Advanced Java

Agenda

- MVC architecture
- Maven
- Spring fundamentals
    - Spring Features
    - Spring Configuration -- XML and Annoatation
    - Spring framework Overview
    - Spring limitations
- Spring Boot
    - Introduction

Application architectures

Model-View architecture

- Also called as Model-1 architecture.
- Application is divided in two major parts.
    - Model: Data handling and Business logic -- Java bean.
    - View: Presentation/appearence of the data -- JSP.
- Java beans are tightly coupled with JSP pages (jsp:useBean). Also a JSP page may be tightly coupled with other JSP pages (e.g. href="...", action="..."). Any changes into bean or jsp will lead to changes in multiple other components.
- This architecture is suitable for small applications.

Model-View-Controller architecture

- Also called as Model-2 architecture.
- Application is divided in three major parts.
    - Model: Data handling and Business logic -- Java bean.

    - View: Presentation/appearence of the data -- JSP.
    - Controller: Handles communication/navigation between views and models.
- Models and views are loosely coupled with each other. Their navigation is centrally controlled by controller layer.
    - View1 --> Controller --> View2
    - View1 --> Controller --> Java Bean and View2
- This architecture is suitable for bigger applications.
- Typically controller is implemented as a servlet that "forwards" the request to the next component.
- There are popular frameworks which implements MVC pattern (e.g. JSF, Spring MVC, Struts MVC). Spring MVC has predefined controller called as "DispatcherServlet".

Maven

- Not in DMC Syllabus
- Maven is Java Build Tool.
- Maven does following.
    - Download dependencies (jar) from central/remote repository into local repository.
    - Compile source code.
    - Package compiled code into JAR or WAR files.
    - Install/Deploy the packaged code.
- Maven can be installed on any OS as a command line utility -- mvn.
- All modern Java IDEs have built-in support for Maven.

Maven repositories

- Local repository
    - Maven downloads all dependencies into local repository.
    - Local repository location.
        - Linux: /home/username/.m2
        - Windows: C:/Users/username/.m2
    - Jars from this path are added into project CLASSPATH.
- Central repository
    - Provided by Maven community.

    - https://mvnrepository.com/repos/central
    - Hosts all standard jars published by respective vendors.
    - If jars not present in local repository, they will be downloaded from central repository.
- Remote repository
    - Hosted on a web-server to maintain organization specific dependencies.
    - Similar to central repository.
    - Need to configure in Maven pom.xml or settings.

pom.xml

- pom.xml is heart of Maven.
- POM - Project Object Model.
- It is located into root of Maven project.
- pom.xml holds build details of the project.
    - profiles
    - dependencies
    - build plugins

dependencies

-
- Third-party jars to be added into the project.
- Dependency is uniquely identified by the groupId, artifactId and version.
- All jars auto-downloaded from Maven repository and added into project CLASSPATH.

profiles

-
- Maven enable building projects in different configurations like dev, test, production, etc.
- It enables doing changes in build steps/config for certain profile.

build plugins

- Allows to add user-defined actions in the build process.
- Implemented by frameworks for customization in build process.

Maven build process

- Maven build process can be understood with build life-cycle, build phases and goals.

Build life cycle

- Maven follows certain build life cycle while building any project.
- Maven build life cycles
  - default: Build the project as per given build phase.
  - clean: Deletes all generated files.

Build phases

- A build life cycle is divided into sequence of multiple build phases.
- Important build phases in default build life cycle.
  - validate: Check project pom.xml syntax. Downloads all dependencies (if not present in local repository).
  - compile: Compile source code of the project.
  - test: Execute given unit tests against the compiled source code using a suitable unit testing framework.
  - package: Pack the generated files into given package (jar or war).
  - install: Copy the package into the local repository. It can be used in other projects on local machine.
  - deploy: Copy the final package to the remote repository for sharing with other developers and projects.

Parent POM

- Project pom.xml can be inherited from parent pom.xml using .
- By default all details (like properties, build, etc) of parent POM are inherited into child POM.
- Child POM can override the changes and can have additional details.
- The final POM (after inheritance) including all settings is referred as Effective POM.

Reference

- Article: http://tutorials.jenkov.com/maven/maven-tutorial.html
- YouTube Video: https://youtu.be/lMXBrlVFYA0

Spring

- Initially developed by "Rod Johnson" in 2003.
- Early versions of spring were relying on XML config (till Java 1.4).
- Later versions (3.0+) added support of annotation config (from Java 5.0+).
- Spring 3.0 became popular spring version.

