upGrad
#LifeKoKaroLift

Introduction to Spring Boot

Course: Developing RESTful Web APIs using Spring Boot

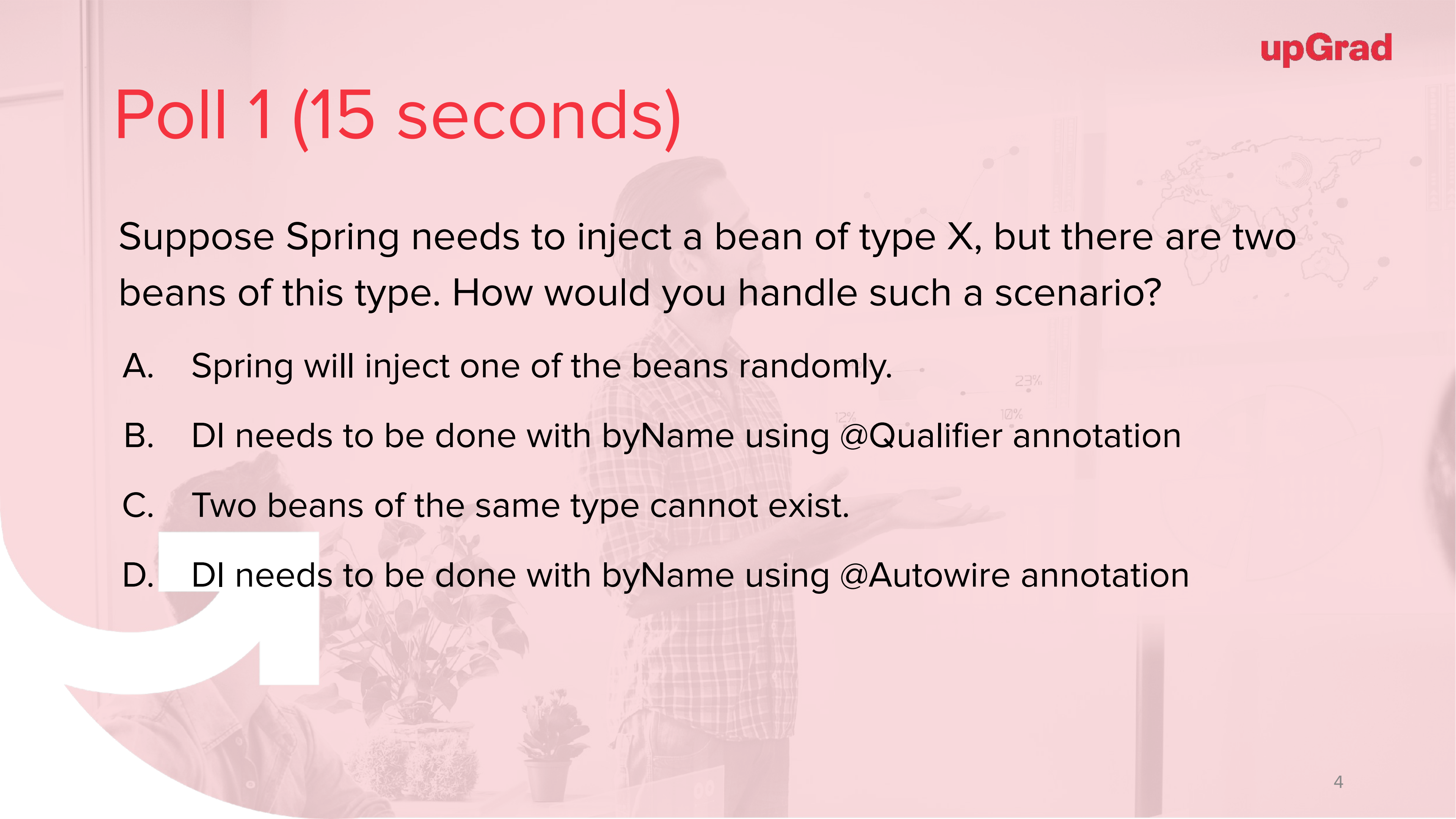
Lecture On: Introduction to Spring Boot

Instructor: Vishwa Mohan

upGrad

Topics covered in the previous class...

1. Introduction to Coupling, IoC and DI
2. Spring IoC Container
3. Different ways to Inject dependencies
4. @Component, @Autowired and @Qualifier annotations
5. Scopes of Spring Bean
6. Other features offered by the Spring framework
7. Maven



Poll 1 (15 seconds)

Suppose Spring needs to inject a bean of type X, but there are two beans of this type. How would you handle such a scenario?

