Service Layer

CS544: Enterprise Architecture

Spring Services

- In this short module we will first discuss the purpose of the Service layer in Spring (enterprise) applications, after which we will also take a look at how to split the spring configuration file into multiple smaller special purpose configuration files.

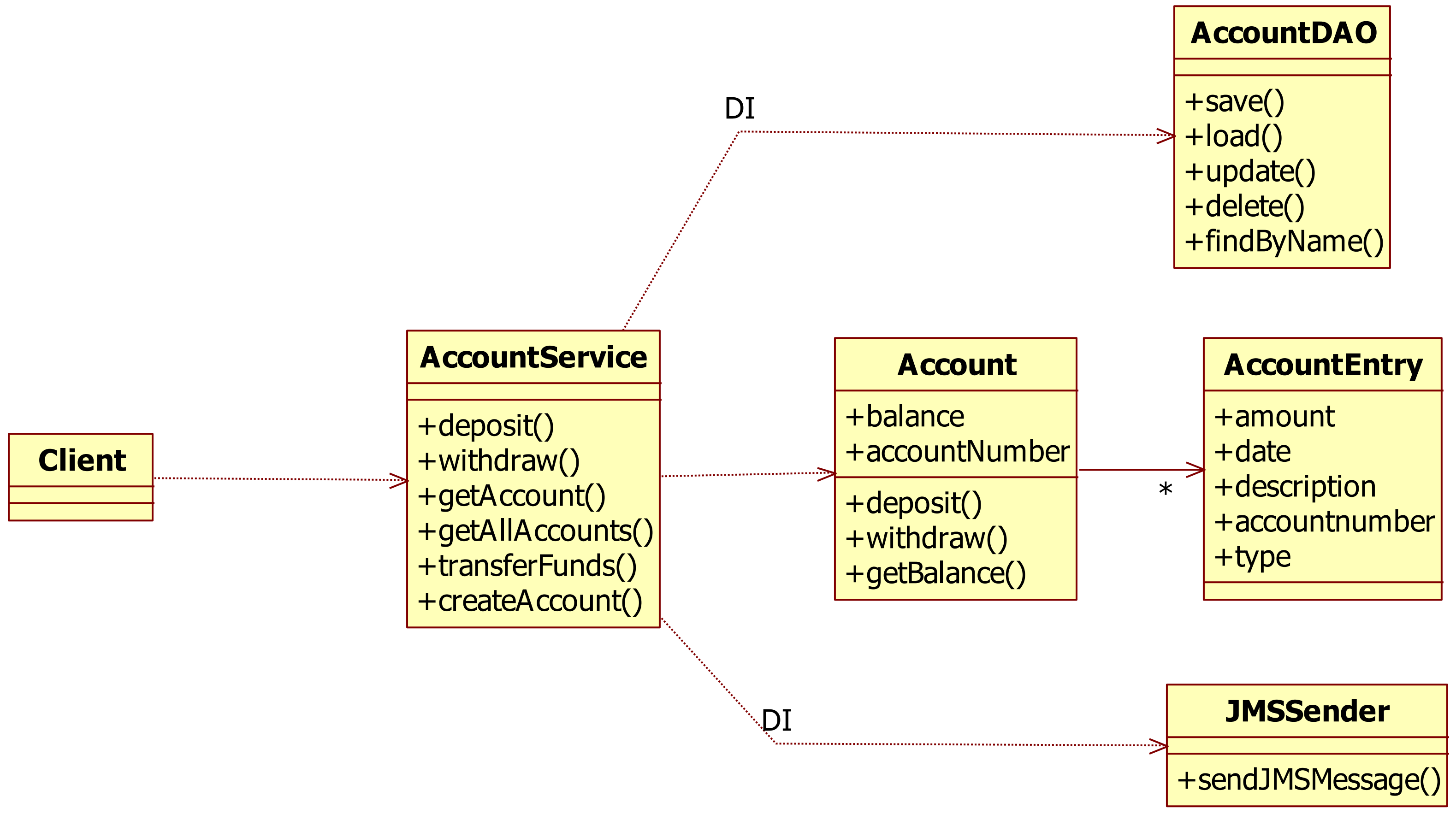
Service Layer:

