Mindtree

Welcome to possible

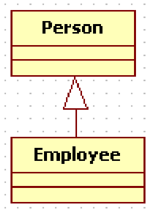
Introduction to Spring

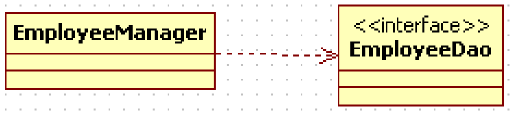
Objectives

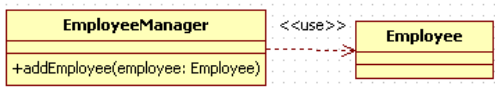
- Introduction to Spring Framework.
- Work with Bean properties in spring
- How to use Setter and Constructor DI?
- Understand overview of bean life cycle and Scope
- Autowiring

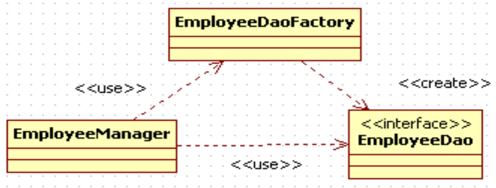
Dependency

- Dependency
  - Dependency refers to tying an component to another one using
    - Inheritance
      - Inheritance results in strong coupling and is extremely overused in OO. Only use inheritance when it really makes sense from the natural classes' definition, not to avoid duplicating code.
    - Association
      - Association creates loose coupling, because when the dependency's (EmployeeDao) interface or behavior changes, you only need to adapt the dependant (EmployeeManager).
    - Use of static methods
      - Dependency through use of static methods.
    - Method parameter
      - Dependency through method parameters.
    - Instantiation by Factory
      - EmployeeManager depends on EmployeeDao and EmployeeDaoFactory









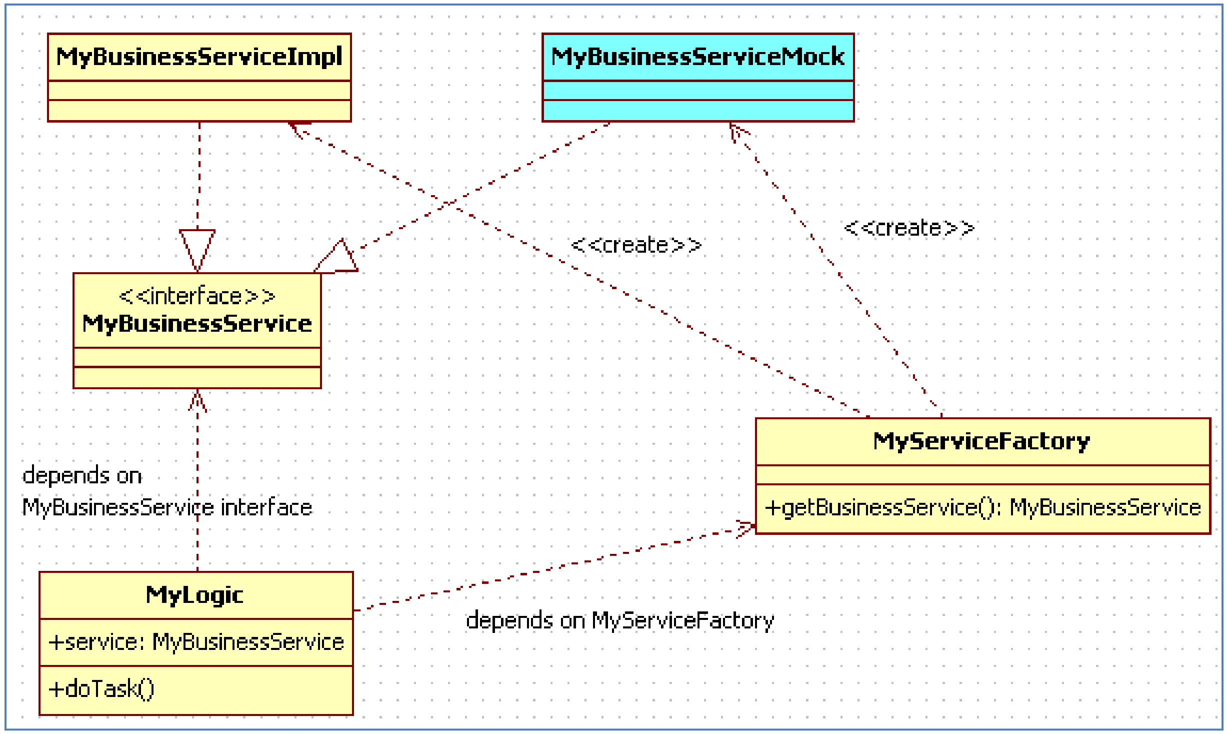
Dependency

- Having dependency means:
  - Dependent component is subject to change when the component it depends on is modified
  - Dependencies are not bad but the goal is to minimize the number of dependencies in your design.
  - Depend on interfaces rather than classes as much as possible.

  - The less likely your application is subject to change, the more stable and maintainable it becomes.

Dependency

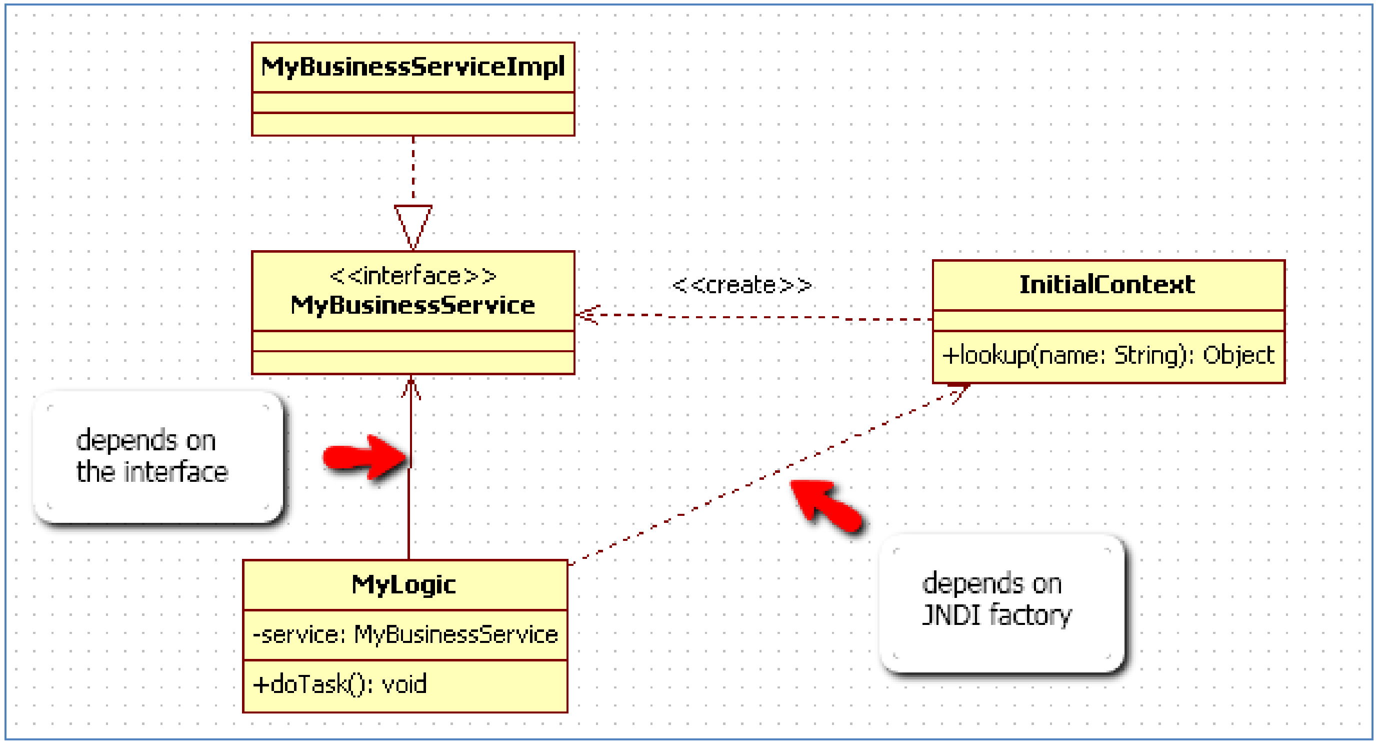
- Factory class



- By adding a level of abstraction (MyServiceFactory), MyLogic has no direct reference to MyBusinessServiceImpl/ MyBusinessServiceMock implementation classes.

