Question 1 [40 points] {55 minutes}

Suppose you have to design a seat reservation system for a movie theatre with the following requirements:

The seat reservation system covers all cinemas in the country. Cinemas have one or more rooms, and each room contains a set of seats. People can use the system to find particular movie-sessions(shows) by various search criteria like city, time, cinema, movie title, movie category and so on. Movies can be categorized into different categories like for example:

New releases -> Family -> Animation -> …
Best sellers -> Drama -> …

When the user has entered search criteria he or she will be presented with a list of shows that fulfill the criteria. Clicking a show gives a graphical presentation of the room of the show, and which seats can be selected, a total prize displayed and a reservation-commit button. When the user makes a reservation, the reservation number is displayed at the screen. The user can then bring along this reservation number to the cinema. This system does not print tickets or handle payments. You can only use it to find movie shows and reserve seats for this show.

The a seat reservation system should support the following additional requirements:

- Any number of movie categories and sub-categories should be possible.

- You can reserve multiple seats for a particular movie in one reservation.
- For every movie we can see the following information: title, runtime, genre, language, year, country
- You should be able to search movies by location, movie genre and movie title.
- You should be able to browse through movie categories.

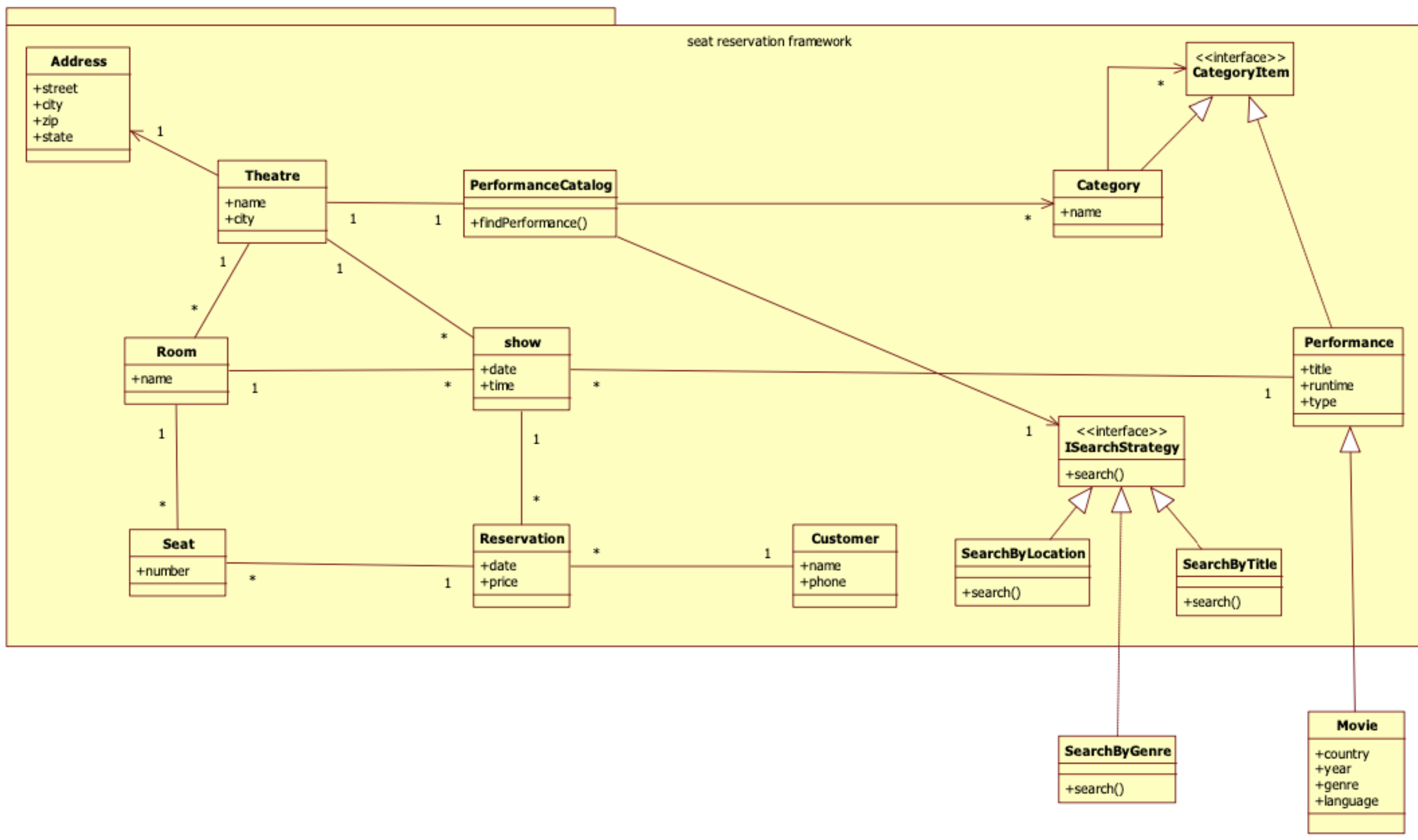
Now another customer wants you to design a seat reservation system for company which owns different theatres throughout the country and offer different shows like musicals, ballet performances, opera performances, etc.