A. Spring will inject one of the beans randomly.

B. DI needs to be done with byName using @Qualifier annotation

C. Two beans of the same type cannot exist.

D. DI needs to be done with byName using @Autowire annotation

Poll 1 (15 seconds)

Suppose Spring needs to inject a bean of type X, but there are two beans of this type. How would you handle such a scenario?

A. Spring will inject one of the beans randomly.

B. DI need to be done with byName using @Qualifier annotation.

C. Two beans of the same type cannot exist.

D. DI need to be done with byName using @Autowire annotation.

Homework Discussion

https://github.com/ishwar-soni/Calculator

upGrad

Today’s Agenda

- **Introduction to Spring Boot**
  - What is Spring Boot and what are its features?
  - Different ways to create Spring Boot applications
  - Different components of a Spring Boot pom.xml file
  - Embedded servers and containers
  - Spring Boot auto-configuration
  - Basic Spring Boot annotations

- **Developing a Spring Boot application and deploying it as JAR and WAR**
  - Use DevTools to improve efficiency

What is Spring Boot?

What is Spring Boot?

upGrad

- In the previous session, you learnt how Spring helps you to create loosely coupled applications without the need to write boilerplate code.
- But when you are developing big enterprise applications using Spring, you have to use a lot of external libraries and provide configurations to create beans for them.
- This compels you to write a lot of ‘boilerplate configurations’.

What is Spring Boot?

upGrad

- This is where Spring Boot comes to the rescue. It eliminates the need to provide boilerplate configuration for each and every external library that you are using.
- So, Spring eliminates the need for ‘boilerplate code for loosely coupled applications’, whereas Spring Boot eliminates the need for ‘boilerplate configurations for Spring applications’.
- With Spring Boot, you need to focus only on the business logic, without thinking about boilerplate code and boilerplate configurations.

What is Spring Boot?

- So, what is Spring Boot actually? As we discussed in the previous session, Spring Boot is one of the sub-projects of the Spring framework, which eliminates the need to provide boilerplate configurations.
- How does it eliminate the need for configuration? It takes an **Opinionated view of building Spring applications**. This means Spring Boot comes with certain default configurations, and it makes some smart assumptions to auto-configure your application.
- These default configurations are generally the ones that are used popularly and help in easy and faster development of applications. All of these defaults are easily overridden although they are mostly preferred.
- It is like the autocomplete feature used by Google search.

What is Spring Boot?

- For example, if you put a JDBC dependency in your pom.xml file, then it will download all other related dependencies, set up an in-memory database for you and create a datasource bean, which will be injected automatically into your application.
- Another example for auto-configuration is Tomcat server. Spring Boot comes with Tomcat as an embedded server. Tomcat is one of the most widely used web application servers.
- So, with Spring Boot, you have to provide minimal configuration to get your application running.

Spring vs Spring Boot

Spring	Spring Boot
It eliminates the need for boilerplate code to develop loosely coupled applications	It eliminates the need for boilerplate configurations to develop Spring applications
IoC and DI are the main features	Auto-configuration is the main feature
You need to list down each and every dependency separately in the pom.xml file	You only need to list down starter projects in pom.xml. These starter projects will download the required dependencies automatically

Spring vs Spring Boot

Spring	Spring Boot
It takes time and effort to run Spring projects as you need to provide a lot of configuration	You can run Spring Boot projects as soon as you create one
It requires additional support to set up the server	It comes with an embedded server, for example, Tomcat and Jetty, which are fully configured
It supports both XML and Annotation configurations	It does not require XML configuration

Spring Boot - Features

Here are some of the Spring Boot features, which make it one of the widely used frameworks in today’s IT ecosystem:

- **Auto-configuration -** Based on the dependencies added in the pom.xml file, Spring Boot performs the necessary configurations automatically.
- **Standalone -** Web applications built with Spring Boot do not need to be deployed on a web server. You just need to click the run button and Spring Boot will use the embedded server to run the application.
- **Opinionated -** Spring Boot uses default configurations to make smart assumptions to auto-configure your application. This reduces development time and improves efficiency.

Spring Boot - Features

upGrad

- **Integration with other frameworks -** It is very easy to integrate Spring Boot applications with other frameworks, such as Spring Data, Spring JDBC and Spring Security, since Spring Boot uses opinionated defaults to configure them.
- **Starter Project -** With Spring Boot, you only need to specify starter projects in the pom.xml file and it will download all the dependencies and configure them. Thus, you do not need to list down each and every dependency individually.
- **Logging and Security -** Applications that use Spring Boot are provided proper security and logging so we can keep track of them as well as their development.

