Final exam

[15 minutes]

a. Explain the 3 main advantages of a **static factory method** compared to a constructor.

b. Suppose I have a **User** class with the attributes **name**, **phone** and **email**. Write the class **User** in Java. Write a **static factory method** in the User class that returns a **singleton** User class that is both **thread save** and **reflection save**.

a.

Factory methods can have meaningful names

Factory methods can return anything

We can have multiple factory methods with the same arguments

b.

```
public class User {
private static User user;
private String name;
private String phone;
private String email;

private User(String name, String phone, String email) {
if(user != null) // to make this reflection safe
return new RuntimeException("instance already exists");
this.name= name;
this.email=email;
this.phone= phone;
}
public static User getUser(String name, String phone, String email) {
if(user == null) {
   synchronize(User.class) {

      if(user == null)

        user= new User(name, phone, email);

   }
}

return user;
}

}
```

[20 minutes]

Suppose I have a Box class with the integer attributes length, width and height.

When we instantiate the Box object with length=10, width=12 and height=20 with the following Java code:

Box box = new Box(10, 12, 20);

Is not very clear what the number 10, 12 and 20 mean. It is easy to make mistake with this code

a. Write in **Java all code of this Box class** with the following requirements:

1. The Box class should be **immutable**
2. When we instantiate the Box object with length=10, width=12 and height=20, it should be very clear from the client code what the values 10, 12 and 20 mean so that we don't make mistakes.

b. Write the client code that creates a Box object with length=10, width=12 and height=20

a.

```
public class Box {
private double length;
private double width;
private double height;

private Box(Builder builder) {
this.length= builder.length;
this.width= builder.width;
this.height= builder.height;
}
public static class Builder {
private double length;
private double height;
private double width;

public Builder withLength(double length){
this.length= length;
return this;
}
public Builder withHeight(double height) {
this.height= height;
return this;
}
public Builder withWidth(double width) {
this.width= with;
return this;
}
public Box build() {
return new Box(this);
}
}
}
```

b.

```
// client code
Box box= new Box.Builder().withLength(10).withWidth(12).withHeight(20).build();
```

[15 minutes]

Suppose we have a Spring Boot application with a BankAccountService class with the methods:

- deposit (int accountNumber, double amount);
- withdraw (int accountNumber, double amount);

Anytime the balance of a bankaccount changes, there are different classes that want to know about this:

- An AccountChangeNotifier will notify the account owner about the change
- An AccountBalanceLogger logs the account change in case something goes wrong we can see this log trace to find out what happened.

Write the Spring boot code of the **BankAccountService**, **AccountChangeNotifier** and **AccountBalanceLogger** classes i ncluding their methods so that the AccountChangeNotifier and AccountBalanceLogger get notified when a balance of a bankaccount changes.

Your solution should implement the following requirements:

1. This is a Spring Boot application
2. It should be easy to add other classes that also want to know about changes to an account
3. We want maximal loose coupling between all the involved classes.
4. The methods of the classes that want to know about changes to an account should be called asynchronous.

```
public class AccountChangeEvent {
public String operation;
public int accountNumber;
public double amount;
public AccountChangeEvent(String operation, int accountNumber, double amount, int
accountNumber) {
this.newAmount= amount;
this.accountNumber= accountNumber;
}
}

@Service
public class BankAccountService {
@Autowired
private ApplicationEventPublisher publisher;

public void deposit(int accountNumber, double amount) {
…
publisher.publishEvent(new AccountChangeEvent("deposit", account.getBalance(),
accountNumber );
}
public void withdraw(int accountNumber, double amount) {
…
publisher.publishEvent(new AccountChangeEvent("withdrawal", account.getBalance(),
accountNumber);
}
}

@EnableAsync
@Component
public class AccountChangeNotifier {
@Autowired
private Logger logger;

@EventListener
@Async
public void listenAccountChange(AccountChangeEvent event) {
  logger.log(…);
}
}
```

```
@EnableAsync
@Component
public class AccountBalanceLogger {
@Autowired
private Logger logger;

@EventListener
@Async
public void listenBalanceChange(AccountChangeEvent event) {
logger.log(… );
}
}
```

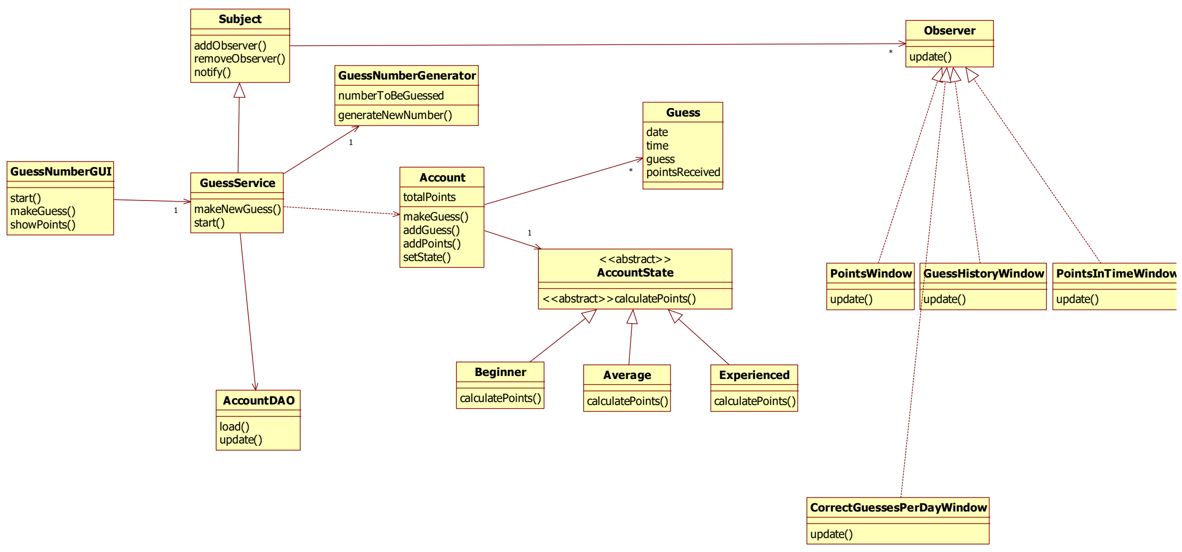
[60 minutes]

