**Spring Boot**

Additional points: [Lambda Tutorial](https://www.youtube.com/watch?v=y6BMg0T6Uck&list=PLqq-6Pq4lTTa9YGfyhyW2CqdtW9RtY-I3&index=25)

* Lambda expression is a way of implementing the interface inline. It basically implements the functional interfaces. Functional interfaces are interfaces with just one abstract method (can have other implemented methods) that needs to be implemented using lambda. Lambda function doesn’t need the function modifiers, return type and the function name. It doesn’t require the return keyword if it’s a single line lambda.

FuntionalInterfaceType lambdaFunction = (arg1, arg2) -> operation using arg1 and arg2.

The functional interface should have an abstract method which has the same argument types as the lambda function.

* + Instead of creating a functional interface for every lambda expression, use the function library in the java.util which has so many common scenario functional interfaces.
* Anonymous inner class also works the same way as lambda
* Closure in Java: Inside the anonymous inner class or the lambda expression, if we are using a outsider variable say b, the b is taken as final or effectively final. Meaning the compiler trusts that the value wont change and will use it inside the anonymous class or the lambda expression. The value of the variable is frozen and works throughout the anonymous inner class and the lambda. If that value gets changed, the compiler will still use the frozen value or throws an error.

Closure requires those variables to be final or effectively final (need no final keyword)

* The ‘this’ keywork works differently in anonymous inner class and the lambda expression. The anonymous class will rewrite this keywork with the instance of the anonymous class created. But lambda does not overwrite this keyword.
* Method Reference: Method reference is a different syntax of using the lambda expression through which methods can be called. The syntax is ClassName::MethodName without parenthesis.
* Streams: The typical for loop and the for in loop are called external iterators meaning the user has to control the iteration.
  + Streams are internal iterators which uses the functional interfaces and thus lambdas.
  + Streams are used in every Collections in Java 8. They work in the fashion of a conveyor belt. We can perform multiple operations on the collections and the result of one operation goes to the next.
  + It also helps in parallel processing.

**Different ways to create a spring boot application:**

* **Create a maven project and add the dependencies.**
* **Create the spring project from spring initializr**
* **From spring CLI**

**Creating Spring Boot project** from pom file and Maven configuration - [You tube Link](https://www.youtube.com/watch?v=bDtZvYAT5Sc&list=PLmbC-xnvykcghOSOJ1ZF6ja3aOgZAgaMO&index=8).

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.4.2.RELEASE</version>  
</parent>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>

<properties>  
 <java.version>1.8</java.version>  
</properties>

The spring-boot-starter-parent project is the parent project which says this maven project is a spring boot project. The parent project includes all the configurations and the basic dependencies used for the spring application. The dependency spring-boot-starter-web is the dependency which contain all the jar files which is required for developing a web application. By default this dependency will have some jar which are outdated like the Java version. This can be manually overwritten using the properties.

@SpringBootApplication  
public class AppInitializer {  
 public static void main(String[] args){  
 SpringApplication.*run*(AppInitializer.class, args);  
 }  
}

The class annotated with the SpringBootApplication is the driver class for the application. The class SpringApplication and the static method run is used to create all the servlet deployment and other configurations used to make the spring boot application stand alone. Calling this method:

* Sets up the default configuration which automates the configuration part.
* It starts the spring application context. (Container for all the services and the controllers etc)
* Performs a class path scan. It scans all the classes for spring annotations. Eg: classes with @service are treated as service classes, classes with @Controller are treated as controllers etc. It tells how those classes needs to behave.
* Starts the tomcat server. Springboot comes with the inbuilt tomcat server.

@RestController  
public class TopicController {  
  
 @RequestMapping("/topics")  
 public static List<Topic> getAllTopics(){  
 return Arrays.*asList*(  
 new Topic("Spring", "Spring Framework", "Spring desc"),  
 new Topic("JAVA", "Java Framework", "Java desc")  
 );  
 }  
}

Due to the @RestController annotation, spring will register that if the end point /topics is passed, it should execute the above method and return the output. Spring MVC automatically converts the List of objects to json.

In the pom file, the parent section doesn’t download the jars(dependencies) required by the application, rather configures the versions, other details for the downloaded dependency. It actually decides which version of the dependencies goes well with each other according to the parent artifact version.

**Embedded Tomcat server:**

* Convenience
* Tomcat configurations can be modified in the application config.
* Standalone application and useful for microservice architecture.

How spring MVC works: [You tube Link](https://www.youtube.com/watch?v=xkDBaldNki4&list=PLmbC-xnvykcghOSOJ1ZF6ja3aOgZAgaMO&index=15)

**Application Controllers:**



The concept of singleton classes are used here. There should be a service class which is initialised only once during the run (each POST, GET request should not create instances rather it should use the singleton (single instance) instance the spring created).

Any changes to the members present inside the singleton class will reflect in the next endpoint hit.

**Annotations**

* @SpringBootApplicaiton – Denotes the driver class and says it’s a spring application
* @Service – denotes it’s a service class, that has the business login. Usually its singleton and can be injected using Dependency Injection.
* @Autoeired – Used for dependency injection
* @RestController – Denotes this is a controller class. URL are checked with these classes if they are mapped.
* @RequestMapping – annotation used for http requests. Default method is GET. We can configure this annotation to use the POST or any other request types using RequestMethod parameter.
* @PathVariable – this annotation is used to denote that the variable passed to the method is a variable in the URL path.

**Configuring Spring boot application**

The spring boot application is the conventional way of creating the spring application. It resolves or implements 80% of the use case but the remaining 20% is to be configured according to the user needs. This can be done by modifying the property file: **application.properties**

**Entity Relation between Classes**

Spring boot can maintain the entity relations between the model classes using the JPA annotations like the @ManyToOne. (Read more)

**Repository Interface**

The Crud Repository is the parent interface through which many common queries can be executed. We can create custom method declarations which doesn’t need implementations.

public interface CourseRepository extends CrudRepository<Course, String> {  
 List<Course> findByTopicId(String topicId);  
}

Here We are finding the courses which are present in a topic. The topic class is a variable inside the Course class and we are finding the courses which have a specific topicId, thus **findByTopicId** where Topic is the class and Id is the variable we use to filter.

**Packaging and Running the Spring boot application**

From the source directory, use **mvn clean install** to install all the dependencies and create a jar file. Then use **java -jar filename.jar**

Change the packaging to war in order to use it in a existing tomcat server.

**Spring Boot Actuatr**

Spring boot actuator is used for management of the spring boot application post deployment. It has a health endpoint which shows the health status of the server and db. We can direct the **/health** endpoint to a different pot using **management.port** in application.properties.

It also has a /bean end point