Java - Spring

https://spring.io/

- dasturlash.uz

Framework

- What exactly is a framework?
- A framework is a particular set of rules, ideas, or beliefs which you use in order to deal with problems or to decide what to do.
- Framework bu qandaydir muommoni yoki ishni osonlik bilan yechib beradigan qoidalar/kodlar to’plami.

- dasturlash.uz

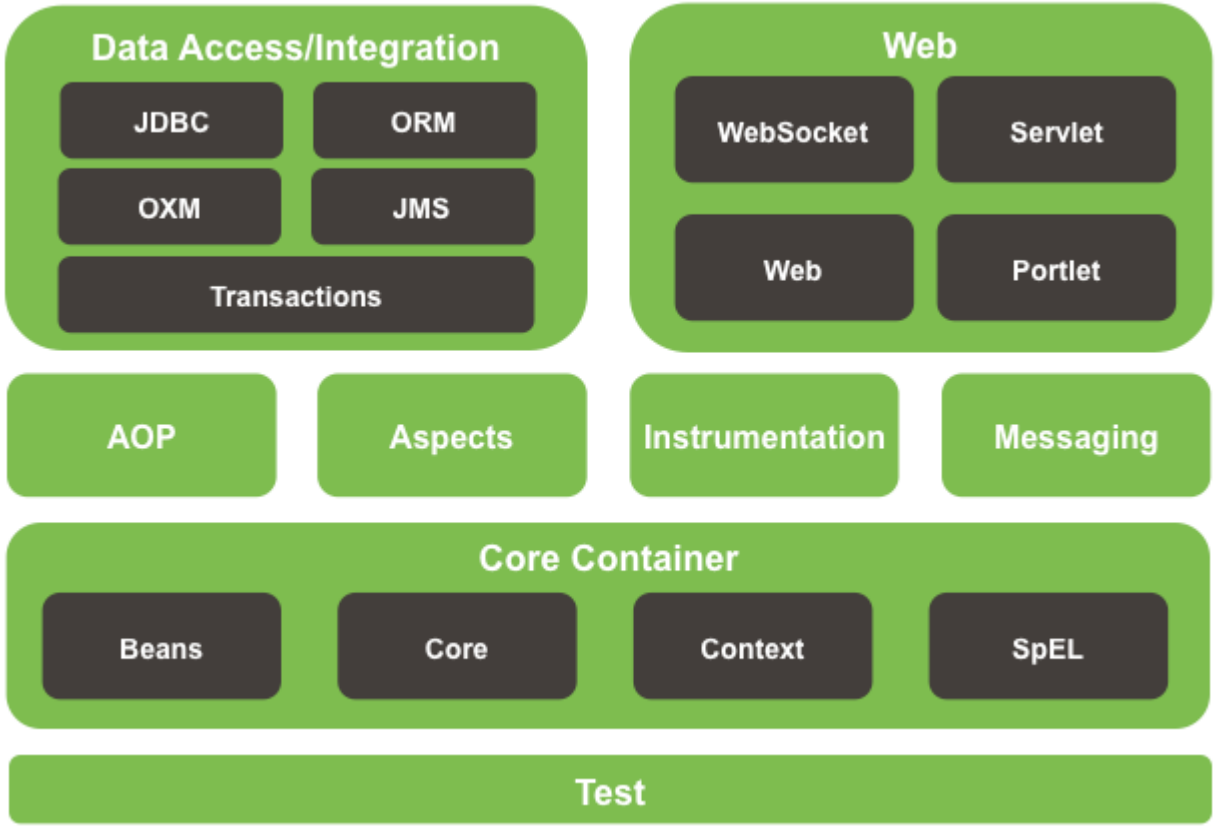
What is Spring

- It is a lightweight , loosely coupled and integrated framework for developing enterprise applications in java.
- An open source Framework
- And I was introduced an alternative to heavier enterprise Java technologies
- For example Spring is an alternative to ejb (Enterprise Java Beans)
- And also:
  - Spring addresses the complexity of enterprise application development. (Решает сложности разработки корпоративных приложением)
- In simple words: Spring simplifies Java development.