Dependency

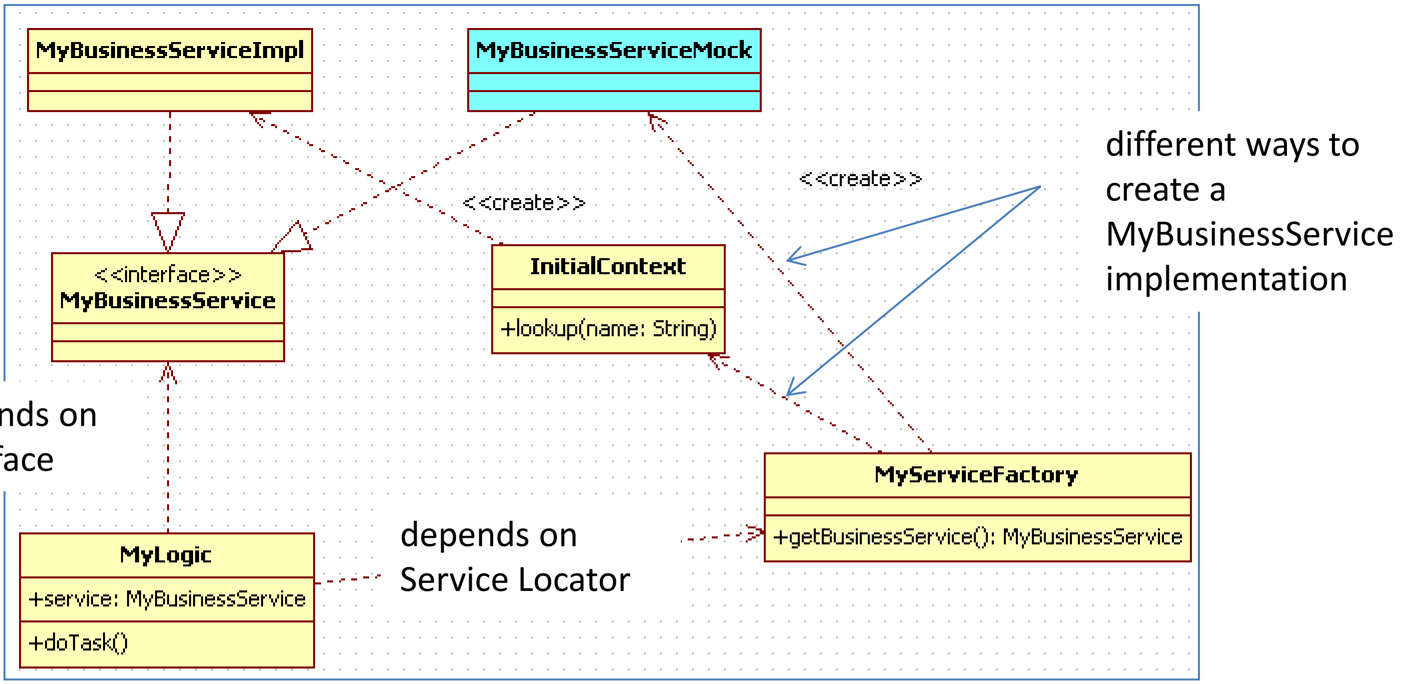
- Service Locator



- A service locator is a kind of factory that has the specialized purpose of obtaining a service.
- The Java Naming and Directory Interface (JNDI) is a good example of a service locator. It is often used by application servers to register resources at start time and later by deployed applications to look them up.

Dependency

- Factory and Service Locator



MyBusinessServiceImpl
MyBusinessServiceMock
<<create>>
<<create>>
different ways to create a MyBusinessService implementation
<<interface>> MyBusinessService
InitialContext
+lookup(name: String)
depends on interface
MyServiceFactory
+getBusinessService(): MyBusinessService
depends on Service Locator
MyLogic
+service: MyBusinessService
+doTask()

- MyServiceFactory abstracts the client application from the way your application gets MyBusinessService implementation class. So this code can be tested without an application server.
- But still MyLogic depends on MyServiceFactory and MyBuinessSerice interface.

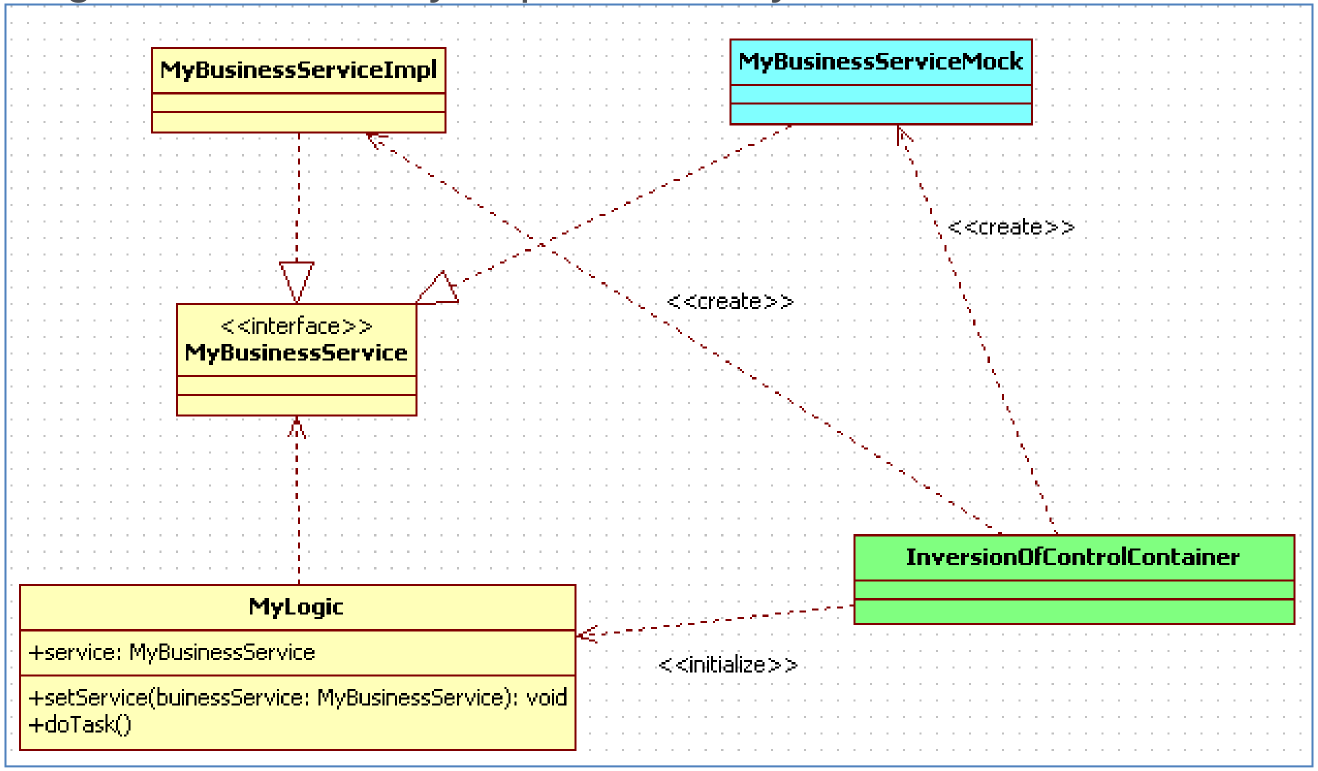
Dependency Injection

- The Dependency Injection pattern is based on the "inversion of control" (IoC) principle.

- The Dependency Injection relates to the way in which an object obtains references to its dependencies
  - The object is passed its dependencies through constructor arguments or after construction through setter methods or interface methods.
  - It is called dependency injection since the dependencies of an object are 'injected' into it.