Why Spring

- Java is Simple.
  - Language syntax is easier than C, C++, ...
  - Built in classes for collections, file IO, ...
  - Java development includes Java coding, Unit testing, Integration testing, Configurations, ...
- Spring is lightweight comprehensive framework that simplifies Java development.
  - "light-weight" - basic version of Spring framework is around 2 MB.
  - "comprehensive" - dependency injection
  - "Simplifies Java development" - Ready-made support/wrappers for different Java technologies, Unit testing, ...
- Good programming practices based on interfaces and POJOs.
- Inversion of Control - Dependency injection.
  - OOP -- Object Oriented Programming -- Composition
  - Outer object --> Inner object(s)
    - Car <>-- Engine
    - Car <>-- Wheels

```
// ...
Car c = new Car();
c.setEngine(e);
```

```
c.setWheels(w);
c.setChasis(ch);
// ...
c.drive();
```

    - Spring automates object creation as well as initialization.

```
Car c = ctx.getBean(Car.class);
c.drive();
```

    - Containers
        - Applet container -- JRE in browser to execute Applet life cycle.
        - Web conatiner -- Component of web-server to execute Servlet/JSP life cycle.
        - EJB container -- Component of application-server to execute EJB life cycle.
        - Spring container -- Part of Spring framework to execute/manage Spring bean life cycle.
            - Spring bean -- Simple Java objects created by Spring.
            - Spring container is also called as IoC container.
    - Inversion-of-Control
        - OOP (Bottom-up) vs POP (Top-down)
        - Manual Object Initialization vs DI Object Initialization (Inverted)
- Test driven approach: Easy testing support.
    - Unit testing/Mocking support in out-of-box.
    - Beans can be tested independently.
- Extensive but modular and flexible.
    - Extensive: Huge -- so many modules, sub-projects, wide spectrum
    - Modular: Separate sub-projects/modules are available
    - Flexible: Can use only modules you need on top of Spring core
- Smooth integration with existing technologies like JDBC, JNDI, ORM, Mailing, etc.

- Eliminate boiler-plate code - ORM, JDBC, Security, etc.
  - JDBC
    - Load & register class (managed by spring)
    - Create connection (managed by spring)
    - Create statement (managed by spring)
    - Execute the query (managed by spring)
    - Process the result (ResultSet) -- supply SQL statement, parameters and process result.
    - Close all (managed by spring)
  - Hibernate
    - Hibernate configuration (.cfg.xml or coding) (managed by spring)
    - Create SessionFactory (managed by spring)
    - Create session (managed by spring)
    - transaction management (managed by spring)
    - CRUD operations or Query execution -- user-defined
    - Cleanup (managed by spring)
- Unified transaction management (local & distributed/global) -- @Transactional
  - Local transactions: Within same database
  - Global transactions - JTA: Across the databases
- Easier Java EE development through Spring Web MVC (Pull) and Web sockets (Push).
- Consistent and "readable" unchecked exceptions - wraps technology-specific exceptions.

Spring - Core feature

- Dependency injection
- Spring IoC container

Spring Dependency Injection - XML Config (spring01)

1. Ensure that you are connected to stable internet before you proceed.
2. Create new Maven project with No archetype. Give group id (com.sunbeam) and project name (spring01). Select destination location.
3. Change Java version to 11 and add dependencies in pom.xml and Save the file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.sunbeam</groupId>
<artifactId>demo01</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>demo01</name>
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.30</version>
        </dependency>
    </dependencies>
</project>
```

4. In src/main/java -> com.sunbeam package, add an interface Box and a POJO class BoxImpl with length, breadth and height fields. Add appropriate business logic methods (e.g. calcVolume()).
5. In src/main/resources create new XML file beans.xml. Copy following XSD namespace <bean ...>. Add bean definitions/configurations in that XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd">

    <bean id="b1" class="com.sunbeam.BoxImpl">
        <property name="length" value="5"/>
        <property name="breadth" value="4"/>
```

```
        <property name="height" value="3"/>
    </bean>

    <bean id="b2" class="com.sunbeam.BoxImpl">
        <constructor-arg index="0" value="10"/>
        <constructor-arg index="1"  value="8"/>
        <constructor-arg index="2" value="6"/>
    </bean>
</beans>
```

6. In src/main/java -> com.sunbeam package, in a Main class main() method.

```
public static void main(String[] args) {
    ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
    BoxImpl b1 = (BoxImpl) ctx.getBean("b1");
    System.out.println(b1.calcVolume());
    BoxImpl b2 = (BoxImpl) ctx.getBean("b2");
    System.out.println(b2.calcVolume());
    ctx.close();
}
```