- Bu java-da korporativ (katta) ilovalarni ishlab chiqish uchun mojjallangan engil, Bir-biriga kuchli bog'lanmagan va yengil integratsiya qilish mumkin bo'lgan framework hisoblanadi.
- Ochiq manbali Framework.
- Spring javani og'ir ishlaydigan enterprise texnalogiyalari o'rniga taqdim etildi.
- Oddiy gaplar bilan Spring simplifies java development.

Spring Modules

- There are more than 21 modules



- dasturlash.uz

Spring Modules

- Test
  - This layer provides support at testing with Junit and TestNG
- Spring Core Container
  - Core and Beans – These modules provide IOC and Dependency Injection features
  - Context – suppotrs internatiolalization (I18N,EJB, JMS,Basic Remoting)
  - Expression Language (**SpEL**) (Язык выражение) It in as extension to the EL defined in JSP.

Spring Expression Language

- SpEL (Example)
- The Spring Expression Language (SpEL) is a powerful expression language that supports querying and manipulating an object graph at runtime. It can be used with XML or annotation-based Spring configurations.
- @Value("John")
  private String firstName;
  @Value("Doe")
  private String lastName;
  @Value("#{user.firstName.concat(' ').concat(user.lastName)}") private String fullName;

- dasturlash.uz

Spring Module

- Data Access / Integration
- JDBC
  - Provides a JDBC abstraction layer that removes the need to do tedious JDBC coding (parsing, opening connection,…)
- ORM
  - Provides integration layer for popular relational mapping API's (JPA, Hibernatem , ….)
- OXM
  - Provides an abstraction layer for using a number of Object/XML mapping. Implementations.
- JMS
  provides Spring's support for Java Messaging Service. It contains features for both producing and consuming messages

Spring Module

- WEB
  - Web, Web-Servlet, Web-Struts, Web-Portlet are provides support to create web application.
- Web-Servlet – provides spring's MVC
- Web – provides basic web-oriented integration features

Charchamaganskiy ?

- dasturlash.uz

Spring IOC Container 1

- The **Spring container** (also knows as IOC container) is at the core of Spring Framework.
- **IoC - Inversion Of Control**
- In Spring Objects get stored and live in container
- The Container will create the objects, write the created objects together configure them and manger their complete life cycle from creation till destruction.
- **Spring container –** Spring Framework ning asosiy qismi xisoblanadi. Uni yana **IOC Container** deb ham atashadi
- **IOC** – **Inversion Of Control** - Boshqarishning inversiyasi deb tarjima qilinadi.
- Spring da Ob’ektlar Container da istiqomat qiladi.
- Container ob’ektlarni yaratish, ularni konfiguratsiya qilish, ularni kerakli ob’ektlar bilan taminlash , hullas ob’ekt yaratilishidan boshlab toki ochirilishiga qadar jarayonlarni boshqaradi.

Spring IOC Container 2

- The main tasks performed by IoC container are:
  - to instantiate the application class
  - to configure the object
  - to assemble the dependencies between the objects
- Spring container uses **Dependency Injection** (**DI**) to manage all the components that from an appliciton
- IoC container ni asosiy qiladigan ishla
  - Application da Class larni yaratish
  - ob'ektni konfiguratsiya qilish
  - ob'ektlar orasidagi bog'liqliklarni to’girlash.
- Spring Container **Dependency Injection** (**DI**) ni Dasturdagi Componentlar/Class larni boshqarish uchun ishlatadi.

Spring IOC Container 3

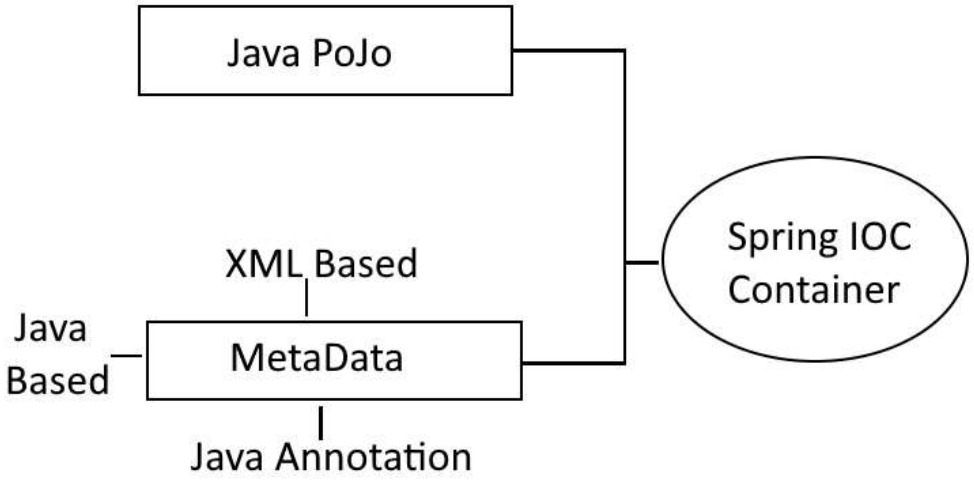
- Spring Container takes two things as input
  - Java based PoJo classes
  - Configuration Metadata (Which contains the complete information about what bean to create, its type, dependencies, etc…)
    - XML config file
    - Java Config file
    - Annotation based