People can use this seat reservation system to find particular performances by various search criteria like city, time, theatre, performance title, performance category and so on.

When the user has entered search criteria he or she will be presented with a list of performances that fulfill the criteria. Clicking a performance gives a graphical presentation of the room, and which seats can be selected, a total prize displayed and a reservation-commit button. When the user makes a reservation, the reservation number is displayed at the screen. The user can then bring along this reservation number to the theatre. This system does not print tickets or handle payments.

The a seat reservation system should support the following additional requirements:

- Any number of performance categories and sub-categories should be possible.
- You can reserve multiple seats for a particular performance in one reservation.
- For every performance we can see the following information: title, runtime and type of performance (musical, ballet, opera, etc.)
- You should be able to browse through performance categories.


seat reservation framework
Address
+street
+city
+zip
+state
Theatre
+name
+city
PerformanceCatalog
+findPerformance()
Category
+name
<<interface>>
CategoryItem
Room
+name
show
+date
+time
Performance
+title
+runtime
+type
<<interface>>
ISearchStrategy
+search()
Seat
+number
Reservation
+date
+price
Customer
+name
+phone
SearchByLocation
+search()
SearchByTitle
+search()
SearchByGenre
+search()
Movie
+country
+year
+genre
+language
1
*

Question 2 [25 points] {15 minutes}

Consider the following given application:

```
public class Person {
      private String name;

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

      public String getName() {
            return name;
      }

      public void setName(String name) {
            this.name = name;
      }

}


public interface IPersonDAO {
    Person getPerson(String ssn);
}

public class PersonDAO implements IPersonDAO{

      public Person getPerson(String ssn) {
            if (ssn.equals("324-554-5678"))
               return new Person("Frank Brown");
            if (ssn.equals("233-332-4444"))
               return new Person("John Doe");
            return null;
      }
}


public interface IPersonService {
    public Person findPerson(String ssn);
}

public class PersonService implements IPersonService{
    private IPersonDAO personDao = new PersonDAO();

      public Person findPerson(String ssn) {
            return personDao.getPerson(ssn);
      }

      public void setPersonDao(PersonDAO personDao) {
            this.personDao = personDao;
      }

}
```

```
public class Application {
      public static void main(String[] args) {




            IPersonService personService = new PersonService();
            Person p = personService.findPerson("324-554-5678");
            System.out.println(p.getName());
            p = personService.findPerson("233-332-4444");
            System.out.println(p.getName());
      }

}



public class ServiceCounter {
      private int count=0;



      public void log(JoinPoint joinpoint) {
            count=count+1;
            System.out.println("method= "+
   joinpoint.getSignature().getName() + " is called " + count + " times" );
      }

}
```

springconfig.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
```

The given application is not a Spring application.
Running this application gives the following output:

