Spring Framework Masterclass Notes

A diagram of a company

Description automatically generated

What is the Spring Framework?

* Spring is a dependency injection framework

What is dependency injection?

What is dependency?

* A requirement or need that a class relies on to perform certain functions

Tight coupling – when a class directly instantiates a specific dependency (BAD CODE!!!)

Tight Coupling example:

public class ComplexBusinessService {

SortAlgorithm sortAlgorithm = new BubbleSortAlgorithm();

}

public class BubbleSortAlgorithm implements SortAlgorithm {…

Lose Coupling example (GOOD CODE)

public class ComplexBusinessService {

SortAlgorithm sortAlgorithm; //= new BubbleSortAlgorithm;

public ComplexBusinessService(SortAlgorithm sortAlgorithm) {

this.sortAlgorithm = sortAlgorithm;

}

}

SortAlgorithm sortAlgorithm = new BubbleSortAlgorithm();

ComplexBusinessService businessService = new ComplexBusinessService(sortAlgorithm);

For Spring Framework, programmers need to tell Spring Framework what to do:

Using @Component – Spring will start managing instances of a class for you

Using @Autowired – Spring will start looking at the different instances that it manages and find a match. For the example, internally, Spring would find a match of the sortAlgorithm and create an instance of the BubbleSortAlgorithm, then pass it to the constructor of the ComplexBusinessService to create an instance of it.

@Component

public class ComplexBusinessService {

@Autowired

SortAlgorithm sortAlgorithm; //= new BubbleSortAlgorithm;

public ComplexBusinessService(SortAlgorithm sortAlgorithm) {

this.sortAlgorithm = sortAlgorithm;

}

}

@Component

public class BubbleSortAlgorithm implements SortAlgorithm {…

Terminology:

*Beans* – different objects that are managed by Spring framework

*Autowiring* – the process in which Spring identifies the dependencies, identifies the matches for the dependencies, and populates them

*Dependency Injection* – a fundamental aspect of the Spring framework, through which the Spring container “injects” objects into other objects or “dependencies”. Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

*Inversion of Control* – taking the control from the classes and giving it to the framework

*IOC Container* – generic terminology that is implementing Inversion of Control

*Application Context* – where all the core logic of Spring Framework happens. Where all the beans are defined.

BinarySearchImpl binarySearch = new BinarySearchImpl(new QuickSearchAlgorithm());

The “new QuickSearchAlgorthithm()” that is passed to the BinarySearchImpl in the above example is a Bean, and we want a framework that can manage them, and then manage and populate all of the dependencies.

Spring Framework allows us to manage Beans and ‘wire’ in the dependencies

Spring using the constructor in a component class to instantiate an instance of a bean is call Constructor Injection

Can also generate a Setter instead of a Constructor in a class. Spring will generate the autowiring the same way using the setter as if there were nothing at all. (No setter or constructor)

If there are mandatory dependencies within a class, it should use Constructor injection. If there are optional dependencies, then you don’t need a constructor – it can use a setter or nothing. (old recommendation)

With Spring, using @Autowired allows people to get away with not using constructors, or just using setters. It would become difficult to maintain constructors if additional dependencies are added to a class and the numbers get upwards of 15 or 20.

In Eclipse, Run As Maven Project and for Goals -> dependency:tree to show all dependencies printed out.

Spring Boot – one of the most popular frameworks for developing microservices: start-up projects, auto-configuration, actuator. Makes developing applications simple.

Spring Cloud – build cloud native applications. Cloud-enable services build with Spring Boot.

Spring Data – consistent data access. Connect to different types of databases (SQL, NoSQL, etc.)

Spring Integration – addresses problems with application integration. Enterprise Application Integration Patterns (book) – Spring Integration integrates the patterns that are recommended in that book

Spring Batch – Batch applications – enables developing batch application easily. Provides opportunities to restart applications when they fail and also being able to track down what failed to very minute details

Spring Security – provides solutions for securing applications, REST services, etc.

Spring HATEOAS – HATEOAS compatible services allow developers to return data as well as links to users to help them get additional details

Spring Sessions, Spring Mobile, Spring Web Services, etc. – There are many other projects that Spring is involved with

Spring Popularity:

1. Spring enables writing testable code – easily integrates with Mockito and JUnit
2. There is no Plumbing Code – no need for unnecessary additional exception handling for functions
3. Flexible Architecture – Spring is very modular. There are spring modules and spring projects for very specific purposes.
4. Staying current with new trends – Spring is able to come up with new projects that helps it stay relevant with recent evolution of microservices and other trends

Ecliipse has save actions such as automatically format code, cleaning code, convert all code to blocks (such as if/else statements), etc.