POJO

- POJO stands for Plain Old Java Object.
- It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any classpath.
- POJOs are used for increasing the readability and re-usability of a program.

- POJO oddiy eski Java ob'ektini anglatadi.
- Bu oddiy Java ob'ekti bo'lib, Java tili spetsifikatsiyasi tomonidan majburlanganidan tashqari hech qanday maxsus cheklovlar bilan bog'lanmagan va hech qanday sinf yo'lini talab qilmaydi.
- POJOlar dasturning o'qilishi va qayta ishlatilishini oshirish uchun ishlatiladi.

Spring IOC Container 4



- dasturlash.uz

IOC Container type

- There are two types of IoC containers. They are:
  - 1. Bean Factory
  - 2. ApplicationContext

- The ApplicationContext interface is built on top od the BeanFactory interface
- It Adds some extra functionality than BeanFactory such as simple integration with Spring' AOP. Message resource handling (for I18n) ….
- XmlBeanFactory is the implementation class for BeanFactory interface

BeanFactory Example

```
Resources res = new ClassPathResources(“config.xml”)

BeanFactory factory = new XmlBeanFactory(res).

// get bean from favtory
```

ApplicationContext Example

- ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface.

```
ApplicationContext context = new
        ClassPathXmlApplicationContext(“config.xmal”);

// get bean fron context
```


dasturlash.uz

Bean

- dasturlash.uz

Bean 1

- A bean is an object that is instantiated, assempled and otherwise managed by a spring IOC container.
- These beans are created with the configuration metadata that you supply to container
- Bean definition contains the information called configuration metadata which is needed for the container to know:
  - How to create a bean
  - Bean’s lifecycle detail
  - Bean’s dependency

- Bean bu IoC container tomonidan yaratilgan va boshqariladigan ob’ekt dir.
- Bu bean lar IoC container ga berilgan konfiguratsiya metama'lumotlariga asosan yaratilgan.
- Bean da konteyner uchun zarur bo'lgan konfiguratsiya metama'lumotlar bor:
  - Bean ni qanday yaratish kerak
  - Bean hayot tartibi
  - Beanga kerak bo’lgan malumotlar

Bean 2

- Bean – How to create:
- **XML Based**

```
<bean id="..." class="....." >
... properties .....
</bean>
```

- **Java Based:**

```
@Component
public class IT {

}
```

```
<bean name="student" class="com.company.Student"></bean>
```

- dasturlash.uz

Bean Properties:

- class – attribute is mandatary and specifies the bean class to be used to create the bean
- name – attribute specifies the bean identifier uniquely. (id or name attributes to specify the bean identifier)
- scope – attribute specifies the scope of the objects. (defines a lifecycle)
- constructor-arg - this is used to inject the dependencies.
- properties – this is used to inject the dependencies
- autowiring mode ….
- lazy - initialization mode. A Lazy – initialized bean tells the IOC container tp create a bean when it is firs requested, rather than at the startup.
- initialization method – a callback to be called just after all necessary properties on the bean have been set by the container
- destruction method – a call back used when the container trying to destroy bean.

- dasturlash.uz

Lest a Spring Project

- dasturlash.uz

Project 1

- 1. Create a maven project (Spring_helloWordl)
- Add following dependency

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.5</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.5.RELEASE</version>
</dependency>
```

Project 2

- Create spring-config.xml file.
- Add Following codes:

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

</beans>
```

Project 2

- Create a Professor Class
  - Age
  - Name

- dasturlash.uz

What else ?