Dependency Injection

- Dependency Injection is done by IoC Container.
- IoC Container creates the implementation and injects it into the MyLogic
- MyLogic class now only depends on MyBusinessService interface.



Mindtree

Mindtree

Welcome to possible

Spring Framework

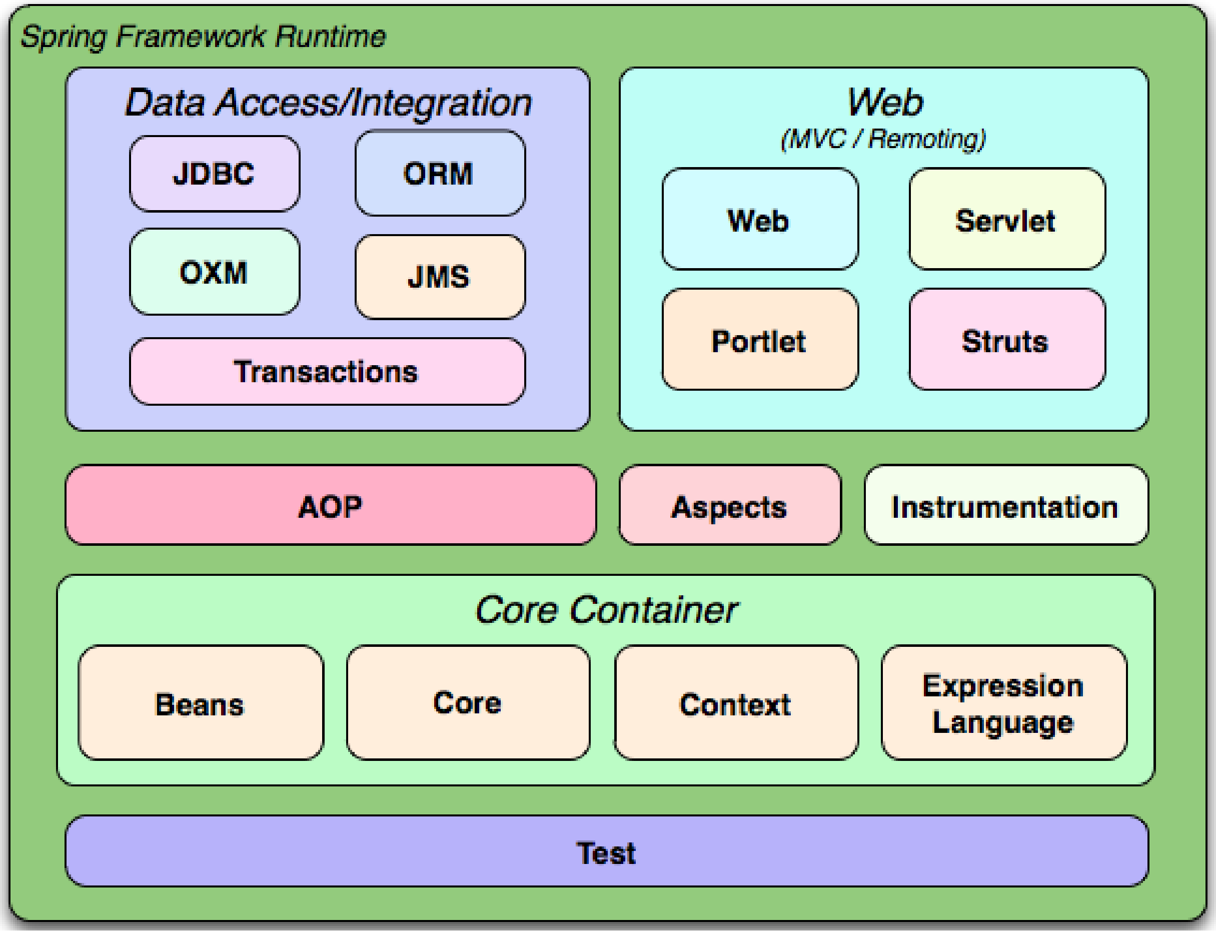
Dependency Injection using Spring Framework

Spring Framework

- What is Spring Framework?
    - Spring is a lightweight container framework to build enterprise components with simple Java objects.
- Why to use Spring Framework?
    - Spring is comprehensive and modular
        - Spring has a layered architecture, you choose any modules of spring framework in isolation
        - Example: You can choose spring only to manage all your business objects or you may use spring to simply use of JDBC, etc.
    - Applications built using Spring are very easy to unit test.
    - Spring supports easy configuration of aspects like transaction, logging, etc
    - Spring is designed so that applications built with it depend on as few of its APIs as possible. Most business objects in Spring applications have no dependency on Spring