```
Frank Brown
John Doe
```

The disadvantage of this simple application is that:

1. If we want to use another DAO class, we have to modify the code.
2. In the given code, the log() method of the ServiceCounter class is not called. We would like to call this method every time we call the findPerson() method on the PersonService class.

Modify the given code (**do NOT write all code again on your paper, this takes too much time**) such that:

1. The application becomes a Spring application.
2. Use Spring dependency injection to inject the PersonDAO into the PersonService. It should be possible to inject another DAO class into the PersonService without modifying any Java code.
3. Use Spring AOP for implementing the functionality that the log() method of the ServiceCounter class is called every time we call the findPerson() method on the PersonService class.

Running the Spring application should give the following output:

```
method= findPerson is called 1 times
Frank Brown
method= findPerson is called 2 times
John Doe
```

RESULT

```
public class Person {
       private String name;

       public Person(String name) {
              super();
              this.name = name;
       }

       public String getName() {
              return name;
       }

       public void setName(String name) {
              this.name = name;
       }

}


public interface IPersonDAO {
    Person getPerson(String ssn);
}

public class PersonDAO implements IPersonDAO{

       public Person getPerson(String ssn) {
              if (ssn.equals("324-554-5678"))
                 return new Person("Frank Brown");
              if (ssn.equals("233-332-4444"))
                        return new Person("John Doe");
              return null;
       }
}
```

```
public interface IPersonService {
    public Person findPerson(String ssn);
}

public class PersonService implements IPersonService{
    private PersonDAO personDao;

      public Person findPerson(String ssn) {
            return personDao.getPerson(ssn);
      }

      public void setPersonDao(PersonDAO personDao) {
            this.personDao = personDao;
      }

}


public class Application {
      public static void main(String[] args) {
          ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
          IPersonService personService = (IPersonService)context.getBean("personService");
          Person p = personService.findPerson("324-554-5678");
          System.out.println(p.getName());
          p = personService.findPerson("233-332-4444");
          System.out.println(p.getName());
      }

}

@Aspect
public class ServiceCounter {
      private int count=0;

      @After("execution(* PersonService.findPerson(..))")
      public void log(JoinPoint joinpoint) {
            count=count+1;
            System.out.println("method= "+ joinpoint.getSignature().getName() + " is called " + count + " times" );
      }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans   http://www.springfr
      http://www.springframework.org/schema/aop http://www.springframew