7. Run the Main class (▷ button).

Spring Dependency Injection - Java Config (spring02)

1. Ensure that you are connected to stable internet before you proceed.
2. Create new Maven project with No archetype. Give group id (com.sunbeam) and project name (spring01). Select destination location.
3. Change Java version to 11 and add dependencies in pom.xml and Save the file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
```

```
<groupId>com.sunbeam</groupId>
<artifactId>demo02</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>demo02</name>
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.30</version>
        </dependency>
    </dependencies>
</project>
```

4. In src/main/java -> com.sunbeam package, add an interface Box and a POJO class BoxImpl with length, breadth and height fields. Add appropriate business logic methods (e.g. calcVolume()).
5. In src/main/java -> com.sunbeam package, add a config class AppConfig.

```
@Configuration
public class AppConfig {
    @Bean // setter based di
    public BoxImpl b1() {
        BoxImpl b = new BoxImpl();
        b.setLength(5);
        b.setBreadth(4);
        b.setHeight(3);
        return b;
    }
    @Bean // constructor based di
    public BoxImpl b2() {
        BoxImpl b = new BoxImpl(10, 8, 6);
```

```
        return b;
    }
}
```

6. In src/main/java -> com.sunbeam package, in a Main class main() method.

```
public static void main(String[] args) {
    AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
    BoxImpl b1 = (BoxImpl) ctx.getBean("b1");
    System.out.println(b1.calcVolume());
    BoxImpl b2 = (BoxImpl) ctx.getBean("b2");
    System.out.println(b2.calcVolume());
    ctx.close();
}
```

7. Run the Main class (▷ button).

Spring Beans

- Spring beans are Java class objects instantiated by Spring container.
- Beans are created and initialized as per user-defined configuration (bean definition).
- Spring bean classes are simple Java POJO classes with one or more business logic method.

Spring configuration

- Spring allows configuring beans in different ways.
    - XML config
    - Java config
    - Mixed config

XML config

- Early versions of Spring (before Spring 2.5) support only XML configuration.
- Spring beans are declared into Spring bean configuration file.
- These files follow fixed syntax/grammer (in terms of XSD).
- XML configuration allows various config
    - Initializing properties (primitive types)
    - Initializing dependency beans
    - Initializing collections (list, maps, etc)
    - Defining bean initialization and de-initialization
    - Defining bean scopes
    - etc.
- Typically ClassPathXmlApplicationContext reads the bean config file and create Spring beans at runtime.

Java config

- Supported from Spring 2.5 onwards and makes extensive use of annotations.
- The bean creation is encapsulated in @Configuration class's @Bean methods.
- Java config also allows defining beans, their dependencies, scopes, initialization, etc.
- An application may have multiple configuration classes. A @Configuration class may import another using @Import.
- Spring Boot recommends java config. In Spring Boot, by default it detects all @Configuration classes in the main package.
- Typically AnnotationConfigApplicationContext gets config from @Configuration classes and create Spring beans at runtime.
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Configuration.html

Mixed config

- XML configuration can process annotations like @Autowired, @Qualifier, @PostConstruct and @PreDestroy using `<context:annotation-driven/>`. It can also detect stereo-type annotations using `<context:component-scan>`.
- In Java config, @Configuration class may import XML config using @ImportResource.

ApplicationContext

- Spring container is created while ApplicationContext is created.
- ApplicationContext enable accessing Spring container features in the application.

- ApplicationContext is an interface and have various implementations for different scenarios/applications.
- ApplicationContext
  - ClassPathXmlApplicationContext
    - ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml"); // beans.xml in resources or src dir.
  - FileSystemXmlApplicationContext
    - FileSystemXmlApplicationContext ctx = new FileSystemXmlApplicationContext("/home/nilesh/beans.xml"); // beans.xml is in some folder.
  - AnnotationConfigApplicationContext
    - AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class); // AppConfig is @Configuration class
  - WebApplicationContext (interface)
    - XmlWebApplicationContext
    - AnnotationConfigWebApplicationContext
- ApplicationContext (Spring Container) create all (singleton) beans when application context is loaded.
- ApplicationContext interface itself is inherited from BeanFactory interface.
  - It provides basic facility of Beans loading and initialization.
- Typical spring applications have single ApplicationContext.

Dependency Injection

- https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-collaborators