Minimum System Requirements

upGrad

- Before we step into developing our first application, let’s take a look at the minimum requirements for building a Spring Boot application.
- For Spring Boot 2.3.3.RELEASE version, following are the system requirements:
    - Java 8 and compatible up to Java 14 (included)
    - Spring Framework 5.2.8.RELEASE or above
    - Build Support
        - Maven (3.3+)
        - Gradle 6 (6.3 or later)
    - Embedded Servlet Containers
        - Tomcat 9.0 - Servlet Version 4.0
        - Jetty 9.4 - Servlet Version 3.1
        - Undertow 2.0 - Servlet Version 4.0

Poll 2 (15 seconds)

Which of the following statement(s) regarding Spring Boot is/are true? (More than one option may be correct)

A. It helps you develop standalone applications.

B. It eliminates the need for IoC and DI.

C. The default configuration provided by the Spring Boot cannot be overridden.

D. It reduces the overall development time.

Poll 2 (15 seconds)

Which of the following statement(s) regarding Spring Boot is/are true? (More than one option may be correct)

A. It helps you develop standalone applications.

B. It eliminates the need for IoC and DI.

C. The default configuration provided by the Spring Boot cannot be overridden.

D. It reduces the overall development time.

upGrad

Developing a Spring Boot Application

Developing a Spring Boot Application

There are four methods to develop a Spring Boot application using Maven:

1. Spring Boot Initializr
2. Spring Starter Project Wizard Of IntelliJ or Eclipse
3. Spring Boot CLI (a command-line interface to build Spring Boot applications using Groovy)
4. Spring Maven Project

Of these methods, Spring Boot Initializr is used the most commonly to develop Spring Boot applications.

Spring Boot Initializr

Let's create the Spring Boot project for our Movie Booking application. You will learn more about the Movie Booking application and start building it from the next session.

1. Go to Spring Initializr website by clicking on the following link:

   https://start.spring.io/

2. You can select project, language and version as shown below:
   a. **Project:** Maven
   b. **Language:** Java
   c. **Spring Boot Version:** 2.3.4

Spring Boot Initializr

3. You can fill the project metadata as shown below:
   a. **Group:** com.upgrad
   b. **Artifact:** mba
   c. **Name:** MovieBookingApplication
   d. **Description:** RESTful API for Movie Booking Application
   e. **Package Name:** com.upgrad.mba
4. You can select packaging and Java version as shown below:
   a. **Packaging:** Jar
   b. **Java:** 11
5. Since we would be building an web application, add ‘Spring Web’ as an dependency.
6. You can check the final project structure by clicking the ‘Explore’ button, or you can download the zipped project by clicking on the ‘Generate’ button.

Spring Boot Initializr

upGrad

7. The project will be downloaded as ‘mba.zip’ or by the Artifact name.
8. Extract the zipped folder and open it in IntelliJ.
9. You can run the project directly without performing any configuration. You will learn how to run a Spring Boot project later in the session.

Hands-On Experience

upGrad

Now that you have learnt how to create a Spring Boot project using Spring Initializr, it is time to create a Spring Boot project using Spring Initializr and open it in IntelliJ.

upGrad

Spring Boot pom.xml File

Spring Boot pom.xml File

upGrad

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

XML and Maven metadata

```
  <parent>
     <groupId>org.springframework.boot</ groupId>
     <artifactId>spring-boot-starter-parent</ artifactId>
     <version>2.3.4.RELEASE</ version>
     <relativePath/> <!-- lookup parent from repository -->
  </parent>
```

Parent POM, which provides default configuration for Spring Boot projects

```
  <groupId>com.upgrad</ groupId>
  <artifactId>mba</artifactId>
  <version>0.0.1-SNAPSHOT</ version>
  <name>MovieBookingApplication</ name>
  <description>RESTful API for Movie Booking Application</ description>
```

