[20 minutes]

Consider the following code:

```
public class Cake {
   private double sugar;   //cup
   private double butter;  //cup
   private int eggs;
   private double flour;   //cup

   public Cake(double sugar, double butter, int eggs, double flour) {
      this.sugar = sugar;
      this.butter = butter;
      this.eggs = eggs;
      this.flour = flour;
   }
   public double getSugar() {
      return sugar;
   }
   public void setSugar(double sugar) {
      this.sugar = sugar;
   }
   public double getButter() {
      return butter;
   }
   public void setButter(double butter) {
      this.butter = butter;
   }
   public int getEggs() {
      return eggs;
   }
   public void setEggs(int eggs) {
      this.eggs = eggs;
   }
   public double getFlour() {
      return flour;
   }
```

```
    public void setFlour(double flour) {
        this.flour = flour;
    }
}
```

The problem with this code is that if you instantiate a cake object, the code looks like this:

```
Cake cake = new Cake(1.25, 1, 3, 3);
```

From this code it is not clear what the values of the arguments in the constuctor mean and it is easy to make mistakes with them.
Another problem is that this Cake class is mutable.

Rewrite this Cake class so that
1. The package class is **immutable**
2. We can create a package class with **self-explaining code so that it is clear from the**
code what the numbers 1.25, 1, 3, 3 mean.

Write both the **code of the Cake class**, and the **code that creates a cake** with sugar=1.25 cups,
butter=1 cup, eggs=3, flour=3 cups.

```java
public class Cake {
  private double sugar;   //cup
  private double butter;  //cup
  private int eggs;
  private double flour;   //cup

  public static class Builder {
     private double sugar;   //cup
     private double butter;  //cup
     private int eggs;
     private double flour;   //cup

  public Builder withSugarInCups(double sugar) {
     this.sugar = sugar;
     return this;
  }
  public Builder withButterInCups(double butter) {
     this.butter = butter;
     return this;
  }
  public Builder withNumberOfEggs(int eggs) {
     this.eggs = eggs;
      return this;
  }
  public Builder withFlourInCups(double flour) {
     this.flour = flour;
     return this;
  }

  public Cake build() {
     return new Cake(this);
  }
}

  public String toString() {
     return "Cake [sugar=" + sugar + ", butter=" + butter
+ ", eggs=" + eggs + ", flour=" + flour + "]";
     }

  public Cake(Builder builder) {
     this.sugar = builder.sugar;
     this.butter = builder.butter;
```

```java
        this.eggs = builder.eggs;
        this.flour = builder.flour;
    }

    public double getSugar() {
        return sugar;
    }

    public double getButter() {
        return butter;
    }

    public int getEggs() {
        return eggs;
    }

    public double getFlour() {
        return flour;
    }
}

public class ApplicationCake {

        public static void main(String[] args) {
                Cake cake = new Cake.Builder()
                                .withFlourInCups(3.5)
                                .withButterInCups(1.0)
                                .withNumberOfEggs(3)
                                .withSugarInCups(1.5)
                                .build();
                System.out.println(cake);
        }
}
```

[10 minutes]
Explain clearly the difference between the factory method pattern and the abstract factory pattern

The factory method pattern is used to create one object and the abstract factory pattern is a factory of factories that is used to create a family of related objects

[30 minutes]

Suppose we want to write our own Spring like framework so that the following application will work using the framework:

```
public class Application implements Runnable{
   @Inject
   BankService bankService;

   public static void main(String[] args) {
      FWApplication.run(Application.class);
   }

   @Override
   public void run() {
      bankService.deposit();
   }
}

public interface BankService {
   public void deposit() ;
}

@Service
public class BankServiceImpl implements BankService{
   @Inject
   private EmailService emailService;

   public void setEmailService(EmailService emailService) {
      this.emailService = emailService;
   }


   public void deposit() {
      emailService.send("deposit");

   }

}

public interface EmailService {
   void send(String content);
}

@Service
public class EmailServiceImpl implements EmailService{
   public void send(String content) {
      System.out.println("sending email: "+content);
   }
}
```

The framework will instantiate all classes annotated with **@Service**, and the framework supports **dependency injection**.