- Object dependencies are typically defined as
  - Constructor arguments
  - Property (Setter) arguments
  - Factory method arguments
- The spring container inject them into beans immediately after bean creation.
- Advantages of Dependency Injection
  - Cleaner code
  - Loose coupling
  - Location transparency (of dependency)
  - Easier testing/mocking
- There are two major varients of dependency injection.
  - Constructor based DI
  - Setter based DI

Dependency Resolution Process

- The ApplicationContext is created and initialized with configuration metadata (XML or annotations) that describes all the beans.
- For each bean, its dependencies are expressed in the form of properties or constructor arguments. These dependencies are injected when the bean is actually created.
- Each property or constructor argument can be a value (primitive type) or reference of another spring bean.
- The value is converted from its specified format to the actual type of that property or constructor argument.

Spring vs Spring Boot

Spring - Hurdles

- Heavy configuration
  - XML configuration
  - Java (annotation) configuration
  - Mixed configuration
- Module versioning/compatibility
  - Spring & Spring-Security version
  - Hibernate & Cache version
  - ...
- Application deployment
  - Web-server deployment
  - Containerisation - Docker
  - Complex Microservice development

Spring Boot

- Spring Boot is a Framework from "The Spring Team" to ease the bootstrapping and development of new Spring Applications.
  - Framework (Not a completely new framework -- rather it is abstraction on existing spring frameworks)
  - Ease bootstrapping - Quick Start (Rapid Application Development)
  - New applications - Not good choice for legacy applications
- Abstraction/integration/simplification over existing spring framework/modules.
  - Spring core

    - Spring Web MVC
    - Spring Security
    - Spring ORM
    - ...
- NOT replacement or performance improvement of Spring framework.
    - Not replacement, but it is abstraction.
    - Underlying same Spring framework is working -- with same speed.
- Spring Boot = Spring framework + Embedded web-server + Auto-configuration - XML configuration - Jar conflicts

Why Spring Boot

- Provide a radically faster and widely accessible "Quick Start".
- Very easy to develop production grade applications.
- Reduce development time drastically.
- Increase productivity.
    - Spring Boot CLI
    - Spring Initializr (https://start.spring.io/)
    - IDE support - Eclipse based Spring Tools IDE (STS)
    - Build tools - Maven/Gradle support
- Managing versions of dependencies with starter projects.
    - Set of convenient dependency descriptors
    - Hierarchially arranged so that minimal dependencies to be added in application
    - e.g. spring-boot-starter-web
        - spring-web
        - spring-webmvc
        - spring-boot-starter
            - spring-core
            - spring-context
            - spring-boot-autoconfigure
            - spring-boot-starter-logging
                - log4j-to-slf4j
                - jul-to-slf4j

        - spring-boot-starter-json
            - jackson-databind
        - spring-boot-starter-tomcat
            - tomcat-embed-core
            - tomcat-embed-websocket
            - jakarta.el
        - hibernate-validator
- No extra code generation (boiler-plate is pre-implemented) and No XML config.
    - Minimal configuration required.
    - Ready to use in-memory/embedded databases, ORM/JPA, MVC/REST, ...
- Opinionated configuration, yet quickly modifiable for different requirements.
    - Easy/quick integration with standard technologies/frameworks.
    - Examples:
        - Web applications -- Tomcat
        - ORM -- Hibernate
        - JSON -- Jackson
        - Test -- JUnit
    - You can change the configuration/technologies by adding alternates on pom.xml (classpath).
- Provide lot of non-functional common features (e.g. security, servers, health checks, ...).
- Easy deployment and Containerisation.
    - Embedded Web Server for web applications.

Spring Boot Hello World Application (boot01)

- step 1: Spring Starter Project
    - Fill type=Maven, group id, artifact id, Spring Boot=3.x.y, Language=Java, Java version=17, packaging=Jar -- click Finish
- step 2: Inherit Boot01Application class from CommandLineRunner and implement run() method.

```
@SpringBootApplication
public class Boot01Application implements CommandLineRunner {
    public static void main(String[] args) {
        SpringApplication.run(Boot01Application.class, args);
```

```
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hello, Spring Boot!");
    }
}
```

- step 3: Modify pom.xml
  - `<version>2.7.18</version>`
  - `<java.version>11</java.version>`
- step 4: Right click on project -> Maven -> Update Project.
- step 5: Right click on main class and Run as "Spring Boot Application".

Assignments

1. Add User registration, Book List and Add New Book feature into JSP bookshop.