Project metadata or Artifact Coordinate

Spring Boot pom.xml File

```
<properties>
    <java.version>11</java.version>
</properties>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

JDK version

Starter POM or Project for Spring Boot web applications

Starter POM or Project for writing unit tests for Spring Boot applications

Spring Boot Starters

- Spring Boot Starters are POM files that contain a list of all related dependency jars, which are used together to implement a type of feature.
- Thus, instead of adding maven dependencies for each jar, you can just mention the starter pom using the **<dependency>** tag and it will help you download all the related dependencies into the project.
- All the starters are named in a similar pattern:

  spring-boot-starter-* , *where * is the type of application*
- For example, when we add **spring-boot-starter-web** into our pom.xml file, it will download all the dependencies that are required to build a RESTful web application using Spring Boot, such as spring mvc, tomcat server and hibernate-validator

Spring Boot Starters

Here are some of the commonly used starters:

- **spring-boot-starter:** Core starter. Used to enable logging, auto-configuration and YAML
- **spring-boot-starter-web:** Used for creating web and RESTful applications with Spring MVC
- **spring-boot-starter-data-jpa:** Used for database access using Spring Data JPA with Hibernate
- **spring-boot-starter-security:** Used for Spring Security
- **spring-boot-starter-test:** Used for testing an application with libraries such as Junit and Mockito
- **spring-boot-starter-aop**: Used for Aspect Oriented Programming with Spring AOP and AspectJ

Spring Boot pom.xml File

upGrad

```
  <build>
     <plugins>
        <plugin>
           <groupId>org.springframework.boot</ groupId>
           <artifactId>spring-boot-maven-plugin</ artifactId>
        </plugin>
     </plugins>
  </build>