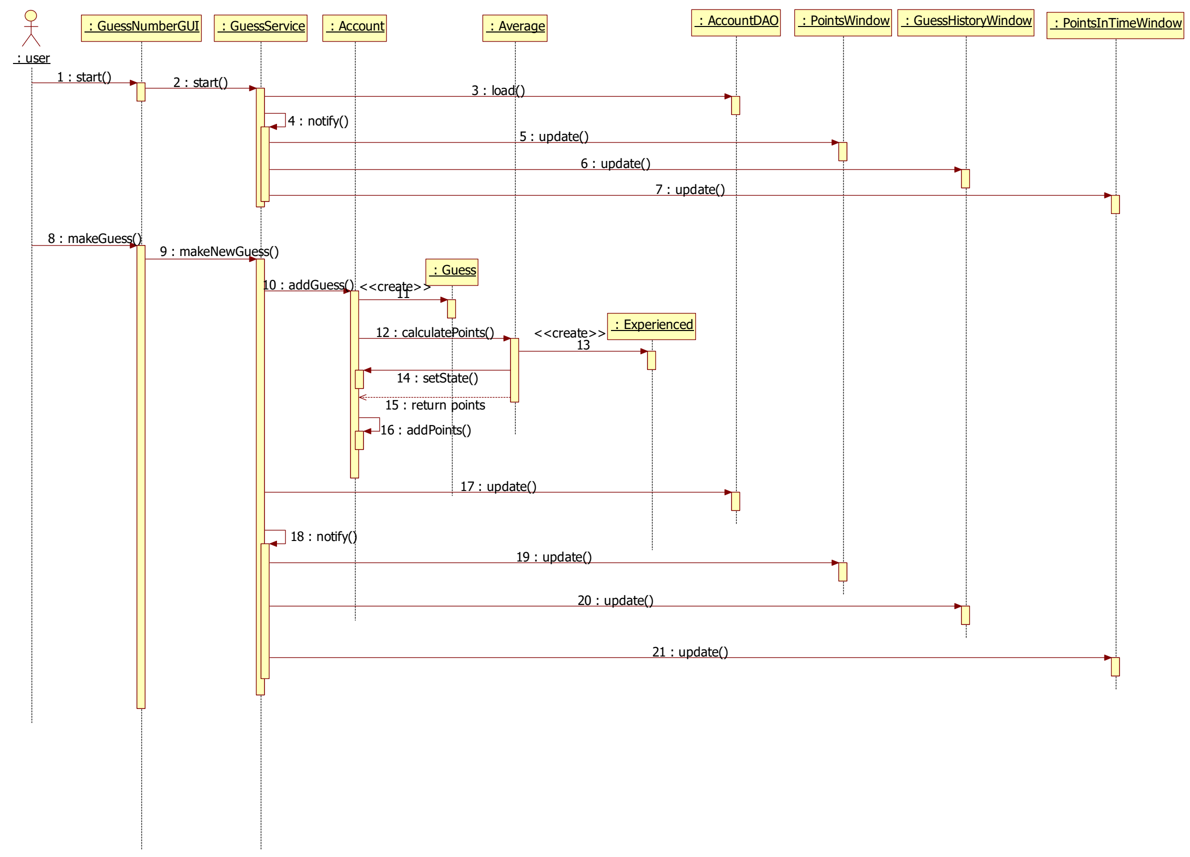
The question is given in the attached PDF.

Open the attached PDF in the browser. Draw your solution in StarUML and upload your solution as JPEG picture.

Make sure you upload the JPEG picture. In StarUML select File->Export Diagram As -> JPEG (or JPG). This question cannot be graded if you upload the StarUML mdj file.

Upload both the JPEG picture of your class diagram and your sequence diagram (you can upload multiple diagrams) . Make sure to draw all necessary information on both diagrams





: user
: GuessNumberGUI
: GuessService
: Account
: Average
: AccountDAO
: PointsWindow
: GuessHistoryWindow
: PointsInTimeWindow
1 : start()
2 : start()
3 : load()
4 : notify()
5 : update()
6 : update()
7 : update()
8 : makeGuess()
9 : makeNewGuess()
: Guess
10 : addGuess()
<<create>>
11
: Experienced
12 : calculatePoints()
<<create>>
13
14 : setState()
15 : return points
16 : addPoints()
17 : update()
18 : notify()
19 : update()
20 : update()
21 : update()