- 1. We need Create a bean
- 2. Get Bean from Spring Container.
- Lets do it.

- dasturlash.uz

Creating a Bean 1

- Creating a bean from Professor class with no value

```
<bean name=“professor1" class="com.company.Professor"></bean>
```

- Create a bean from Professor class with values

```
<bean name=“professor2" class="com.company.Professor">
    <property name="name" value="Alish"></property>
    <property name="age" value="22"></property>
</bean>
```

```
public class Professor {
    private Integer age;
    private String name;

    public Professor () {
    }


    // getter - setter

}
```


dasturlash.uz

Creating a Bean 2

- Ctreate Professor bean using constructor value

```
<bean name=“professor3" class="com.company.Professor">
      <constructor-arg value="10"></constructor-arg>
      <constructor-arg value="Alish"></constructor-arg>
  </bean>
```

```
public class Professor {
    private Integer age;
    private String name;

    public Professor(Integer age, String name) {
        this.age = age;
        this.name = name;
    }
    // getter - setter
}
```

Get Bean From IoC container

```
public class Main {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("spring-config.xml");
        Professor professor = (Professor) context.getBean(“professor1");
        System.out.println(professor);
    }
}
```

Lest Create a Lesson class

- Lesson class hash name and Professor atribute.
- We want set into professor a bean with name professor2

```
public class Lesson {
    private String name;
    private Professor professor;
}
```

Think about it?

- How we can set a bean into another bean.

Create a Lesson Bean

```
<bean name="professor2" class="com.company.Professor">
      <property name="name" value="Alish"></property>
      <property name="age" value="22"></property>
   </bean>

   <bean name="lesson" class="com.company.Lesson">
      <property name="name" value="Alish"></property>
      <property name="professor" ref="professor2"></property>
   </bean>
```

Dependency Injection

- dasturlash.uz

Dependency Injection 1

- Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.
- Dependency Injection makes our programming code loosely coupled.

- Dependency Injection (DI) – bu dasturni boshqarish, ishlash va test qilish oson bo’lishi uchun dasturlash kodidan bog’liqlikni olib tashlaydigan Design pattern.
- Dependency Injection (DI) - bu design pattern bo’lib u dasturlash kodidan bog'liqlikni olib tashlaydi.
- Dependency Injection bizning dasturlash kodimizni bir biriga bog‘liqlikni kamaytiradi..

- dasturlash.uz

Dependency Injection 2

- When writing a complex Java Application, application classes should be as independent as possible od other Java Classes to increase the possibility to reuse test classes and to test them independently of other classes while unit testing
- Java da Murakkab dastur yozayotganimizda. Dasturdagi classlarni aloxida test qila olishimiz uchun ularni bir-biriga bog’lik bo’lmasligi kerak.

- dasturlash.uz

Dependency Injection 3

- There is dependency between the Lesson and the Professor

```
Piblic Class Lesson {
        private String name;
        private Professor professor;

        public Lesson (){
                professor = new Professor();
        }

}
```

Dependency Injection 4

- In an inversion of control scenario we do. Dependency Injection orqali ob’ektni set qilish :

```
public class Lesson{
        privte Professor professor;

        public Lesson(Professor professor){
                this.professor = professor;
        }
}
```

Dependency Injection 5

- We Have several ways to achieve DI in Spring
  - 1. Constructor injection
  - 2. Setter injection
  - 3. Fields injection
  - 4. Lookup method injection

DI - constructor 1

- We can inject the dependency by constructor
  - <constructor-arg> property is used for constructor injection.
- Constructor dependency allows injection
  - Primitive or Sting values
  - Objects
  - Collections
  - Set
- Example-> see in example project.