</project>
```

This plug-in is used to:

- Package your application as an executable jar or as war archives
- Run Spring Boot applications
- Generate build information
- Start your Spring Boot application prior to running integration tests

Spring Boot Dependency Management

upGrad

- When we are building a Spring Boot Project, we do not specify the versions of the dependencies or the starter POMs in the project’s pom.xml file.
- We only specify the version of **spring-boot-starter-parent** and simply add the dependencies into the pom.xml file.
- Spring Boot will decide suitable versions of the dependencies and download them.
- When we change the version of **spring-boot-starter-parent**, Spring Boot will upgrade all the dependencies automatically.
- However, if you still want to specify the version of the dependency, then you can do so in the pom.xml file.

upGrad

Poll 3 (15 seconds)

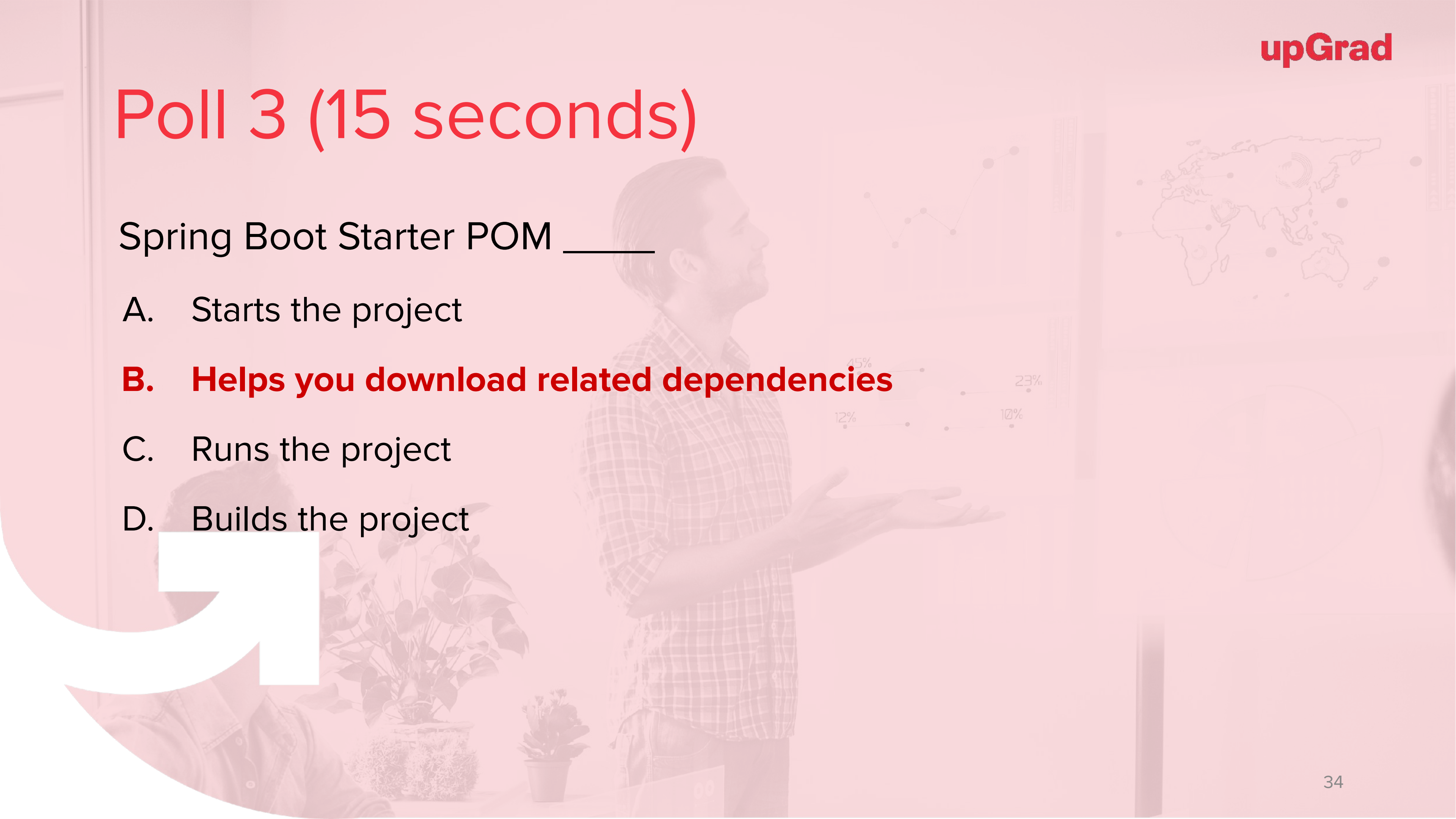
Spring Boot Starter POM _____

A. Starts the project

B. Helps you download related dependencies

C. Runs the project

D. Build the project



Poll 3 (15 seconds)

Spring Boot Starter POM _____

A. Starts the project

B. Helps you download related dependencies

C. Runs the project

D. Builds the project

upGrad

Poll 4 (15 seconds)

Which of the following statement(s) regarding **spring-boot-maven-plugin** is/are true? (Multiple options can be the correct choice.)

A. It helps you package your application as Jar only.

B. It helps you package your application as Jar or War.

C. It is added to the pom.xml file using the <plugin> tag.

D. It is added to the pom.xml file using the <dependency> tag.

upGrad

Poll 4 (15 seconds)

Which of the following statement(s) regarding **spring-boot-maven-plugin** is/are true? (Multiple options can be the correct choice.)

A. It helps you package your application as Jar only.

B. It helps you package your application as Jar or War.

C. It is added to the pom.xml file using the <plugin> tag.

D. It is added to the pom.xml file using the <dependency> tag.

Embedded Servers/Containers

Embedded Servers/Containers

upGrad

- When we build a web application using Java (Servlet/JSP) or Spring, we have to perform the following steps:
    1. Develop the web application
    2. Package it as War
    3. Download and install a web server
    4. Configure the web server
    5. Deploy the application (War packaging) on the server
    6. Run the server, which in turn runs the application
- Thus, considerable time is spent from Step 2 to Step 6, and this deviates us from the main goal (Step 1). How to solve this issue?

Embedded Servers/Containers

upGrad

- With Spring Boot, web applications can be packaged as Jar. This Jar file contains both the application code and an embedded server (servers that are embedded as part of the deployable application).
- Thus, with Spring Boot, you can build web applications in just two steps:
  1. Develop the web application
  2. Click the run button (just like a normal Java application)
- Spring Boot supports three embedded servers/containers for building applications: **Tomcat**, **Undertow** and **Jetty**.

Embedded Servers and Containers

upGrad

- Tomcat is the (Opinionated) default embedded server for Spring Boot applications. Nevertheless, you can configure your application to have Jetty or Undertow as the default embedded server through the pom.xml file.
- The choice of a container is usually dependant on factors such as memory requirement, speed, available configuration options, comfort, features and policy, to name a few.

Change Embedded Server to Jetty

upGrad

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <exclusions>
            <!-- Exclude the Tomcat dependency -->
            <exclusion>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
      </exclusions>
</dependency>
<!-- Use Jetty instead -->
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jetty</artifactId>
  </dependency>
```

Poll 5 (15 seconds)