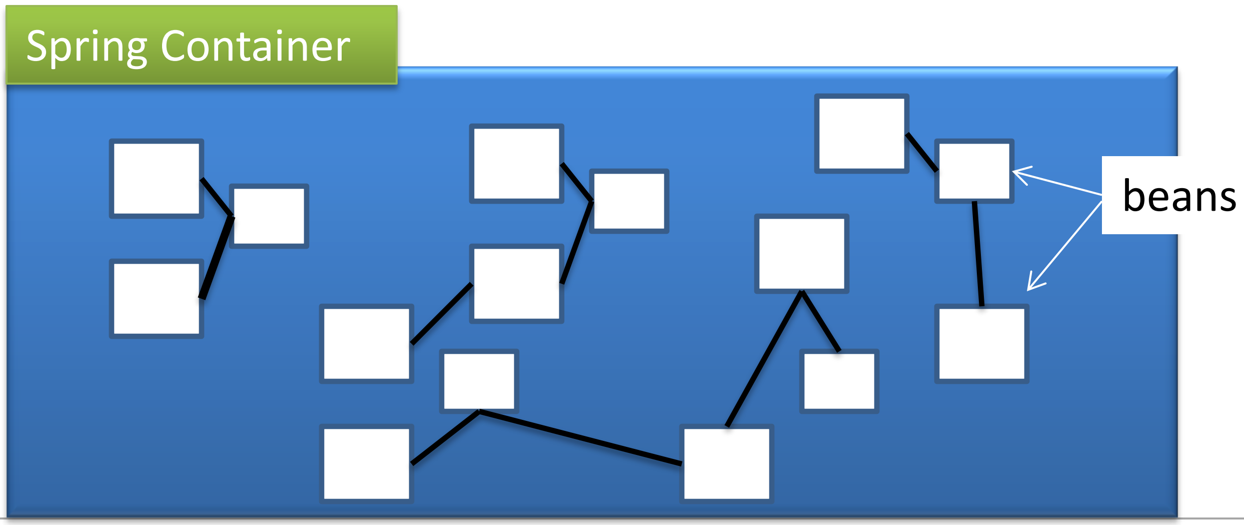
Spring Framework

- Modules of Spring Framework



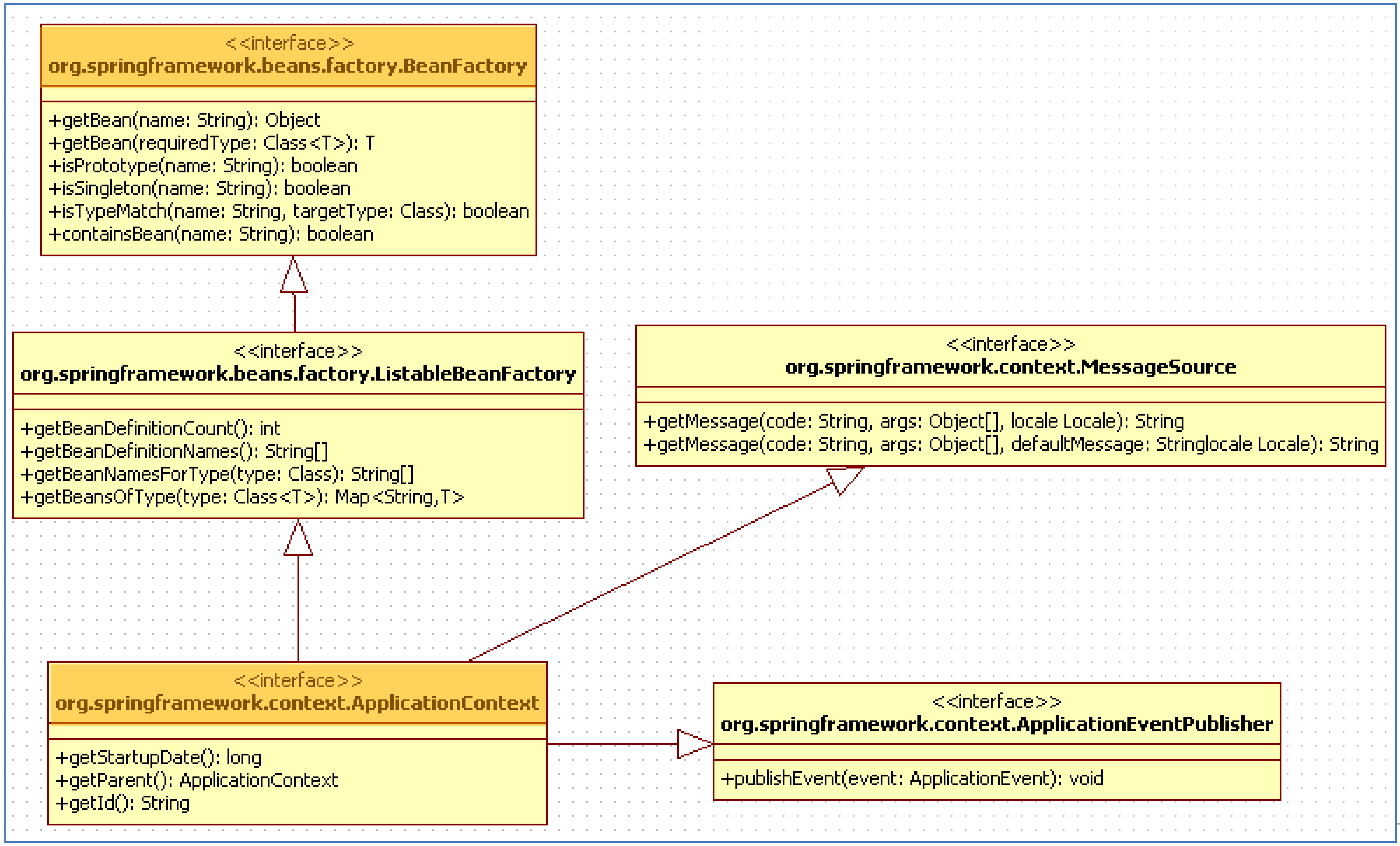
Spring Container

- Spring Container
  - Spring Container is the place were your application objects live in a Spring-based application.
  - The Spring container will create the objects, wire them together, configure them, and manage their complete lifecycle
  - The container is at the core of the Spring Framework. Spring's container uses dependency injection (DI) to manage the components that make up an application.



Spring Container and Beans

- Spring Container



<<interface>>
org.springframework.beans.factory.BeanFactory
+getBean(name: String): Object
+getBean(requiredType: Class<T>): T
+isPrototype(name: String): boolean
+isSingleton(name: String): boolean
+isTypeMatch(name: String, targetType: Class): boolean
+containsBean(name: String): boolean
<<interface>>
org.springframework.beans.factory.ListableBeanFactory
+getBeanDefinitionCount(): int
+getBeanDefinitionNames(): String[]
+getBeanNamesForType(type: Class): String[]
+getBeansOfType(type: Class<T>): Map<String,T>
<<interface>>
org.springframework.context.MessageSource
+getMessage(code: String, args: Object[], locale Locale): String
+getMessage(code: String, args: Object[], defaultMessage: Stringlocale Locale): String
<<interface>>
org.springframework.context.ApplicationContext
+getStartupDate(): long
+getParent(): ApplicationContext
+getId(): String
<<interface>>
org.springframework.context.ApplicationEventPublisher
+publishEvent(event: ApplicationEvent): void

Mindtree

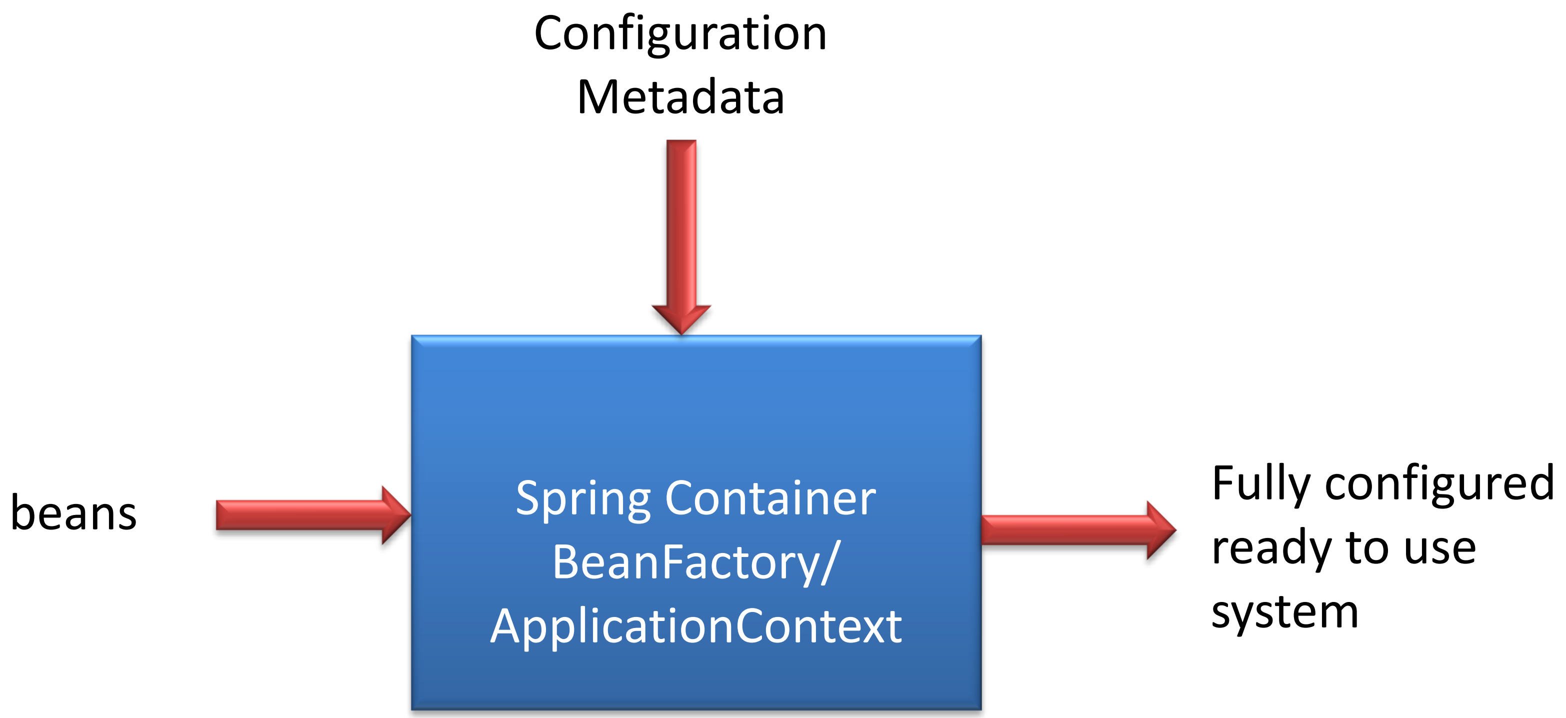
Spring Container and Beans

- Spring Container
  - BeanFactory
    - BeanFactory is an implementation of Factory design pattern, whose responsibility is to create and dispense beans.
    - BeanFactory is capable of creating associations between collaborating objects as they are instantiated.

  - ApplicationContext
    - ApplicationContext is an advanced Spring container.
    - ApplicationContext provides everything BeanFactory provides, plus it offers:
      - Generic way to load file resources
      - Supports Internationalization (i18N)
      - Can publish events to beans that are registered as listeners.
      - And many more.