DI - constructor injection 1

```
public class Student {
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

```
<bean id="student" class="com.company.Student">
    <constructor-arg value="10" type="int"></constructor-arg>
    <constructor-arg value="Ali"></constructor-arg>
</bean>
```

- dasturlash.uz

DI - constructor injection 2

```
public class Lesson {
    private String name;
    private Professor professor;

    public Lesson(String name, Professor professor) {
        this.name = name;
        this.professor = professor;
    }
}
<bean id="lesson" class="com.company.Lesson">
    <constructor-arg value="Ali"></constructor-arg>
    <constructor-arg ref="professor"></constructor-arg>
</bean>

<bean id="professor" class="com.company.Professor">
    <constructor-arg value="10" type="int"></constructor-arg>
    <constructor-arg value="Ali"></constructor-arg>
</bean>
```

DI - constructor injection 3

- **Constructor by reference**

```
<bean id=“professor" class="com.company.Professor">
   <constructor-arg value="10" type="int"></constructor-arg>
   <constructor-arg value="Ali"></constructor-arg>
</bean>

        <!--  Constructor by reference    -->
<bean id="lesson2" class="com.company.Lesson">
   <constructor-arg value="Ali"></constructor-arg>
   <constructor-arg>
      <ref bean=“professor"></ref>
   </constructor-arg>
</bean>
```

```
public class Lesson {
    private String name;
    private Professor professor;

    public Lesson(String name, Professor professor) {
        this.name = name;
        this.professor = professor;
    }
}
```

DI - constructor injection 4

- *Constructor list*

```
public class Book {
    private List<String> headers;

    public Book(List<String> headers) {
        this.headers = headers;
    }
}
```

```
<bean id="book" class="com.company.Book">
      <constructor-arg>
          <list>
              <value>Java is a programming language</value>
              <value>Java is a Platform</value>
              <value>Java is an Island of Indonasia</value>
          </list>
      </constructor-arg>
  </bean>
```

DI - constructor injection 5

- *Constructor list reference*

```
<bean id="book2" class="com.company.Book">
    <constructor-arg type="double" value="500.0"></constructor-arg>
    <constructor-arg>
        <list>
            <ref bean="author1"></ref>
            <ref bean="author2"></ref>
        </list>
    </constructor-arg>
</bean>

<bean id="author1" class="com.company.Author"></bean>
<bean id="author2" class="com.company.Author"></bean>
```

```
public class Book {
    private double price;
    private List<Author> authors;

    public Book(double price, List<Author> authors) {
        this.price = price;
        this.authors = authors;
    }
}
```

DI - constructor injection 6

- *Constructor Map*

```
<bean id="q" class="com.company.Question">
   <constructor-arg>
      <map>
         <entry key="Java is a Programming Language"  value="Ajay Kumar"></entry>
         <entry key="Java is a Platform" value="John Smith"></entry>
         <entry key="Java is an Island" value="Raj Kumar"></entry>
      </map>
   </constructor-arg>
</bean>
```

```
public class Question {
    private Map<String, String> answers;
    public Question(Map<String, String> answers) {
        this.answers = answers;
    }
}
```

DI - constructor injection 7

- More Examples in spring_core_3_constructor_injection

Setter Injection

- We Have several ways to achieve DI in Spring
  - 1. Constructor injection
  - 2. Setter injection
  - 3. Fields injection
  - 4. Lookup method injection

Setter Injection 1

- We Can inject the dependency by setter method also
- The <property> sub element of <bean> is used for setter injection
- We can inject
  - Primitive and String-based values
  - Dependent object (contained object)
  - Collection values
  - Map values
- Example in project.
- Bog’liklikni biz setter metodi orqali ham inject qilsak bo’ladi. Yani ob’ektni setter metodi orqali ham inject qilishimiz mumkin.
- Setter metodi orqali inject qilish <bean> ning <property> tagi orqali qilinadi.
- Biz quyidagilarni type larni inject qilishimiz mumkin:
  - Primitive and String-based values
  - Dependent object (contained object)
  - Collection values
  - Map values

- dasturlash.uz

Setter Injection 2

```
public class Professor {
    private int id;
    private String name;
    private int age;

    // getter-setter

}
```

- *Setter injection primitive and String*

```
 <bean id="professor" class="com.company.Professor">
    <property name="id" value="1"/>
    <property name="name" value="Ali"/>
    <property name="age">
        <value>19</value>
    </property>
</bean>
```

- dasturlash.uz

Setter Injection 3

- *setter injection with reference*

```
public class Lesson {
    private String name;
    private Professor professor;
            // getter - setter
}
```

```
<bean name="lesson" class="com.company.Lesson">
    <property name="name" value="IT"/>
    <property name="professor" ref="professor"/>
</bean>
```

```
<bean id="professor" class="com.company.Professor">
     ....
 </bean>
```

Setter Injection 4

- *setter injection with list*