THE SERVICE LAYER

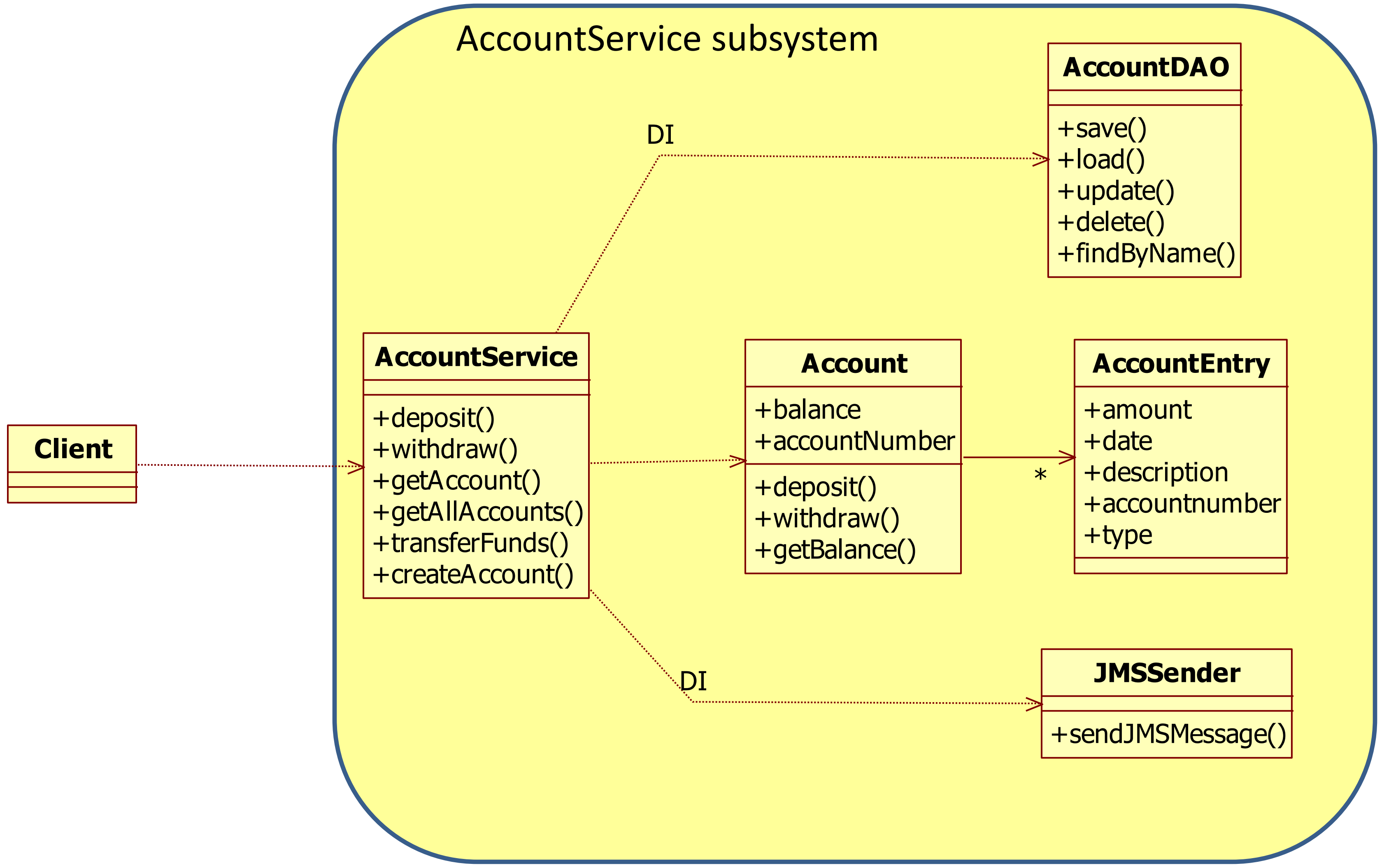
The Service Layer

- We discussed previously how the service layer can be seen as a technology independent controller
- Another point of view is that it is a place where multiple repository (DAO) methods are combined into a single transaction, sometimes adding additional things like logging
- Because of this we will see here how the service layer can be seen as an entry point to a sub system.