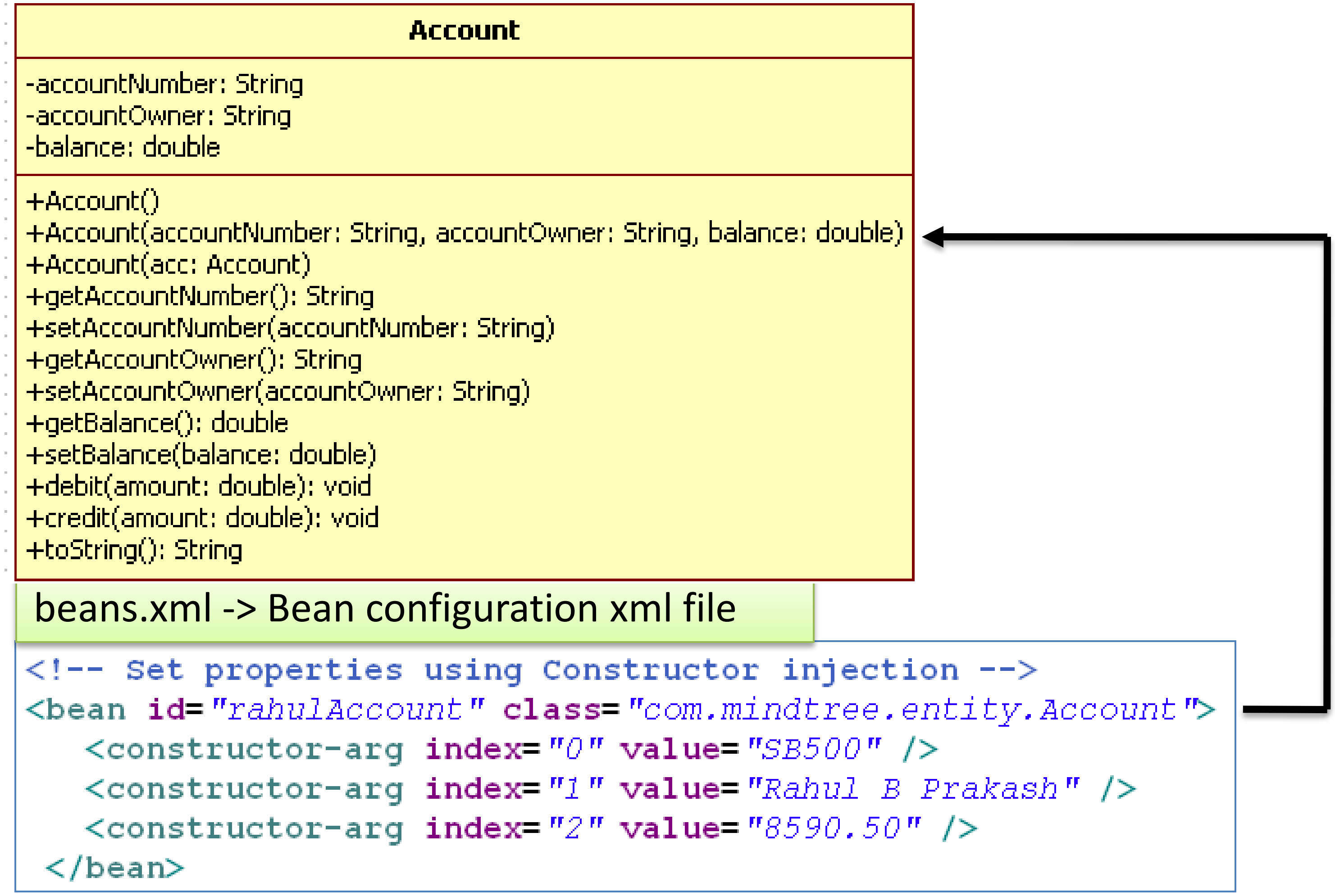
Spring Container and Beans

- Spring Container


Configuration Metadata
beans
Spring Container BeanFactory/ ApplicationContext
Fully configured ready to use system

Spring container in action

- Constructor based dependency injection

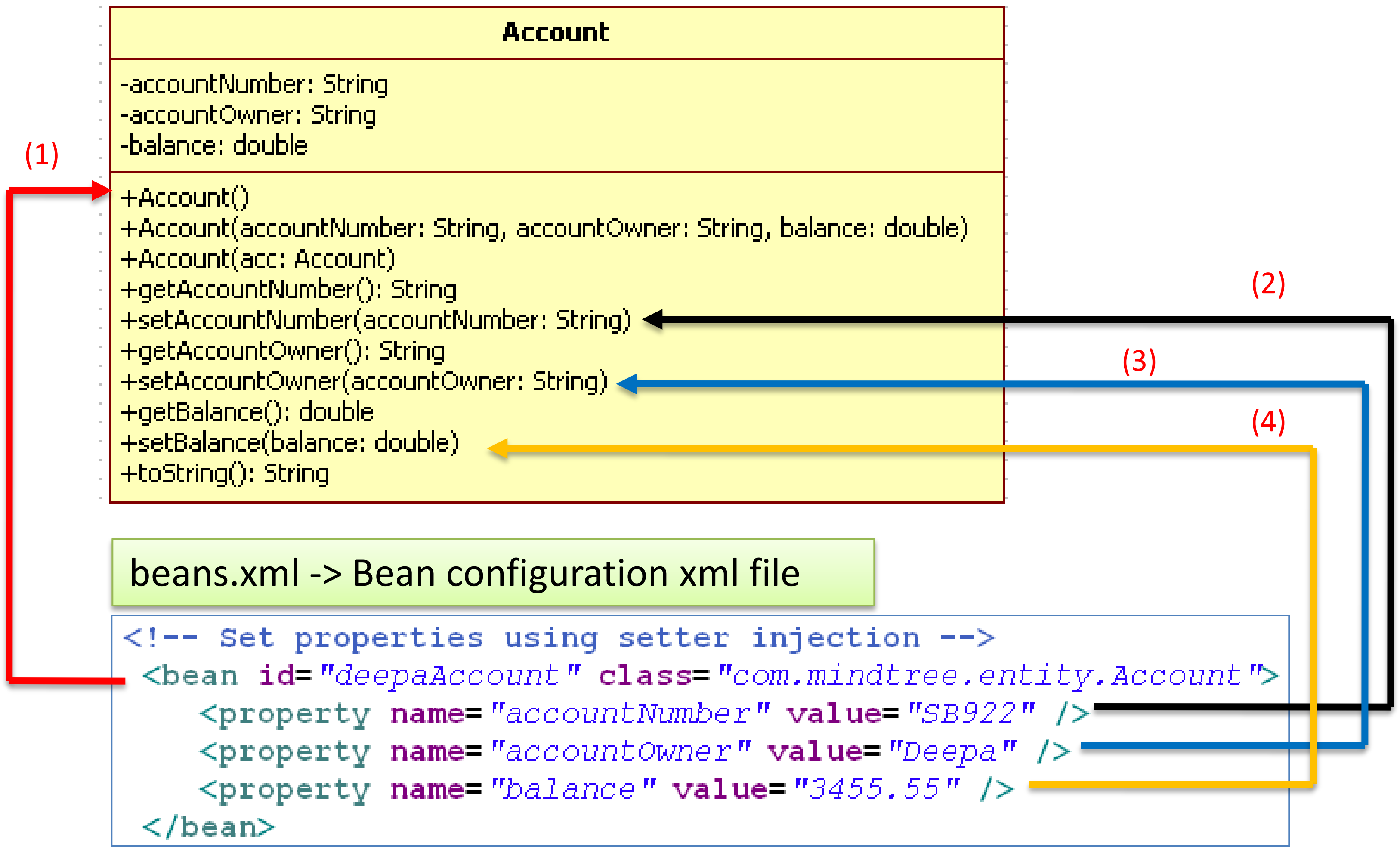


```
Account
-accountNumber: String
-accountOwner: String
-balance: double
+Account()
+Account(accountNumber: String, accountOwner: String, balance: double)
+Account(acc: Account)
+getAccountNumber(): String
+setAccountNumber(accountNumber: String)
+getAccountOwner(): String
+setAccountOwner(accountOwner: String)
+getBalance(): double
+setBalance(balance: double)
+debit(amount: double): void
+credit(amount: double): void
+toString(): String
```

beans.xml -> Bean configuration xml file

```
<!-- Set properties using Constructor injection -->
<bean id="rahulAccount" class="com.mindtree.entity.Account">
   <constructor-arg index="0" value="SB500" />
   <constructor-arg index="1" value="Rahul B Prakash" />
   <constructor-arg index="2" value="8590.50" />
 </bean>
```