```
public class Book {
    private List<String> headers;
            // getter -setter
}
```

```
<bean name="book" class="com.company.Book">
      <property name="headers">
         <list>
            <value>Java is a programming language</value>
            <value>Java is a platform</value>
            <value>Java is an Island</value>
         </list>
      </property>
  </bean>
```

Setter Injection 5

- *setter injection with Map*

```
<bean name="question" class="com.company.Question">
      <property name="answers">
         <map>
            <entry key="1" value="A"></entry>
            <entry key="2" value="B"></entry>
         </map>
      </property>
  </bean>
```

```
public class Question {
    private Map<String, String> answers;
            // getter - setter
}
```

Setter Injection 5

- All examples on spring_core_4_setter_injection project

Field injection

- We Have several ways to achieve DI in Spring
  - 1. Constructor injection
  - 2. Setter injection
  - 3. Fields injection
  - 4. Lookup method injection

Field injection 1

- Field injection can be done by Autowiring In Spring
- Autowiring lets spring automatically inject bean dependency
- Autowiring feature of spring framework enables you to inject the object dependency implicitly.
- In internally uses setter or constructor injection

- Field Injection Spring da Autowiring orqali qilinadi.
- Autowiring bu Spring tamonidan kerakli bog’liklikni/ob’ektni aftomatik rafishda inject qilishga.
- Autowiring da dependency setter yoki constructor orqali inject qilinadi.
- Autowiring da faqat Object yokiy Referense tiplarni autowired qilsak bo’ladi.
- String yoki primitive tiplarni autowired qilib bo’lmaydi.

Field injection 2

- Advantage :
  - It required the less code because we do not need to write the code to inject the dependency explicitly.
- Disadvantage
  - Ca not be used for primitive or String values
  - No control of programmer

- Afzalliklari :
  - Kam qod yoziladi sababi ob’jectni set qiladigna kodlarni biz yozib o’tirmaymiz.
- Kamchiliklari:
  - Primitive va String lar uchun ishlatib bo’lmaydi.
  - Programmani konstol qilib bo’lmas emish.

- dasturlash.uz

Field injection - Autowiring modes 1

- No – It is the default autowiring mode. It means no autowiring bydefault.
- No – default xolatda ishlatiladi. Manosi Autowiring qilib o’tirmas shart emas degani.
- byName - The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
- byName – byName mode da ob’ekt o’zgaruvchining nomiga qarab set qilinadi. Bunda bean ni nomi va class dagi Autowired qilina yotgan ob’ekt nomi birxil bo’lishi kerak. byName – setter injectionni ishlatadi.
- byType – byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.
- bytype – byType mode ob’ektni turiga qarab inject qiladi. O’zgaruvchi va bean nomlari xar qil bo’lishi mumkin . Ob’ektni set qilish uchun setter injectdan foydalanadi.

Field injection - Autowiring modes 2

- Constructor - the constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.
- constructor – construktor mode da class dagi constructorga qaran kerakli dependensini inject qiladi. Bunda class dagi eng katta konstructor tanlanadi.

- autodetext – first tries to wire using autowire by constructor , if it does not work then tries byType. It is depricated in 3.
- autodetext - bunda oldin constructor orqali inject qilishga urinadi agat to’g’ri kelmasa byTye inject qiladi. Spring ni 3chi versiyasida man qilingan.

- dasturlash.uz

Autowiring - byName

- byName simply check the property name in class and tries to find the bean with matching name.
- Bunda oddigina class o’zgaruvchisi nomini tekshiradi. O’zgaruvchini nomi bilan birxil bo’lgan bean bo’lsa uni inject qiladi.

Autowiring - byName example

```
public class Lesson {
    private String name;
    private Professor professor;
}
```

- autowired professor byName

```
<bean id="lesson" class="com.company.Lesson" autowire="byName">
    <property name="name" value="IT"/>
</bean>

<bean id="professor" class="com.company.Professor">
    <property name="id" value="1"/>
    <property name="name" value="Ali"/>
</bean>
```

Autowiring - byType

- Simply checks the data type of the property in class and tries to find the bean with matching datatype.
  Bean id and reference name may be different.
- If you have multiple bean of one type it will not work. And asks to qualifier bean
- Bunda Class da inject qilinmoqchi bo'lgan class tipli bean ni topib inject qiladi. Bean ning nomi va class da o'zgaruvchilarning nomi xarxil bo'lishi mumkin.
- Agar ikkita birxil qitli bean lar bo'lsa axatolik beradi.

Autowiring - byType example

- autowired Author byType