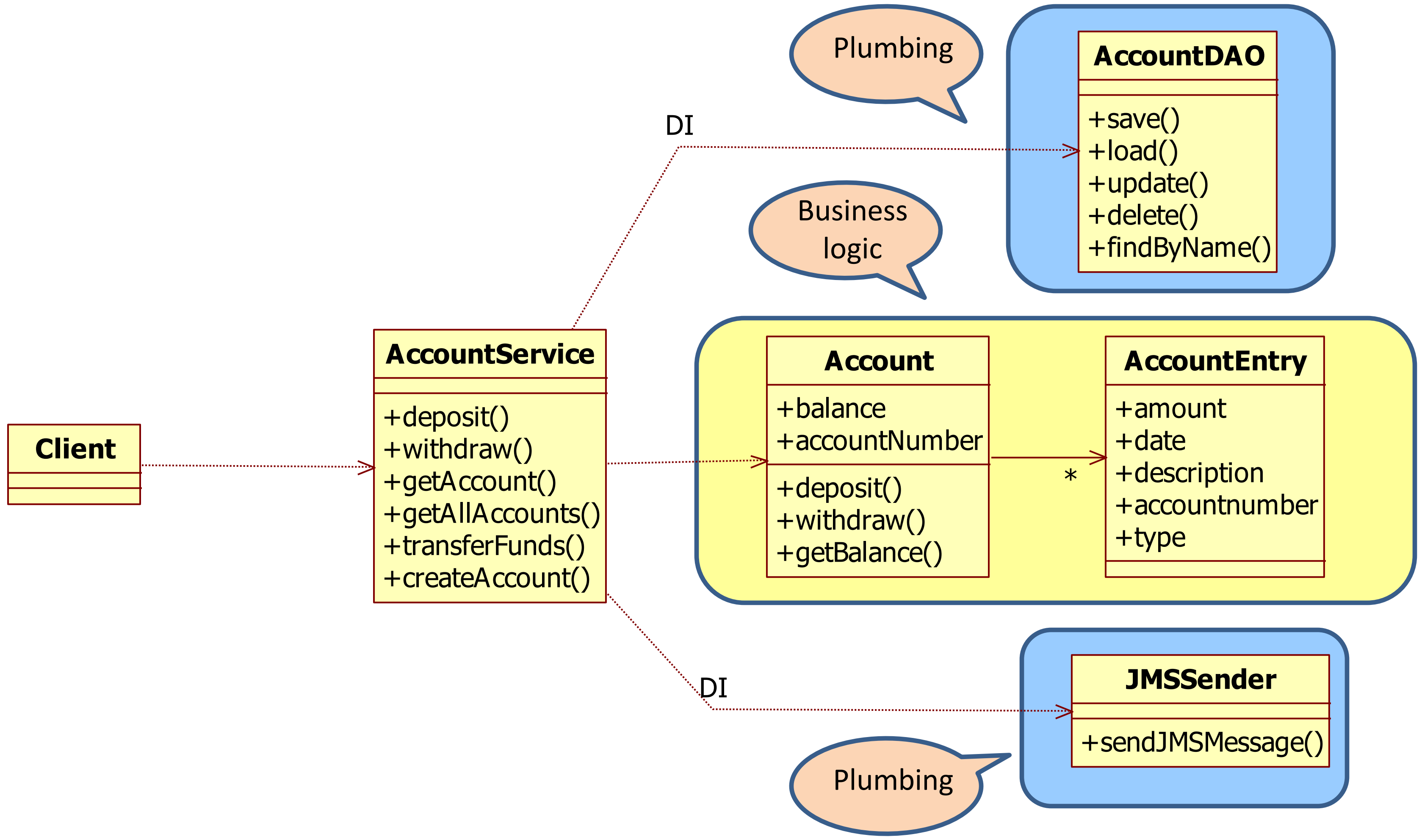
Service Object



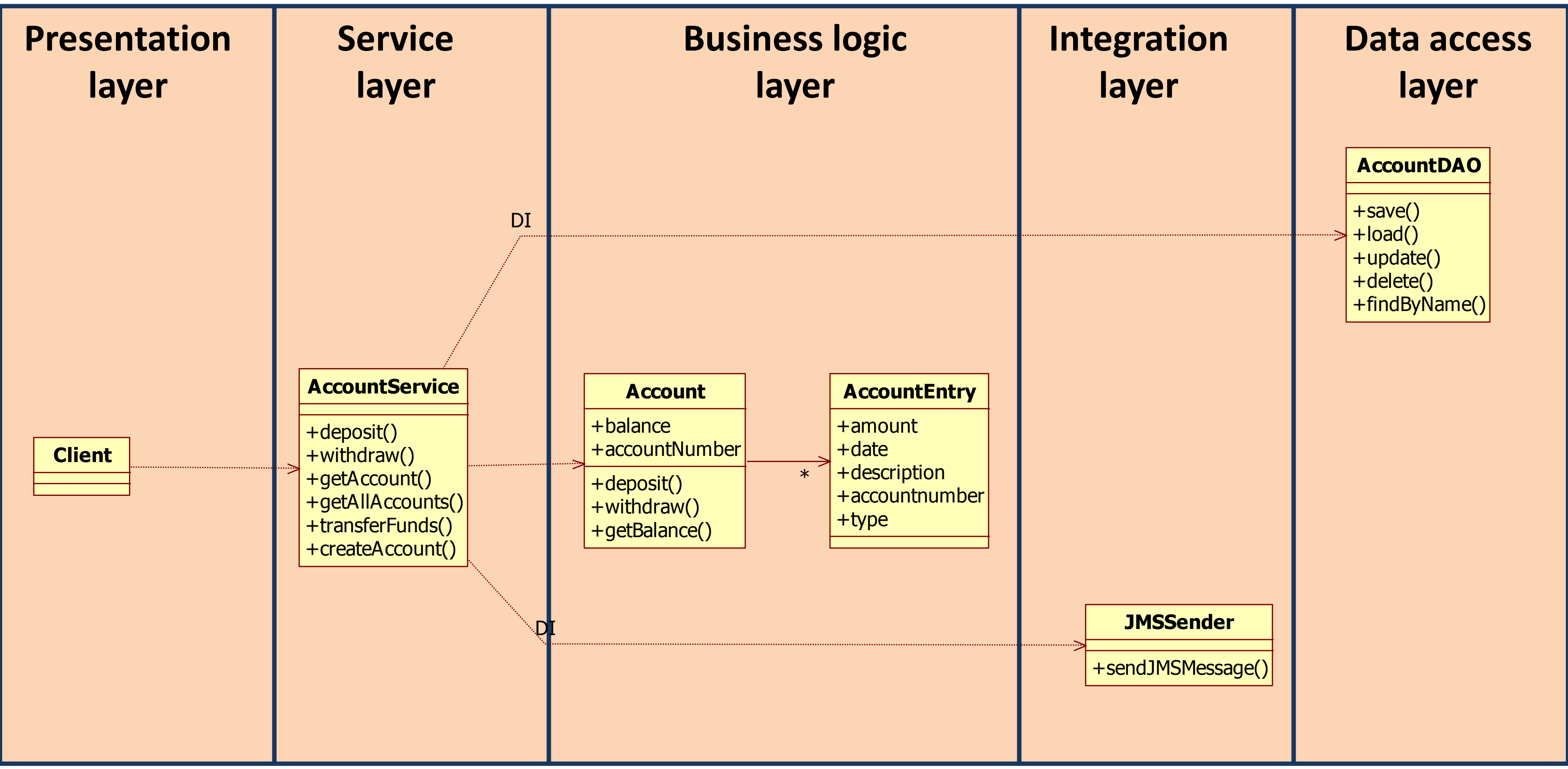
Entry of a complex subsystem



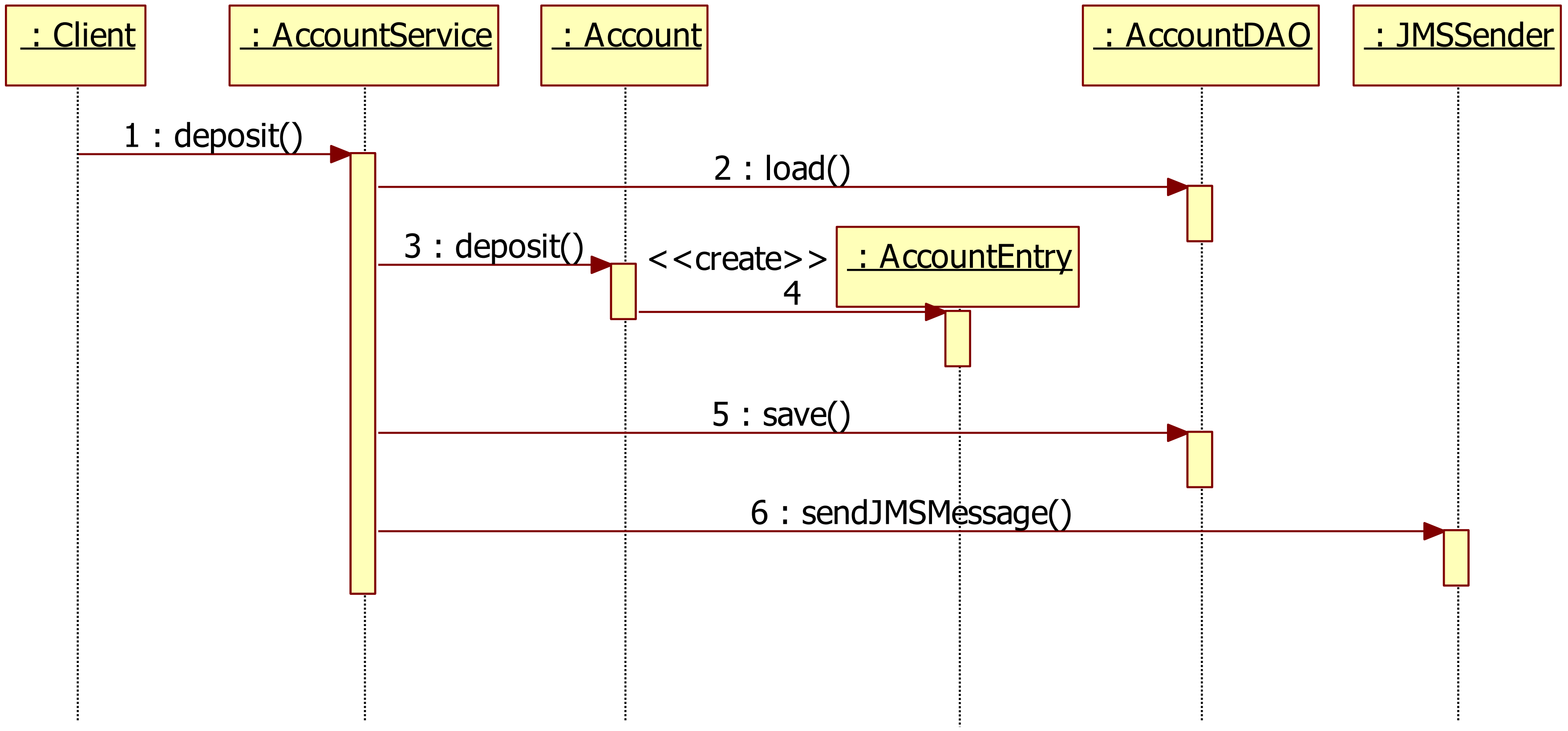
Separation of concern



Application layers



Service object



The Service Layer

- As our application grows we notice that we can differentiate between technology specific controllers and on a deeper level business control (Service Layer).
- A bigger application doesn’t mean a bigger mess, just a different organization; all that is needed is some awareness of what is going on.