Spring container in action: Setter based dependency Injection



```
Account
-accountNumber: String
-accountOwner: String
-balance: double
+Account()
+Account(accountNumber: String, accountOwner: String, balance: double)
+Account(acc: Account)
+getAccountNumber(): String
+setAccountNumber(accountNumber: String)
+getAccountOwner(): String
+setAccountOwner(accountOwner: String)
+getBalance(): double
+setBalance(balance: double)
+toString(): String
```

beans.xml -> Bean configuration xml file

```xml
<!-- Set properties using setter injection -->
 <bean id="deepaAccount" class="com.mindtree.entity.Account">
    <property name="accountNumber" value="SB922" />
    <property name="accountOwner" value="Deepa" />
    <property name="balance" value="3455.55" />
 </bean>
```

Mindtree

Spring Container in action

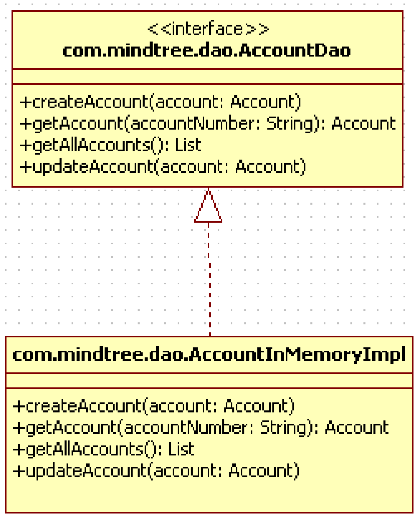
Bean configuration xml file

```
<bean id="accountDao" class="com.mindtree.dao.AccountInMemoryImpl">
</bean>

<!-- Set properties using Constructor injection -->
<bean id="rahulAccount" class="com.mindtree.entity.Account">
   <constructor-arg index="0" value="SB500" />
   <constructor-arg index="1" value="Rahul B Prakash" />
   <constructor-arg index="2" value="8590.50" />
 </bean>

<!-- Set properties using setter injection -->
 <bean id="deepaAccount" class="com.mindtree.entity.Account">
    <property name="accountNumber" value="SB922" />
    <property name="accountOwner" value="Deepa" />
    <property name="balance" value="3455.55" />
 </bean>
```

(1)

(2)

(3)



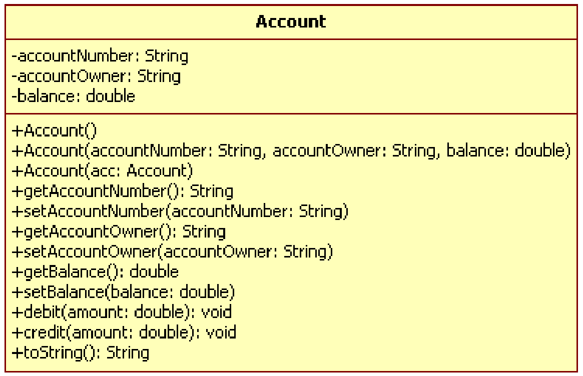
<<interface>>
com.mindtree.dao.AccountDao
+createAccount(account: Account)
+getAccount(accountNumber: String): Account
+getAllAccounts(): List
+updateAccount(account: Account)
com.mindtree.dao.AccountInMemoryImpl
+createAccount(account: Account)
+getAccount(accountNumber: String): Account
+getAllAccounts(): List
+updateAccount(account: Account)

```
ApplicationContext context  =
            new ClassPathXmlApplicationContext("beans.xml");

Account rahulAcc =
          context.getBean("rahulAccount", Account.class);
Account deepaAcc =
          context.getBean("deepaAccount", Account.class);
AccountDao accountDao =
          context.getBean("accountDao", AccountDao.class);
accountDao.createAccount(rahulAcc);
accountDao.createAccount(deepaAcc);

List<Account> accounts = accountDao.getAllAccounts();
```

(4)

(5)

(6)

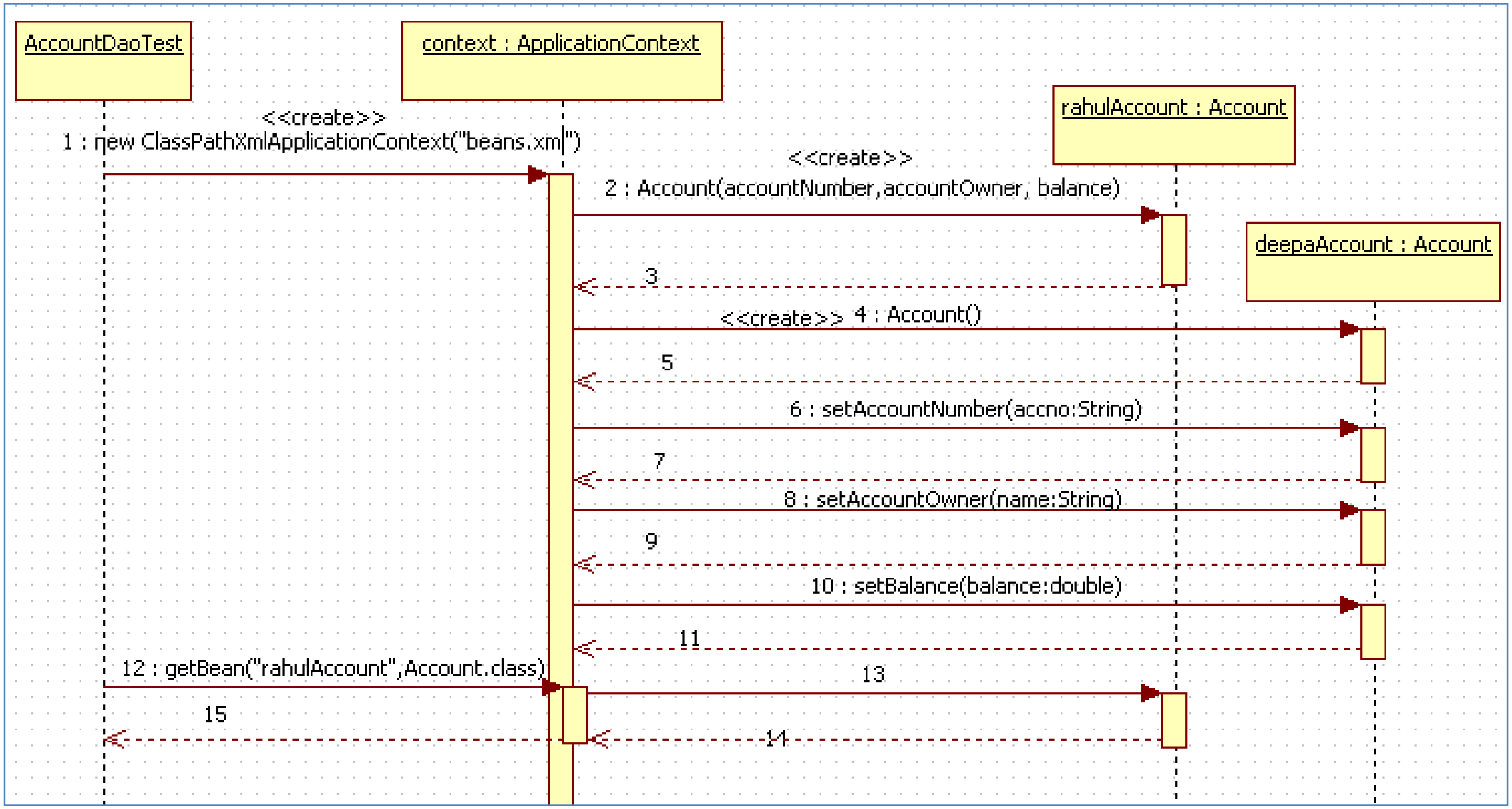


Account
-accountNumber: String
-accountOwner: String
-balance: double
+Account()
+Account(accountNumber: String, accountOwner: String, balance: double)
+Account(acc: Account)
+getAccountNumber(): String
+setAccountNumber(accountNumber: String)
+getAccountOwner(): String
+setAccountOwner(accountOwner: String)
+getBalance(): double
+setBalance(balance: double)
+debit(amount: double): void
+credit(amount: double): void
+toString(): String