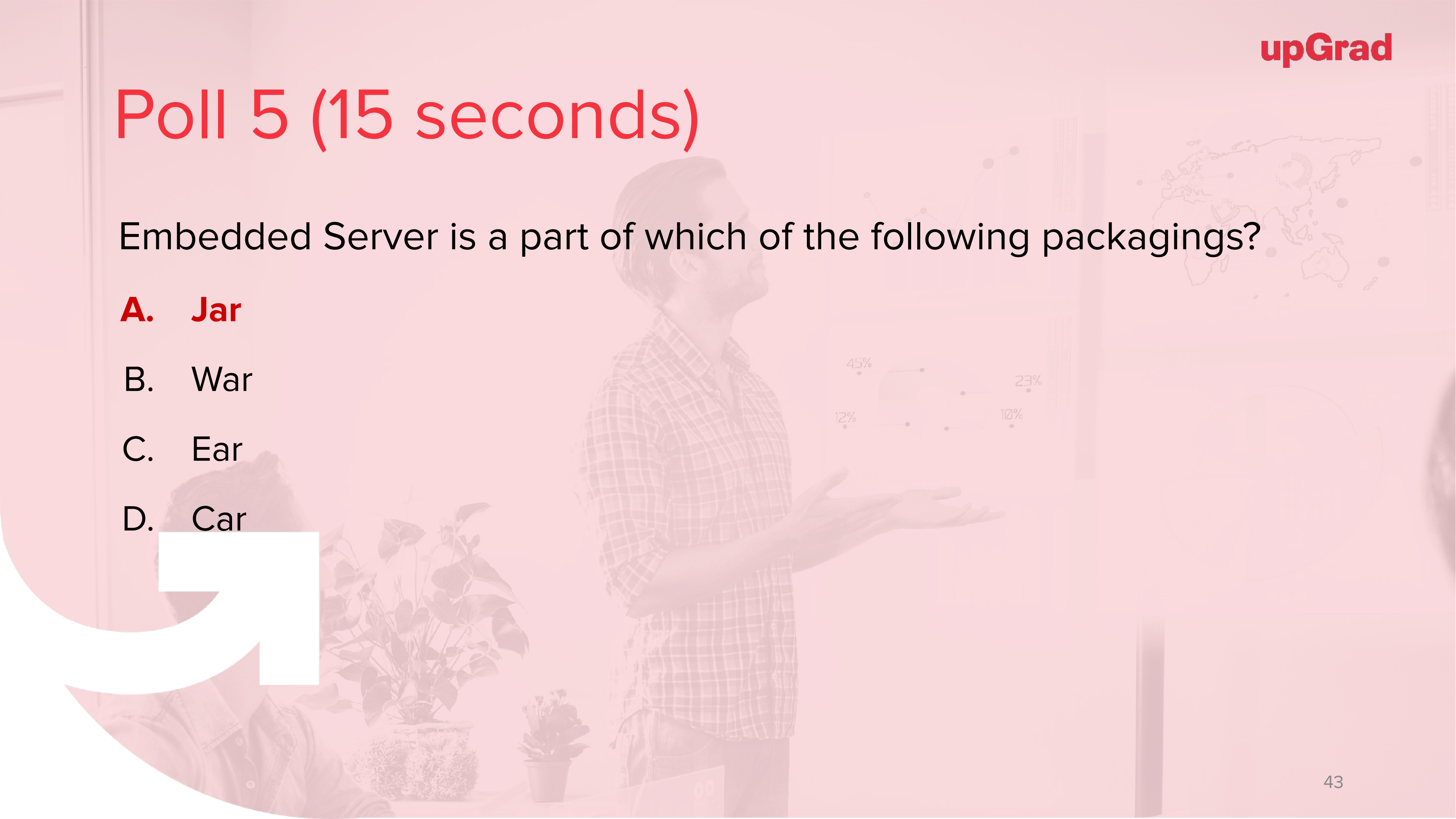
Embedded Server is a part of which of the following packagings?

A. Jar

B. War

C. Ear

D. Car



Poll 5 (15 seconds)

Embedded Server is a part of which of the following packagings?

A. Jar

B. War

C. Ear

D. Car

Spring Boot Auto-Configuration

Spring Boot Auto-Configuration

upGrad

Spring Boot = Spring + Auto-Configuration

```
for-each (dependency present in the class path) {
        if (bean already created by the user) {
                continue;
        }
        if (custom configuration provided in the application.properties file) {
                createBeanWithCustomProperties();
        } else {
                createBeanWithOpinionatedDefaults();
        }
}
```

Spring Boot Auto-Configuration

- So, Spring Boot is nothing but ‘Spring on steroids’ or ‘Spring with auto-configuration’.
- Normally, auto-configuration is triggered by annotating the Main class with **@SpringBootApplication** annotation.
- This annotation is a combination of the following three annotations:
  - ***@SpringBootConfiguration* -** It enables you to provide a Java-based configuration in the Main class.
  - ***@EnableAutoConfiguration* -** ***triggers auto-configuration***
  - ***@ComponentScan* -** It enables you to scan the @Component classes, which are present in the package that contains the Main class or its subpackages.