```
public class Book {
   private Author author;
}
```

```
<bean id="authorAli" class="com.company.Author">
      <property name="name" value="Mir Ali"/>
   </bean>

<bean id="book" class="com.company.Book" autowire="byType"></bean>
```

Autowiring - constructor

- In This case spring container injects the dependency by highest parameterized constructor.
- Bunda kerakli ob’ektlar eng katta konstructorga qarab inject qilinadi.
- If you have 3 constructors on a class zero-arg, one –arg , two-arg
- Then injection will be performed by calling the two-arg constructor.

Autowiring - constructor example

- autowired Author constructor

```
public class Driver {
    private Car car;

    public Driver(Car car) {
        this.car = car;
    }
}
```

```
<bean id="car" class="com.company.Car">
      <property name="name" value="Matiz"/>
</bean>

<bean id="driver" class="com.company.Driver" autowire="constructor"></bean>
```

Autowiring - No

- no autowired in Manager

```
<bean id="manager" class="com.company.Manager" autowire="no">

</bean>
```

```
public class Manager {
    private Car car;
}
```

Lookup method injection

- We Have several ways to achieve DI in Spring
  - 1. Constructor injection
  - 2. Setter injection
  - 3. Fields injection
  - 4. Lookup method injection

Lookup method injection 1

- Suppose singleton bean A needs to use non-singleton (prototype) bean B, perhaps on each method invocation on A(getBeanB()), we expect to get new instance of bean B for every request.
- But The container only creates the singleton bean A once, and thus only gets one opportunity to set the properties.
- The container cannot provide bean A with a new instance of bean B every time one is needed. To get new new instance of bean B for every request we need to use lookup-method injection

Lookup method injection 2

```
class abstract Myclass {
 public String getUri(){
    // create a new instance of DefaultUri
    DefaultUri defaultUri = createDefaultUri();
    return "test"
 }

 protected abstract DefaultUri createDefaultUri();
}
```

```
<bean id="defaultUri" scope="prototype" class="DefaultUri">
</bean>

<bean id="myBean" class="com.MyClass"
        <lookup-method name="createDefaultUri" bean="defaultUri" />
</bean>
```

Autowiring – more all examples

- All example in project spring_core_5_authowiring

Scope

- Singleton – this scopes the bean definition to a single instance per SpringIoC container (default).
  - If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition.
- Prototype – This scopes a single bean definition to have any number of object instances.
  - If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made.
- Request – this scope a bean definition to an HTTP request. Only valid in the context of a web aware SpeingApplicationContext.
- Session – this. Scopes a bean definition to an HTTP session
- Global-session – this scopes a bean definition to a global HTTP session

```
<bean id="car" class="com.company.Car" scope="">
 </bean>
```

- dasturlash.uz

Bean Life Cyclce

- The life cycle of a Spring bean is easy to understand.
- When a bean is instantiated, it may be required to perform some initialization to get it into a usable state.
- Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.
- To define setup and teardown for a bean, we simply declare the <bean> with ***initmethod*** and/or ***destroy-method*** parameters

Bean Life Cyclce – init

- Thus, you can simply implement the above interface and initialization work can be done inside afterPropertiesSet() method as follows −

```
public class ExampleBean implements InitializingBean {
   public void afterPropertiesSet() {
      // do some initialization work
   }
}
```

- In the case of XML-based configuration metadata, you can use the **init-method** attribute to specify the name of the method that has a void no-argument signature. For example −

```
<bean id = "exampleBean" class = "examples.ExampleBean" init-method = "init"/>
```

```
public class ExampleBean {
   public void init() {
      // do some initialization work
   }
}
```

Bean Life Cyclce – destroy

- you can simply implement the above interface and finalization work can be done inside destroy() method as follows −

```
public class ExampleBean implements DisposableBean {
   public void destroy() {
      // do some destruction work
   }
}
```

- In the case of XML-based configuration metadata, you can use the destroy-method attribute to specify the name of the method that has a void no-argument signature. For example −

```
public class ExampleBean {
   public void destroy() {
      // do some destruction work
   }
}
```