Mindtree

Spring container in action

- Sequence Diagram


AccountDaoTest
context : ApplicationContext
rahulAccount : Account
deepaAccount : Account
<<create>>
1 : new ClassPathXmlApplicationContext("beans.xml")
<<create>>
2 : Account(accountNumber,accountOwner, balance)
3
<<create>> 4 : Account()
5
6 : setAccountNumber(accno:String)
7
8 : setAccountOwner(name:String)
9
10 : setBalance(balance:double)
11
12 : getBean("rahulAccount",Account.class)
13
14
15

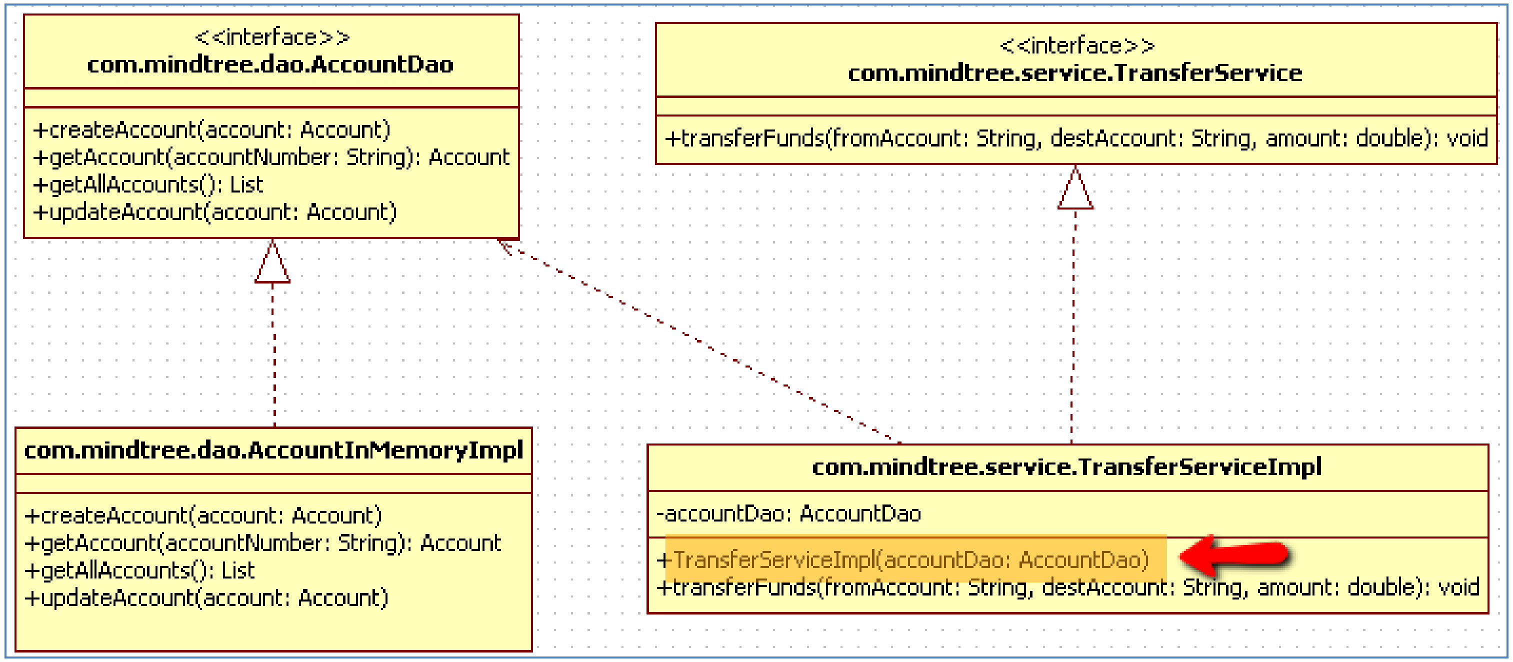
Autowiring collaborators

- The Spring container can *autowire relationships between collaborating beans.*
  - *You can allow Spring to* resolve collaborators (other beans) automatically for your bean by inspecting the contents of the ApplicationContext.

Mode	Explanation
no	No autowiring. Bean references should be defined via “ref” attribute.
byName	Autowiring by property name. Spring looks for a bean with the same name as the property that needs to be autowired.
byType	Allows a property to be autowired if exactly one bean of the property type exists in the container
constructor	Analogous to *byType, but applies to constructor arguments*
autoWire	Attempts to autowire by constructor first and then using byType

Autowiring collaborators

- Autowiring using autowire=“constructor”



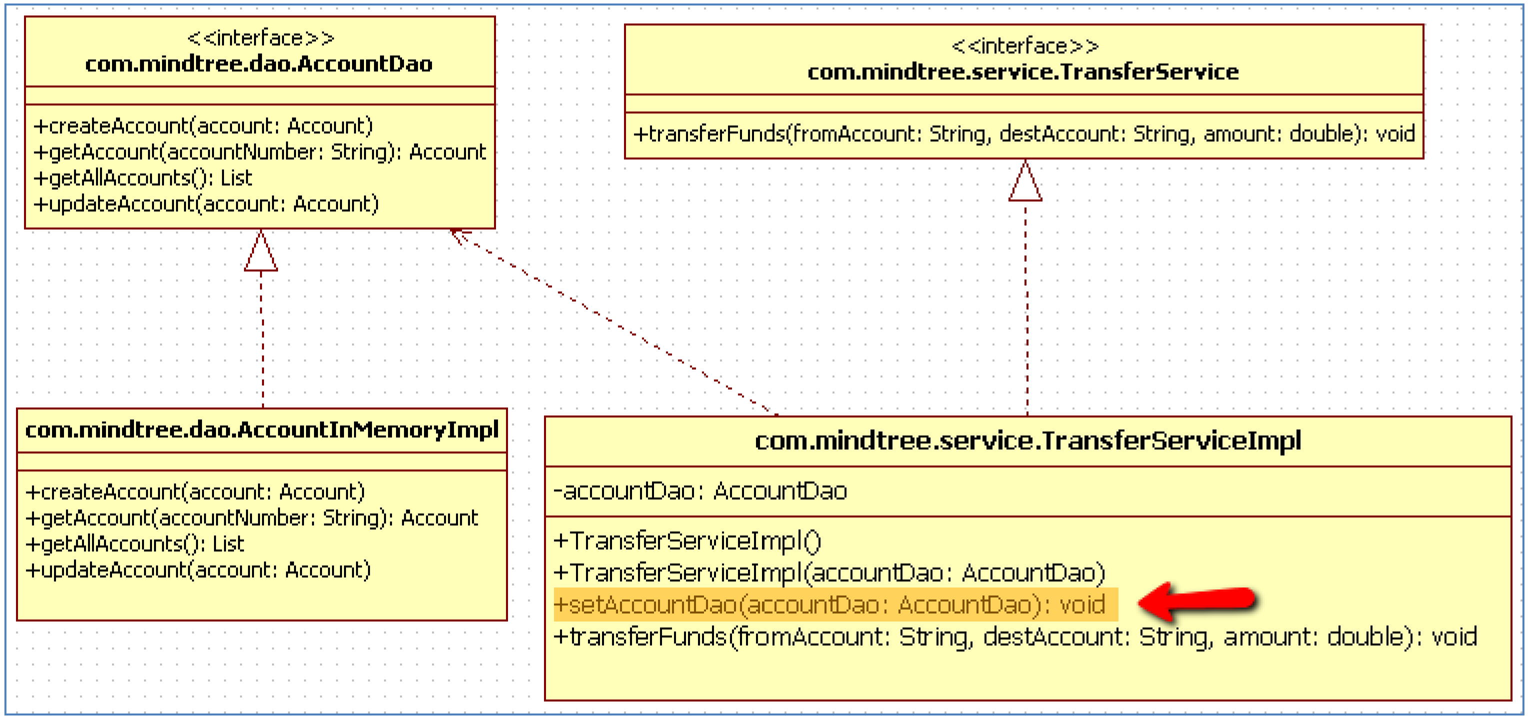
<<interface>>
com.mindtree.dao.AccountDao
+createAccount(account: Account)
+getAccount(accountNumber: String): Account
+getAllAccounts(): List
+updateAccount(account: Account)
<<interface>>
com.mindtree.service.TransferService
+transferFunds(fromAccount: String, destAccount: String, amount: double): void
com.mindtree.dao.AccountInMemoryImpl
+createAccount(account: Account)
+getAccount(accountNumber: String): Account
+getAllAccounts(): List
+updateAccount(account: Account)
com.mindtree.service.TransferServiceImpl
-accountDao: AccountDao
+TransferServiceImpl(accountDao: AccountDao)
+transferFunds(fromAccount: String, destAccount: String, amount: double): void

```
<bean id="accountDao" class="com.mindtree.dao.AccountInMemoryImpl">
</bean>

 <bean id="transferService"
    class="com.mindtree.service.TransferServiceImpl" autowire="constructor" />
```