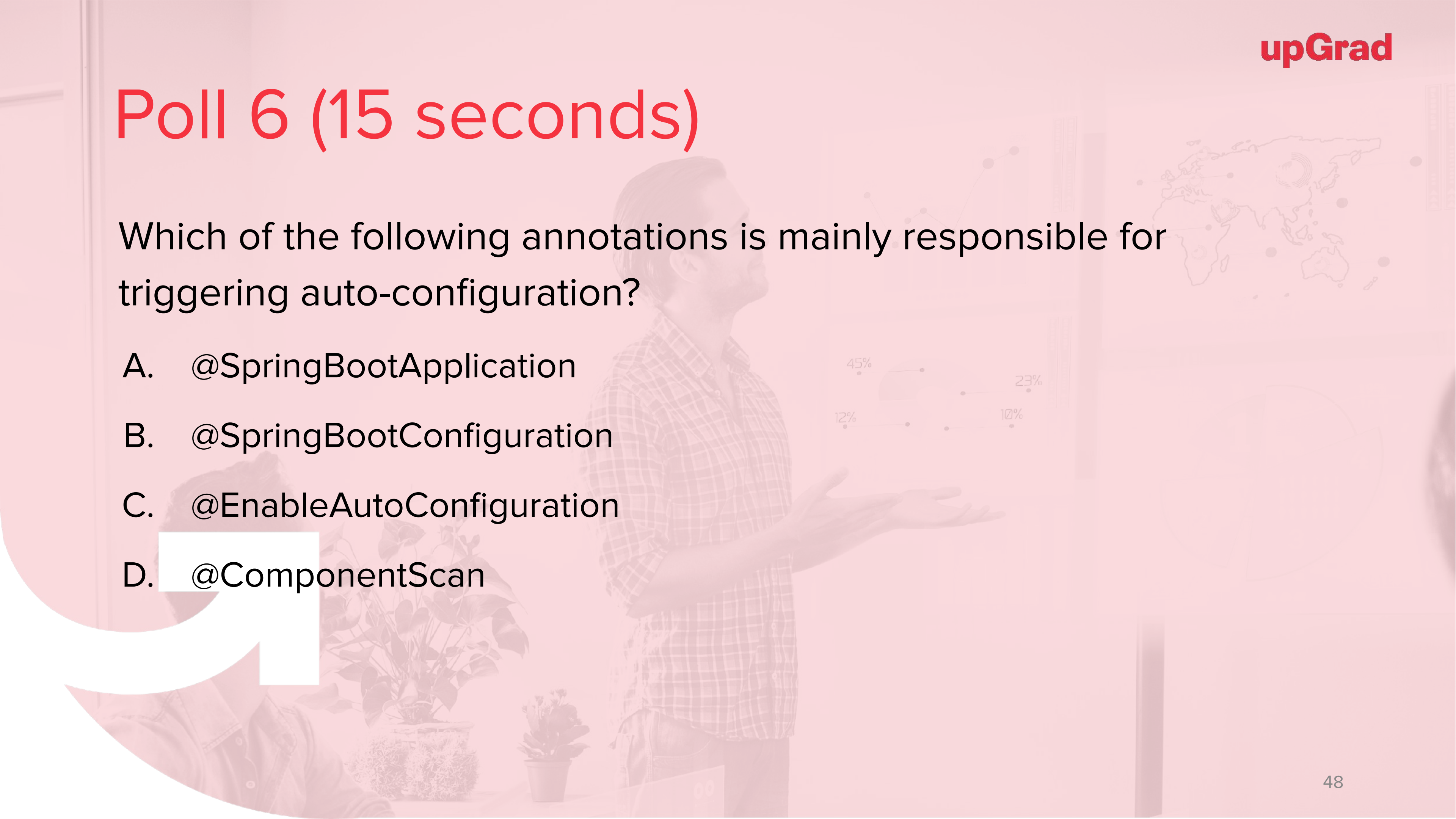
Spring Boot Auto-Configuration

upGrad

- You can disable auto-configuration entirely by removing the *@SpringBootApplication* annotation from the Main class.
- Or you can disable some specific auto-configuration using the *exclude* attribute of the *@SpringBootApplication* annotation.
- For example, if you want to disable auto-configuration of the database, then you can do so as shown below:

```
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})
```

- You can also provide a custom configuration using the *application.properties* file. You will learn about it later in the session.
- You can also provide a custom configuration using the *spring.factories* file. You can read more about it here.



Poll 6 (15 seconds)

Which of the following annotations is mainly responsible for triggering auto-configuration?