The code of the framework is as follows:

```
public class FWContext {

   private static List<Object> objectMap = new ArrayList<>();

   public FWContext() {
      try {
         // find and instantiate all classes annotated with the @Service
annotation
         Reflections reflections = new Reflections("");
         Set<Class<?>> types =
reflections.getTypesAnnotatedWith(Service.class);
         for (Class<?> implementationClass : types) {
            objectMap.add((Object) implementationClass.newInstance());
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      performDI();
   }

   private void performDI() {
      try {
         for (Object theTestClass : objectMap) {
             // find annotated fields
             for (Field field : theTestClass.getClass().getDeclaredFields()) {
                  if (field.isAnnotationPresent(Inject.class)) {
                     // get the type of the field
                     Class<?> theFieldType =field.getType();
                     //get the object instance of this type
                     Object instance = getBeanOftype(theFieldType);
                     //do the injection
                     field.setAccessible(true);
                     field.set(theTestClass, instance);
                  }
```

```
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public Object getBeanOftype(Class interfaceClass) {
        Object service = null;
        try {
            for (Object theTestClass : objectMap) {
                Class<?>[] interfaces = theTestClass.getClass().getInterfaces();

                for (Class<?> theInterface : interfaces) {
                    if
(theInterface.getName().contentEquals(interfaceClass.getName()))
                        service = theTestClass;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return service;
    }

}

@Retention(RUNTIME)
@Target(FIELD)
public @interface Inject {
}

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Service {

}

public class FWApplication {

    public static void run(Class applicationClass) {
        // create the context
        FWContext fWContext = new FWContext();
```

```
      try {
        // you have to write the missing code
    // create instance of the application class
    Object applicationObject = (Object) applicationClass.newInstance();
    // find annotated fields
    for (Field field :
    applicationObject.getClass().getDeclaredFields()) {
      if (field.isAnnotationPresent(Inject.class)) {
           // get the type of the field
           Class<?> theFieldType = field.getType();
           // get the object instance of this type
           Object instance = fWContext.getBeanOftype(theFieldType);
           // do the injection
           field.setAccessible(true);
           field.set(applicationObject, instance);
      }
    }
    //call the run() method
    if (applicationObject instanceof Runnable)
      ((Runnable)applicationObject).run();


      } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }
}
```

Write the missing code in the run() method of the FWApplication class.

You can write pseudo code as long as all steps of what the code should do is clear.

[10 minutes]

Explain clearly why all spring beans should have an interface if we want to make use of AOP in Spring.

Because Spring implements AOP using a proxy. This means the client that would normally talk to the target object will now talk to the proxy object in front of the target object. So in the code of the client we need to use the interface of the target class, otherwise we get an exception.

[10 minutes]

Describe how **Aspect Oriented Programming** relates to one or more of the SCI principles you know. Your answer should be about half a page, but should not exceed one page (handwritten). The number of points you get for this question depends on how well you explain the relationship between **Aspect Oriented Programming** and the principles of SCI.

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.