Maven Build Life Cycle:

* Validate
* Compile
* Test
* Package
* Integration Test
* Verify
* Install
* Deploy

When working with Maven, there is the local repository, a.k.a. the Local System, and the Remote Maven (central) repository that stores all versions of all dependencies (i.e. JUnit 4.2, 4.3, 4.4, etc.)

mvn compile – compiles code

mvn clean compile – cleans the project and removes all files generated in the previous build

mvn test compile – compiles both source files and test files

mvn install vs mvn deploy

* install installs project to local repository, while deploy will push it out if you want to deploy it to an external server for others to see

help:effective-pom -> prints out all content from different pom(Project Object Model) files

***Spring Level 2 – Spring Framework In Depth***

1. Autowiring Types and Qualifiers
2. Bean Scope & Life Cycle
3. IOC Container & Application Context
4. XML & Java Application Contexts
5. Component Scan
6. External Properties
7. Container/Context & Dependency Injection (CDI)

Dependencies

@Component

public class ToDoController {

@Autowired

ToDoBusinessService businessService;

}

To fix the error for having two components for autowiring, you can add Primary to one of the components, or you can rename the variable of the implementation of the autowired object in the class. Example below:

@Component

public class BinarySearchImpl {

@Autowired

private SortAlgorithm bubbleSortAlgorithm; //rename this to bubbleSortAlgorithm instead of just “sortAlgorithm”

}

\*\*\*Even if you rename the object, if there is another component that has @Primary above it, @Primary gets priority over the named object\*\*\*

@Qualifier([qualifier name]) – can be used to specifier which candidate to use for the dependency

Bean Scope:

Default – Singleton

1. Singleton – One instance per Spring Context
2. Prototype – New bean whenever requested
3. Request – One bean per HTTP request
4. Session – One bean per HTTP session

Above class definition, add @Scope() to tell Spring what the scope should be (i.e. @Scope(“prototype”))

Singleton is the default, but to specify it, use ConfigurableBeanFactory.SCOPE\_PROTOTYPE or ConfigurableBeanFactory.SCOPE\_SINGLETON

A computer screen shot of a computer screen

Description automatically generated

Whenever you are trying to get a bean that is NOT a prototype and one of the dependencies *is* a prototype, you should add a proxy to it if you want to ensure that a new instance of the dependency is created each time.

If the java application file is outside a package folder, in order to get Spring Framework to scan the files and pick up the components (beans), we can add the following:

@ComponentScan(“com.spring.basics.componentscan”) <- as an example, this will cause Spring to scan the componentscan package folder

The lifecycle of a bean is controller by the Spring IOC container.

Actions that you want to have performed before a bean is destroyed can be defined in post construct. (@PostConstruct)

CDI – javax annotations for injection

* Java EE Dependency Injection Standard (JSR-330)
* Spring supports most annotations
  + @Inject (@Autowired)
  + @Named (@Component & @Qualifier)
  + @Singleton (Defines scope of Singleton)

Disambiguation

* IOC Container
  + Inversion of Control container – manages the beans, creates instances of services, creates beans for controllers, autowires the service beans to the controller beans, etc.
  + Generic concept – not really framework specific
* Application Context
  + Bean Factory++
    - Spring’s AOP Features
    - I18n Capabilities (Internationalization)
    - WebApplicationContext for web applications, etc.
      * Spring recommends to use WebApplicationContext except when memory is limited
* Bean Factory
  + Actual container which instantiates, configures, and manages a number of beans.

@RestController

public class WelcomeController {

private WelcomeService service = new WelcomeService(); //tightly coupled

@RequestMapping(“/welcome”)

public String welcome() {

return service.retrieveWelcomeMessage();

}

}

When Spring is used:

@Component

public class WelcomeService {

//blah blah blah

}

@RestController

public class WelcomeController {

@Autowired

Private WelcomeService service;

@RequestMapping(“/welcome”)

public String welcome() {

return service.retrieveWelcomeMessage();

}

}

Component Annotations

* @Component – Generic component

//the following three are specific to the application layer (repository=data, service=services/business, controller=controllers/UI)

* @Repository – encapsulating storage, retrieval, and search behavior typically from a relational database
* @Service – Business Service Façade
* @Controller – Controller in MVC pattern

***Introduction to JUnit***

class Class1 {

method1

method2

method3

method4

…

}

What is Unit Testing?

* Unit testing would be writing individual tests for each of the methods in the class above, groups of the methods, and/or the class itself.

What is JUnit?

* JUnit is a framework which will help to call the specific methods and check whether or not the output is as expected.

JUnit helps to simplify automated testing and allows for easier/better CI whenever a change is made to the code.

