Practice midterm
Part 1

[10 minutes]

Explain the brewers cap theorem and explain what this means for distributed systems / databases.

From the 3 qualities Availability, Consistency and Partition tolerance we can have only 2 of these qualities for 100%.

This means we can have Availability and Consistency but then we are not easy to scale horizontally

If we have Availability and Partition tolerance we cannot have strict consistency, only eventual consistency

If we have Consistency and Partition tolerance we don't have availability.

[10 minutes]

When we implement an integration broker / ESB, we typically send messages to channels. There are 3 different types of messages we can send. One of them is an **event** message. Give the 3 types of messages we can send, and give an example of each.

Event message: lowBudged message
Command message: getLastTradePrice message
Document message: newPurchaseOrder message

[10 minutes]

In domain driven design we have 4 different types of classes. One type of class is an **Entity**. Give the name of the other 3 types of classes. For every type of class, give its important characteristics and give an example of each.

Entity: has identity, mutable Example: Order
Value object: has no identity, idempotent, self validating, …. Example: Address
Domain service: stateless. Example: shippingCostCalculator
Domain event: idempotent. Example: OrderReceivedEvent

[15 minutes]

Suppose we have the following PackageReceiver class:

```
@Service
public class PackageReceiver {

  public void receivePackage(Package thePackage) {

   ...

  }
}
```

We need to implement the functionality that whenever we call **receivePackage(Package thePackage)** on the PackageReceiver, the **notifyCustomer(Package thePackage)** method is called on the following CustomerNotifier class:

```
@Service
public class CustomerNotifier {

  public void notifyCustomer(Package thePackage) {
    System.out.println("send email to customer that we received the
          package with code "+thePackage.getPackagecode());
  }
}
```

The most important requirement is that the PackageReceiver and the CustomerNotifier class should be as loosely coupled as possible.

Write the code of the PackageReceiver and the CustomerNotifier so that we get the desired behavior.

```
@Service
public class PackageReceiver {
@Autowired
private ApplicationEventPublisher publisher;

  public void receivePackage(Package thePackage) {

    publisher.publishEvent(new NewPackageEvent(thePackage));

  }
}


@Service
public class CustomerNotifier {

@EventListener
public void onEvent(NewPackageEvent event) {
```

```
  notifyCustomer(event.getPackage());
}
  public void notifyCustomer(Package thePackage) {
    System.out.println("send email to customer that we received the
          package with code "+thePackage.getPackagecode());
  }
}
```

[10 minutes]

Suppose ApplicationA needs to call ApplicationB. One colleague tells you to use REST and another colleague tells you to use messaging. **Explain clearly** in what circumstances would you use **REST** and in what circumstances would you use **messaging**?

Rest

When the communication is synchronous
When I do not need a buffer
When ApplicationA and ApplicationB are not within the same organization

Messaging

When I need a buffer.
When the communication is asynchronous
When ApplicationA and ApplicationB are within the same organization
When I need broadcasting

[25 minutes]

Suppose you need to design a bank account application that allows users to perform the following actions:

- Deposit money to an account
- Withdraw money from an account
- View the details of an account
- Transfer money from one account to another account.

Our bank account application supports different currencies, so you can deposit in dollars, but also other currencies.
We need to implement the following business rules:

- You cannot withdraw more money than you have on your bank account
- Whenever the amount of a transaction is larger than $20.000, then the bank account application and maybe other applications need to know about this so they can check if this is not a fraudulent transaction

We need to design the bank account application using **Domain Driven Design**.

Give the **name of all domain classes** of the bank account application.
For every domain class specify the **type of the class (Entity,...).**
For every domain write its **attributes** and **method signatures** (name of the class, arguments and return value)

Write your solution like the following example:

Interface Counter
Methods:
void increment()
void decrement()
void setStartValue(int start)

class CounterImpl implements Counter :<<Entity>>
Attribute: value
Methods:
void increment()
void decrement()
void setStartValue(int start)

class Account:<<Entity>>
Attributes: accountNumber, balance
Methods:
void deposit()
void withdraw()

class Money:<<Value object>>
Attributes: amount, currency
Methods:
void add(Money money)
void subtract(Money money)

class AccountEntry:<<Value object>>
Attributes: amount, date, description

class TransferFundsService:<<Domain service>>
Methods:
void transferFund(Money money, Account fromAccount, Account toAccount, String description)

class LargeTransactionEvent:<<Domain event>>
Attributes: amount, date, description, fromAccountNumber, toAccountNumber, customerName

[10 minutes]

Describe how a **DDD aggregate** relates to one or more of the SCI principles you know. Your answer should be about 2 till 3 paragraphs. The number of points you get for this question depends on how well you explain the relationship between a **DDD aggregate** and the principles of SCI.

[10 minutes]

In Spring integration we have different types of channels. Give 4 different types of channels that are supported by Spring integration and explain how these channels are different in their behavior

- **Synchronous point-to-point channel**
- **Asynchronous point-to-point channel**
- **Synchronous publish-subscribe channel**
- **Asynchronous publish-subscribe channel**
- **Datatype channel**

[5 minutes]

Give 5 different examples of containers that go on a container diagram

- **Web servers**
- **Application servers**
- **ESBs**
- **Databases**
- **Other storage systems**
- **File systems**
- **Windows services**
- **Standalone/console applications**
- **Web browsers**

[10 minutes]

Give the advantages and disadvantages of the pipe-and-filter architectural style

Benefits

- **Filters are independent**
- **Filters are reusable**
- **Order of filters can change**
- **Easy to add new filters**
- **Filters can work in parallel**

Drawbacks

- **Works only for sequential processing**
- **Sharing state between filters is difficult**