```
<bean id = "exampleBean" class = "examples.ExampleBean" destroy-method = "destroy"/>
```

- dasturlash.uz

Bean Life Cyclce – init and destroy

- https://www.tutorialspoint.com/spring/spring_bean_life_cycle.htm
- Shu linkdan topasiz example chalarni.

BeanPostProccessor

- BeanPostProcessor interface defines callback methods that you can implement to provide your own instantiation logic.
- **BeanPostProcessor interface** is used for extending the functionality of framework if want to do any configuration Pre- and Post- bean initialization done by spring container.
- BeanPostProcessor class has two methods.
- 1) postProcessBeforeInitialization - as name clearly says that it's used to make sure required actions are taken before initialization. e.g. you want to load certain property file/read data from the remote source/service.
- 2) postProcessAfterInitialization - any thing that you want to do after initialization before bean reference is given to application.
- Example:

```
publiv  class HelloWorld implements BeanPostProcessor {

public Object postProcessBeforeInitialization (Object.., name..)
    // beforeInitialization
}

public Object postAfterInitialization(Object bean, String name){
   // afterInitialization
}
```

- dasturlash.uz

Sequence of the questioned methods in life cycle as follows :

- Sequence of the questioned methods in life cycle as follows :
- 1) BeanPostProcessor.postProcessBeforeInitialization()
- 2) init()
- 3) BeanPostProcessor.postProcessAfterInitialization()
- 4) destroy()

Annotation Based

- dasturlash.uz

- Annotation yordamida bean yaratish imkoni bo’lishi uchun xml ga quyidagini qo’shamiz:
- <context:component-scan base-package="com.company"/>

@Component

- @Component - is an annotation that allows Spring to automatically detect our custom beans.
- In other words, without having to write any explicit code.

- @Component yozilgan class dan Spring aftomatik rafishda bean yaratadi.
- Dexqoncha aytsa ortiqcha kod yozmasdan baen yaratish imkonini beradi.
- Spring will:
  - Scan our application for classes annotated with *@Component*
  - Instantiate them and inject any specified dependencies into them
  - Inject them wherever needed.

@Component example

- @Component
  public class IT {
  ……
  }

- Same as :
  <bean id="it" class="…….IT"/>

Other annotations

- **Spring Stereotype Annotations**
- *@Controller*, *@Service*, and *@Repository*
- They all provide the same function as *@Component*
- They are like *@Component* aliases with specialized uses.
- Yuqoridagi annotation larni bari birxil funksiyani bajaradi. Ammo mano jihatdan kelib chiqqan holda turli joylarda ishlatiladi.
- @Controller
  public class ControllerExample { }
- @Service
  public class ServiceExample { }
- @Repository
  public class RepositoryExample { }
  @Component
  public class ComponentExample { }

@Autowired

- dasturlash.uz

@Autowired

- @Autowired annotation can be used to autowire bean on the setter method, constructor, a property or methods with arbitrary (ixtiyoriy) names.
- Tries find bean by Type.
- If there are several types uses byName.
- @Autowired annotatsiya kerakli beanlarni o’zi topib inject qilish uchun ishlatiladi. @Autowired settor metod, constructor yoki o’zgaruvchilar uchu nishlatilishi mumkin.
- @Autowired birinchi type bo’yicha qidiradi agar.
- Agar bitta type dan birnechta bean bo’lsa name bo’yicha qidiradi.

@Autowired - example

- @Autowired
  private Student studentJon;

- @Autowired
  public void setStudent(Student student) {
  this.student = student;
  }

- @Autowired
  public Lesson(Student student){ .... }

@ComponentScan

@ComponentScan 1

- **Spring uses the *@ComponentScan* annotation to actually gather them all into its *ApplicationContext*.**

- <context:component-scan base-package="com.company"/>
- @ComponentScan({"com.baeldung.component.inscope", "com.baeldung.component.scannedscope"})

```
@Configuration
@ComponentScan(basePackages = "com.company")
public class Config {
}
```

- dasturlash.uz

@ComponentScan 2

```
public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
        MainController mainController = (MainController) context.getBean("mainController");
        CardService cardService = (CardService) context.getBean("cardService");
        UserMenu userMenu = (UserMenu) context.getBean("userMenu");
        mainController.start();

    }
```