JUnit Assert functions: assertEquals(), assertTrue(), assertFalse(), assertNull(), assertNotNull(), assertArrayEquals()

Important JUnit Annotations:

* @Before – run something before every test. Note that in IntelliJ, using junit.juniper.api, there is only @BeforeEach and @BeforeAll

***Spring Boot***

The most important goal of Spring Boot is to *enable building production ready applications quickly*. The second most important goal is to *provide common non-functional features:*

* *Embedded servers*
* *Metrics*
* *Health checks*
* *Externalized configuration*

What Spring Boot is NOT – Spring Boot is not a code generating framework. There is ZERO code generation. Also, Spring Boot is neither an application server nor a web server.

Features:

* Quick Starter Projects with Auto Configuration
  + Web
  + JPA
* Embedded Servers
  + Tomcat
  + Jetty
  + Undertow
* Production-ready features
  + Metrics and health checks
  + Externalized configuration

Add logging by opening application.properties and putting in this:

logging.level.org.springframework = DEBUG

Spring Boot Auto-Configuration: attempts to automatically configure your Spring application based on the jar dependencies that you have added. It helps us automatically configure a Spring application based on the dependencies that are present on the classpath.

What is the core problem that Spring Framework solves?

* The core problem that Spring Framework solves is testability.
* The most important feature of the Spring Framework is Dependency Injection.
* At the core of all Spring Modules are Dependency Injection and/or IOC (Inversion of Control)
  + These allow applications to be developed in a loosely coupled way
  + Loosely coupled application can be easily unit tested
* Spring Framework also solved Duplication/Plumbing Code (try/catch, exception handling, etc.)
* Good integration with other frameworks.

Spring MVC Framework a decoupled way of developing web applications. Simple concepts like Dispatcher Servlet, ModelAndView, and View Resolver make it easy to develop web applications.

Spring Boot provides auto-configuration to remove a lot of the grunt work from Spring MVC, servlets, Hibernate/JPA, etc. The ideas are:

* How about auto-configuring a Data Source if a Hibernate jar file is on the classpath?
* How about auto-configuring a Dispatcher Servlet if a Spring MVC jar file is on the classpath?

Spring Boot looks at the frameworks available on the classpath and the existing configuration for the application. Based on these, Spring Boot provides the basic configuration needed to configure the application with these frameworks

Spring Boot Starter Projects are all built around well known patterns.

Other goals of Spring Boot are to provide a few monitoring features:

* spring-boot-starter-actuator: to use advanced features like monitoring & tracing to your application right out of the box
* spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat: allows you to pick your specific choice of Embedded Servlet Container
* spring-boot-starter-logging: for logging using logback
* spring-boot-starter-log4j2: logging using Log4j2

Spring Boot aims to enable production ready applications in a short amount of time:

* Actuator: Enables Advanced Monitoring and Tracing of applications
* Embedded Server Integrations: since the server is integrated into the application, there is no need to have a separate application server installed on the server
* Default Error Handling

Additional Spring Boot Starter Projects

* spring-boot-starter-web-services: SOAP Web Services
* spring-boot-starter-web: Web & RESTful applications
* spring-boot-starter-test: Unit testing and Integration Testing
* spring-boot-starter-jdbc: Traditional JDBC (Java Database Connectivity)
* spring-boot-starter-hateoas: Add HATEOAS (Hypermedia as the Engine of Application State) features to your services
* spring-boot-starter-security: Authentication and Authorization using Spring Security
* spring-boot-starter-data-jpa: Spring Data JPA (Java Persistence API) with Hibernate
* spring-boot-starter-cache: Enabling Spring Framework’s caching support
* spring-boot-starter-data-rest: Expose simple REST services using Spring Data REST

Pointcut – the types of methods to be intercepted by AOP @Before call

Advice = logic within AOP Aspect class method that runs before/during method interceptions

Aspect = a combination of the point cut/JoinPoint and Advice

JoinPoint = specific interception instance of a method

Weaving and Weaver = the process and framework of implementing AOP around application methods

Application Properties needed for H2 Web Browser Application:

# Enabling H2 Console

spring.h2.console.enabled=true

# Custom H2 Console URL

spring.h2.console.path=/h2-console

#Database source

spring.datasource.url=jdbc:h2:mem:testdb

Spring JDBC Example:

(main application file)

@Repository

public class PersonJdbcDAO {

@Autowired

JdbcTemplate jdbcTemplate;

//select \* from person

public List<Person> findAll() {

return jdbcTemplate.query("select \* from person", new BeanPropertyRowMapper<Person>(Person.class));

}

}

(sql file)

create table person

(

id integer not null,

name varchar(255) not null,

location varchar(255),

birth\_date timestamp,

primary key(id)

);

INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH\_DATE)

VALUES(10001, 'Ranga', 'Hyderabad', CURRENT\_TIMESTAMP());

INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH\_DATE)

VALUES(10002, 'James', 'New York', CURRENT\_TIMESTAMP());

INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH\_DATE)

VALUES(10003, 'Peter', 'Tampa', CURRENT\_TIMESTAMP());

INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH\_DATE)

VALUES(10004, 'John', 'Los Angeles', CURRENT\_TIMESTAMP());

INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH\_DATE)

VALUES(10005, 'Samuel', 'San Francisco', CURRENT\_TIMESTAMP());

INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH\_DATE)

VALUES(10006, 'Matthew', 'Chicago', CURRENT\_TIMESTAMP());

JDBC Example:

@Override

public List<Todo> retrieveTodos(String user) throws SQLException {

Connection connection = datasource.getConnection();

PerparedStatement st = connection.preparedStatement(“SELECT \* FROM TODO where user = ?”);

st.setString(1, user);

ResultSet resultSet = st.executeQuery();

List<Todo> todos = new ArrayList<>();

while(resultSet.next()) {

Todo todo = new Todo(resultSet.getInt(“id”),

resultSet.getString(“user”),

resultSet.getString(“desc”),

resultSet.getTimestamp(“target\_date”),

resultSet.getBoolean(“is\_done”));

todos.add(todo);

}

st.close();

connection.close();

return todos;

}

Adding logging.level.root=debug to the application.properties file will give you an AUTO-CONFIGURATION REPORT in the console when the program runs.

JDBCTemplate

When Spring boot runs the auto configuration, it goes through all the classes on the classpath, so when it sees the following, it auto configures everything:

* DataSourceAutoConfiguration: javax.sql.DataSource
* DataSourceTransactionManagerAutoConfiguration: org.springframework.jdbc.core.JdbcTemplate
* H2ConsoleAutoConfiguration: org.h2.server.web.WebServlet (only configured if spring.h2.console.enabled=true in application.properties file)
* JdbcTemplateAutoConfiguration: javax.sql.DataSource – org.springframework.jdbc.core.JdbcTemplate
* JdbcTemplateAutoConfiguration.JdbcTemplateConfiguration#jdbcTemplate: org.springframework.jdbc.core.JdbcOperations

In JPA, we would define something called an entity and define the relationships between entities. With JPA, map the fields within an entity to the columns in the table.

Once the entities and the relationships have been defined, the JPA implementation will take care of identifying the entities and creating the correct queries for you based on the operations you would want to perform.

So, with JPA, the responsibility of writing the queries shifts from the developer to the framework.

Hibernate is the most popular implementation of JPA – so Hibernate implements JPA

The entity manager executes everything that is needed for executing queries:

* Update and Insert -> entityManager.merge(entity);
* Find by Id -> entityManager.find(entity.class, [primary key])
* Delete by id -> pass id to method, create an instance of entity by calling findById, then execute entityManager.remove(entity)
* Find All -> TypedQuery<entity> namedQuery = entityManager.createNamedQuery(“name of query”, entity.class); return namedQuery.getResultList();

Connect to MySQL database:

pom.xml

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

application.properties

spring.jpa.hibernate.ddl-auto=none (can be set to ‘create’ if you want hibernate to create schema for you)

spring.datasource.url=jdbc:mysql://localhost:3306/person\_example

spring.datasource.username=personuser

spring.datasource.password=YOUR\_PASSWORD

Java EE documentation: <https://docs.oracle.com/javaee/7/index.html>

Java Servlet is a very simple Java class that can take a request in and gives a response back as an output.

To pass in parameters to a URL, add a ?, then set the parameter values:

* Example – <http://localhost:8080?name=Matsumoto>

If more than one parameter, separate parameters by an ‘&’ symbol

* Example – <http://localhost:8080?name=Matsumoto&password=somepassword>

Dispatcher Servlet -> Front Controller

Login Controller -> Handler

login.jsp -> view

View Resolver

login -> /WEB-INF/views/login.jsp

Spring MVC Architecture:

A diagram of a model

Description automatically generated

Eclipse shortcuts:

Ctrl + Space -> get suggestions from Eclipse (import classes, name instances, constructor options, etc.)

Ctrl + 1 -> bring up options to fix errors and helps solve problems (rename classes, defining variables, assigning values, etc.)

Ctrl + Shift + R (and T) -> Search in all files/resources in a project

F3 -> Go to Declaration

F4 -> Type Hierarchy

Ctrl + Shift + F -> Automatically formats code

Ctrl + Shift + O -> Cleans up unused items

Ctrl + Shift + L -> brings up a list of all shortcuts available