Mindtree

Autowiring collaborators

- Autowiring using autowire=“byType”



<<interface>>
com.mindtree.dao.AccountDao
+createAccount(account: Account)
+getAccount(accountNumber: String): Account
+getAllAccounts(): List
+updateAccount(account: Account)
<<interface>>
com.mindtree.service.TransferService
+transferFunds(fromAccount: String, destAccount: String, amount: double): void
com.mindtree.dao.AccountInMemoryImpl
+createAccount(account: Account)
+getAccount(accountNumber: String): Account
+getAllAccounts(): List
+updateAccount(account: Account)
com.mindtree.service.TransferServiceImpl
-accountDao: AccountDao
+TransferServiceImpl()
+TransferServiceImpl(accountDao: AccountDao)
+setAccountDao(accountDao: AccountDao): void
+transferFunds(fromAccount: String, destAccount: String, amount: double): void

```
<bean id="accountDao" class="com.mindtree.dao.AccountInMemoryImpl">
</bean>

 <bean id="transferService"
    class="com.mindtree.service.TransferServiceImpl" autowire="byType" />
```

Mindtree

Autowiring collaborators

- Annotation based autowiring
  - In Spring 2.5, autowire annotation was introduced to autowire beans together by type without the “autowire” attribute in the configuration file.
  - The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished.

```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:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
         http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:annotation-config />

</beans>
```

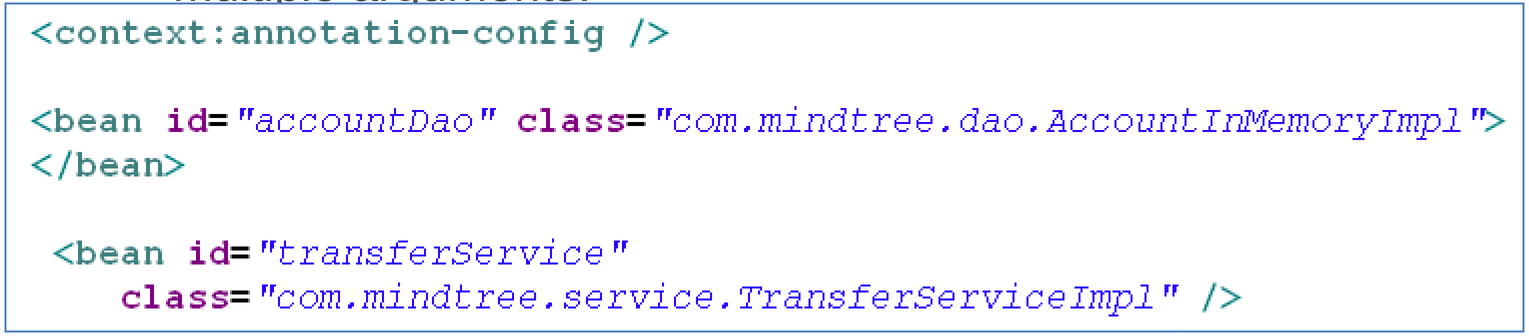
Include context namespace

looks for annotations on beans in the same application context in which it is defined

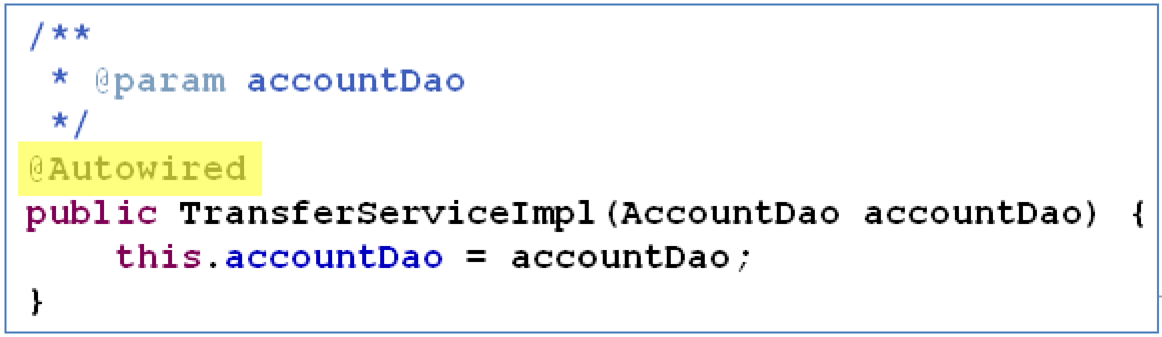
Mindtree

Autowiring collaborators

- Annotation based autowiring
  - @Autowired can be applied to setter methods, constructors and fields.
  - You can also apply @Autowired to methods with arbitrary names and/or multiple arguments.



```
<context:annotation-config />

<bean id="accountDao" class="com.mindtree.dao.AccountInMemoryImpl">
</bean>

 <bean id="transferService"
    class="com.mindtree.service.TransferServiceImpl" />
```



```
/**
 * @param accountDao
 */
@Autowired
public TransferServiceImpl(AccountDao accountDao) {
    this.accountDao = accountDao;
}
```

Notice that there is no autowire attribute nor explicit wiring using “ref” attribute”

Mindtree

Mindtree

Welcome to possible

Explore More!!

Never let your curiosity die!

Explore

- **Fine-tuning annotation-based autowiring with qualifiers**
  - http://static.springsource.org/spring/docs/2.5.x/reference/beans.html#beans-annotation-config

- **Lifecycle callbacks**
- **Internationalization using MessageSources**
  - http://static.springsource.org/spring/docs/2.5.x/reference/beans.html#beans-factory-extension

- Spring Best practices
  - http://www.mydeveloperconnection.com/html/SpringBP.htm

Mindtree
Welcome to possible

References

Contains the reference that will supplement the self learning and will be needed for completing the assignments & practice questions

Reference

- **Spring Collections (List, Set, Map, and Properties) example**
  - http://www.mkyong.com/spring/spring-collections-list-set-map-and-properties-example/
  - http://www.mkyong.com/wp-content/uploads/2010/03/Spring-Collection-Example.zip
- Spring bean scopes
  - http://static.springsource.org/spring/docs/2.5.x/reference/beans.html#beans-factory-scopes-singleton
  - http://static.springsource.org/spring/docs/2.5.x/reference/beans.html#beans-factory-scopes-prototype
  - http://www.mkyong.com/spring/spring-bean-scopes-examples/
- Creating beans using static factory method
  - http://www.coderanch.com/t/500009/Spring/Creating-Beans-invoking-static-factory
- Beginners guide to dependency injection
  - http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection?_sm_au_=iVVDsF4M4QtKLnHH

Welcome to possible

India | USA | UK | Germany | Sweden | Belgium | France | Switzerland | UAE | Singapore | Australia | Japan | China