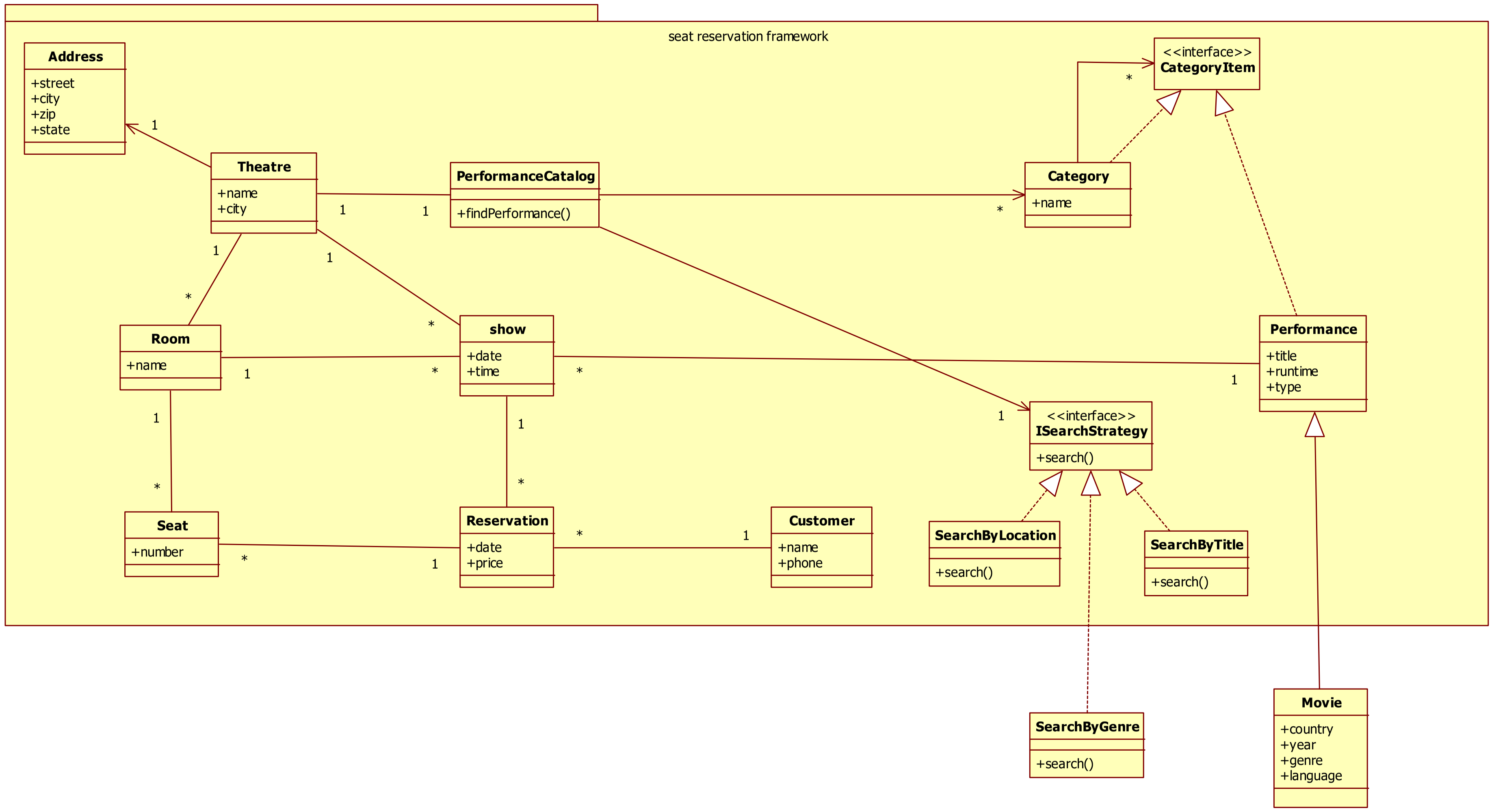
Because this is our second seat reservation system, we decide to make a generic seat reservation framework.

The seat reservation framework should support the following additional requirements:

- The framework supports different searching algorithms like searching by location and searching by performance title. It should be easy to add more searching algorithms.

Draw the **UML class diagram** of the seat reservation system for a movie cinema using the seat reservation framework.

So this class diagram should show the **design of the framework**, and the design of the **seat reservation application using the framework**. 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 seat reservation framework and application, you do not need to worry about GUI classes, service classes, database classes, etc.**


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
*