     <aop:aspectj-autoproxy/>
     <bean id="personService" class="PersonService">
        <property name="personDao" ref="personDao" />
    </bean>
    <bean id="personDao" class="PersonDAO"/>
    <bean id="serviceCounter" class="ServiceCounter"/>
</beans>
```

Question 3 [30 points] {40 minutes}

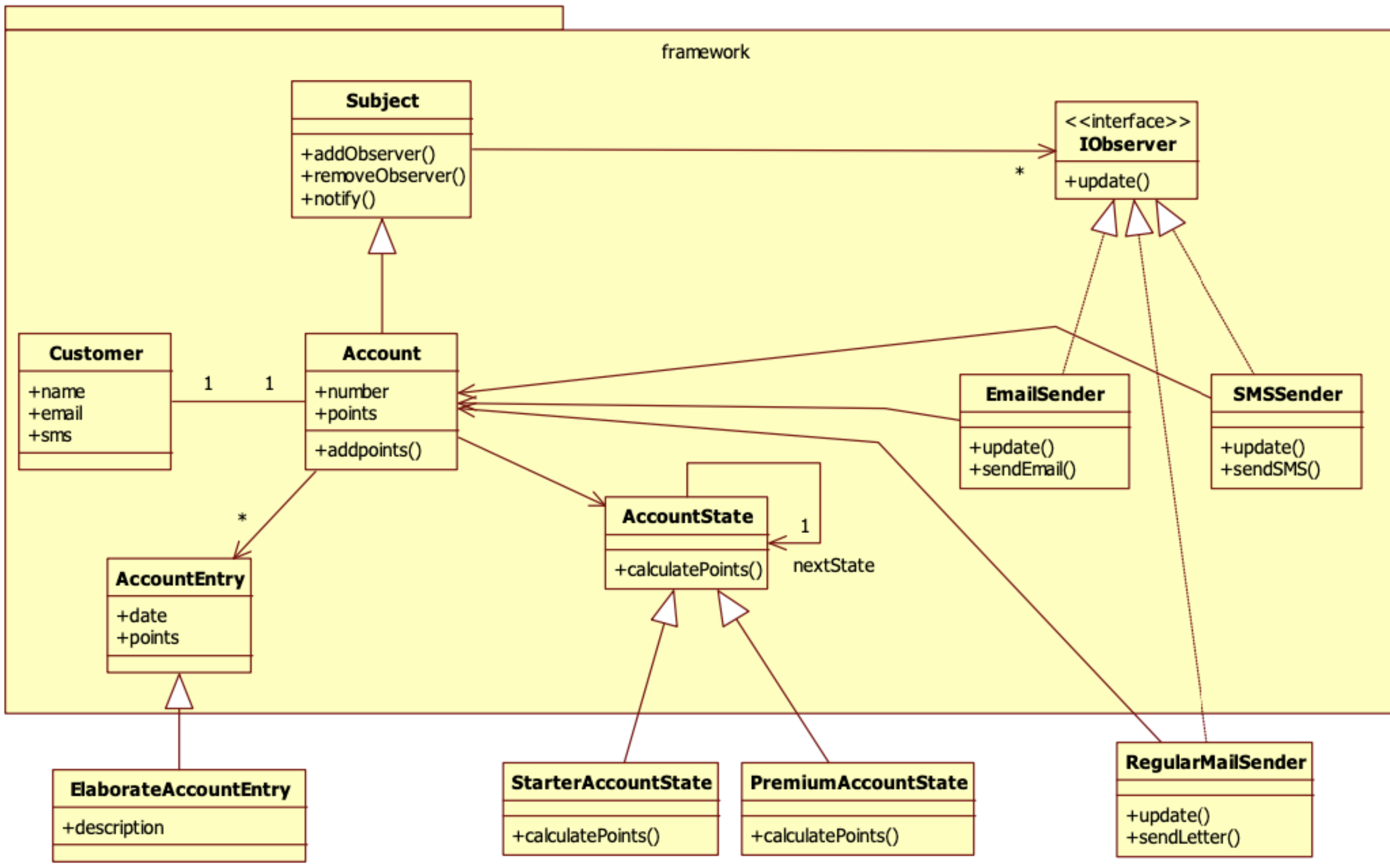
We want to design a points award framework that allows us to write applications that records the number of points in a points award program like frequent flyer miles programs or hotel reward programs. The framework should support the following requirements:

- Members have a certain type of Account, like bronze, silver and gold, and depending on the account type you get a certain number of bonus points
- The possible accounts are application specific
- When your account reaches a certain number of points, you will be upgraded to a higher account type where you receive more points.
- Customers can subscribe themselves to notifications on certain events, for example when their account state changes (from silver to gold for example) or when new points are added to their account. Customers can decide themselves how they want to be notified, for example by email or by SMS. The framework should support both notifications by email and notifications by SMS.
- It should be easy to add more notification options, for example by regular mail.
- The history of added points should be available. We should be able to see how many points are added at what date.

Draw the **class diagram** of a particular points award application using the points award framework. The points award application should support the following requirements:

- The award program supports 2 types of accounts: Starter accounts and Premium accounts. Premium accounts receive 2 times more points as Starter accounts.
- If new points are added, customers can be notified by email, SMS or receive a letter by regular mail.
- The points awards program allows us to see the history overview that shows the date, the number of points that are added and a description that describes the reason why we get these points.

In the class diagram, show clearly which classes are within the framework, and which classes are outside the framework (based on the requirements for the framework, and the framework best practices we studied in this course) . **Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc) to communicate the important parts of your design. You only need to show the domain model of the point awards framework and application, you do not need to worry about GUI classes, service classes, database classes, etc.**


framework
Subject
+addObserver()
+removeObserver()
+notify()
<<interface>>
IObserver
+update()
*
Customer
+name
+email
+sms
1
1
Account
+number
+points
+addpoints()
EmailSender
+update()
+sendEmail()
SMSSender
+update()
+sendSMS()
*
AccountEntry
+date
+points
AccountState
+calculatePoints()
1
nextState
ElaborateAccountEntry
+description
StarterAccountState
+calculatePoints()
PremiumAccountState
+calculatePoints()
RegularMailSender
+update()
+sendLetter()