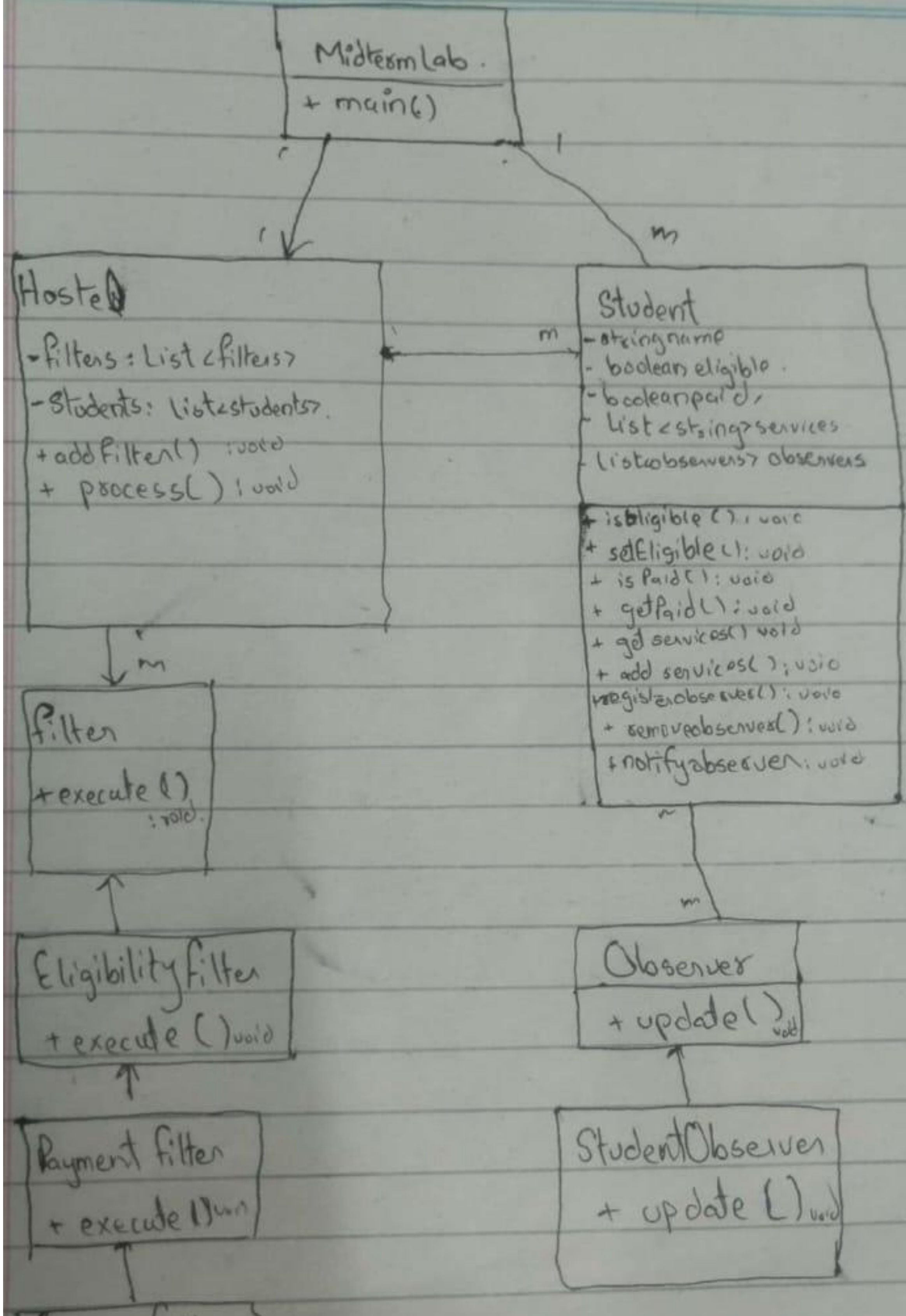
```java

// Observer interface
interface Observer {
    void update(Student student);
}

// Concrete observer (StudentObserver)
class StudentObserver implements Observer {
    @Override
    public void update(Student student) {
        System.out.println("Notification for student: " + student.getName());
        if (student.isEligible() && student.isPaid()) {
            System.out.println(student.getName() + " is eligible and has paid. Services availed: " + student.getServices());
        } else {
            System.out.println(student.getName() + " does not meet all the criteria.");
        }
    }
}

// Filter interface (for the Pipe and Filter pattern)
interface Filter {
    void execute(Student student);
}

// Concrete EligibilityFilter
class EligibilityFilter implements Filter {
    @Override
    public void execute(Student student) {
        // Simple eligibility criteria
        if (student.getName().length() > 3) {
            student.setEligible(true);
        } else {
            student.setEligible(false);
        }
    }
}

```



```java
// Filter interface (for the Pipe and Filter pattern)
interface Filter {
    void execute(Student student);
}

// Concrete EligibilityFilter
class EligibilityFilter implements Filter {
    @Override
    public void execute(Student student) {
        // Simple eligibility criteria
        if (student.getName().length() > 3) {
            student.setEligible(true);
        } else {
            student.setEligible(false);
        }
    }
}

// Concrete PaymentFilter
class PaymentFilter implements Filter {
    @Override
    public void execute(Student student) {
        // Assume student pays if their name length is even (just for example)
        if (student.getName().length() % 2 == 0) {
            student.setPaid(true);
        } else {
            student.setPaid(false);
        }
    }
}
```


```java
// Concrete PaymentFilter
class PaymentFilter implements Filter {
    @Override
    public void execute(Student student) {
        // Assume student pays if their name length is even (just for example)
        if (student.getName().length() % 2 == 0) {
            student.setPaid(true);
        } else {
            student.setPaid(false);
        }
    }
}

// Concrete ServicesFilter
class ServicesFilter implements Filter {
    @Override
    public void execute(Student student) {
        // Assigning some services based on name length
        if (student.getName().length() > 4) {
            student.addService("Food");
            student.addService("Laundry");
        } else {
            student.addService("None");
        }
    }
}
```

CLASS DIAGRAM:



MidtermLab.
+ main()
1
1
m
Hostel
-filters : List <filters>
-Students: list<students>.
+ addFilter() :void
+ process() : void
m
Student
- stringname
- boolean eligible.
- booleanpaid,
- List<string>services
- list<observers> observers
+ isEligible () : void
+ setEligible(): void
+ isPaid() : void
+ getPaid() : void
+ get services() void
+ add services( ); void
+registerobserver() : void
+ removeobserver() : void
+ notifyobserver: void
m
filter
+ execute () : void.
Eligibility filter
+ execute ()void
Payment filter
+ execute ()void
m
Observer
+ update()void
StudentObserver
+ update ()void