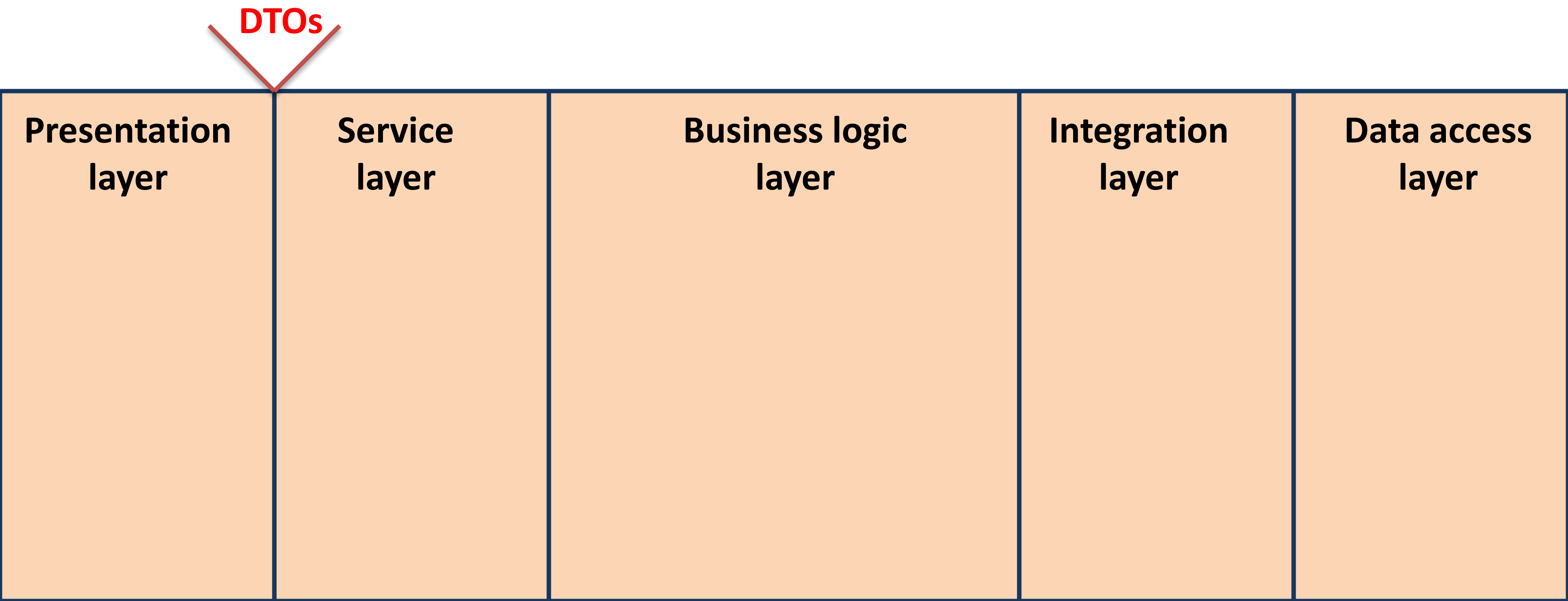
Data Access Objects

- In the course we mostly use Data Access Objects (DAOs aka Repositories)
- There are certain people who argue against their use, saying it's a 'cargo cult'
  - https://www.youtube.com/watch?v=zTKI7qeyLlw

- If you want to use them you can also generate them with Spring-Data

Data Transfer Objects

- Data Transfer Objects are classes to hold the data between the service layer and the view.
  - Valid approach without the OpenSessionInView
  - Does create a lot more classes, a lot more bloat



Service Layer:

MULTIPLE CONFIGURATION FILES

Multiple XML configuration files using include

```
public class Application {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(accountService.xml);
    IAccountService accountService = context.getBean("accountService", IAccountService.class);
    …
  }
}
```

Import other XML configuration files

accountService.xml

```
<beans …>
    <import resource="dataAccess.xml"/>
    <import resource="jmsService.xml"/>
    <bean id="accountService" class="bank.service.AccountService">
        <constructor-arg index="0" ref="accountDAO" />
        <constructor-arg index="1" ref="jmsSender" />
    </bean>
</beans>
```

dataAccess.xml

```
<beans …>
    <bean id="accountDAO" class="bank.dao.AccountDAO" />
</beans>
```

jmsService.xml

```
<beans …>
    <bean id="jmsSender" class="bank.jms.JMSSender" />
</beans>
```

Multiple XML configuration files through the ApplicationContext

3 XML files

```java
public class Application {
  public static void main(String[] args) {
    String[] xmlResources ={"accountService.xml", "dataAccess.xml", "jmsService.xml"};
    ApplicationContext context = new ClassPathXmlApplicationContext(xmlResources);
    IAccountService accountService = context.getBean("accountService", IAccountService.class);
    …
  }
}
```

accountService.xml

```xml
<beans …>
    <bean id="accountService" class="bank.service.AccountService">
        <constructor-arg index="0" ref="accountDAO" />
        <constructor-arg index="1" ref="jmsSender" />
    </bean>
</beans>
```

dataAccess.xml

```xml
<beans …>
    <bean id="accountDAO" class="bank.dao.AccountDAO" />
</beans>
```

jmsService.xml

```xml
<beans …>
    <bean id="jmsSender" class="bank.jms.JMSSender" />
</beans>
```

Multiple configuration classes with @Import

```
@Configuration
@Import(EmailConfig.class)
public class AppConfig {
  @Autowired
  EmailService emailService;

  @Bean
  public CustomerService customerService(){
    CustomerService customerService = new CustomerServiceImpl();
    customerService.setEmailService(emailService);
    return customerService;
  }
}
```

Import the EmailConfig class

Autowire the emailService

```
@Configuration
public class EmailConfig {

  @Bean
  public EmailService emailService(){
    return new EmailServiceImpl();
  }
}
```

Multiple Configuration Files

- Since our application itself is often large, (and found in layers) it makes sense to separate a large spring configuration file along the same lines, into the same layers.

Active Learning

- How does the Service layer support the concept of separation of concerns?

with dependency inyection

- What are the two ways that you can connect multiple configuration files together?

using include and through ApplicationContext

Summary

- Almost all Spring applications make use of service objects
- Service objects connect enterprise service objects (DOA's, loggers, email senders, etc) with the business logic objects.
  - Providing Separation of Concerns
- The Spring configuration file can be split up in several configuration files

Main Point

- The service layer provides a separation of concerns needed in larger applications.
- *Science of Consciousness*: Life is found in layers, just like our applications