A. @SpringBootApplication

B. @SpringBootConfiguration

C. @EnableAutoConfiguration

D. @ComponentScan

upGrad

Poll 6 (15 seconds)

Which of the following annotations is mainly responsible for triggering auto-configuration?

A. @SpringBootApplication

B. @SpringBootConfiguration

C. @EnableAutoConfiguration

D. @ComponentScan

application.properties Configuration File

application.properties Configuration File

upGrad

- So far, you have learned that Spring Boot creates beans only for those dependencies in the classpath for which the developer has not created beans.
- While creating beans, Spring Boot first checks whether the developer has provided custom properties for that bean in the ***application.properties*** file.
- If the dependency needs 10 properties to be instantiated as bean but the developer has provided values for only 2 of the properties, then Spring Boot creates a bean using 2 custom values and 8 default values.
- By default, this file is empty, which means Spring Boot uses only the opinionated defaults for the auto-configuration of beans.

application.properties Configuration File

- For example, if Spring Boot finds a Tomcat dependency in the classpath, then it will configure and set up the server for you. By default, the server will run on port 8080. But you can change it by defining the following property in the ***application.properties*** file:

  ```
  server.port=8081
  ```

- You can get the complete list of the application properties here.

Alternatives to application.properties File

- The ***application.properties*** file is one of the means for providing custom properties to override the opinionated defaults used by Spring Boot.
- The other alternatives (based on order of precedence) are:
    - Command-line arguments,
    - Java-system properties,
    - Environment variables and
    - YAML files.
- With these, you can externalise your configuration so the same application code can work in different environments.

Configuration Using Command-Line Arguments

upGrad

- You can provide custom properties using command-line arguments by prefixing the arguments with '--' (double hyphen).
- For example, if you want to run an application on server port 8081, then you can do so using command-line arguments as shown below:

```
java -jar mba.jar --server.port="8081"
```

- Command-line properties take precedence over any other source of properties.

Configuration Using Java-System Properties

- You can also provide custom configurations by using Java-system properties
- You can set a Java-system property in one of the following two ways:
    a. `java -Dserver.port="8081" -jar mba.jar`
    b. `System.setProperty("server.port", "8081");`
- You can also access Java system properties as shown below:

```
System.getProperty("server.port")
```

Configuration Using Environment Variables

- You can also provide custom configurations by using environment variables.
- You can set environment variables just before running your application as shown below:

```
export SERVER_PORT="8081"

java -jar mba.jar
```

Configuration Using YAML Files

- YAML files are exactly the same as properties files and both of them follow the same rule. The only difference is that YAML files are more convenient when dealing with data of hierarchical configuration.
- For example, say you want to provide a custom configuration for a database. You can do so using application.properties as shown below:

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=ishwar
spring.datasource.password=oracle
```

- You can do the same using a YAML file as shown below:

```
spring:
    datasource:
        url: jdbc:oracle:thin:@localhost:1521:xe
        username: ishwar
        password: oracle
```

Configuration Using YAML Files

upGrad

- YAML files are also a superset of JSON. You will learn about JSON later in the course.
- Due to the absence of repetitive prefixes and the hierarchical configuration, the YAML configuration looks clearer and more readable.
- To use YAML files for a custom configuration, you need to add the SnakeYAML library to the classpath.

Configuration Using YAML Files

- A YAML file is also a better choice for interacting with other frameworks or libraries since properties files can be used only in Java projects.
- You can load YAML files using ***YamlPropertiesFactoryBean*** (load YAML as Properties) or ***YamlMapFactoryBean*** (load YAML as Map) classes.
- One major shortcoming of YAML files is that they cannot be loaded using the @PropertySource annotation. This annotation works only for properties files.

Custom Configuration Using Profiles

- An application goes through multiple stages while development, typically ‘dev’, ‘testing’ and ‘prod’.
- At different stages, we want to configure our application differently. For example, when an application is at the ‘dev’ stage, we don’t want to integrate it with the prod database; otherwise, it would pollute all the data.
- However, when an application is deployed on the prod server, we want it to work with the read database or the prod database.
- One way to do this is to modify the custom configuration in the application.properties everytime the application moves from one stage to another.
- But this approach is quite cumbersome and error-prone. This is where *Profiles* help you provide environment-specific properties for applications.

Custom Configuration Using Profiles

upGrad

- You can provide environment-specific custom configurations in the ***application-{profile}.properties*** file.
- Therefore, there can be multiple properties files, one for each environment, and one properties file for the default settings. For example:

  application.properties

  application-dev.properties

  application-testing.properties

  application-prod.properties

- You can also control bean creation for specific profiles by marking a component class with the ***@Profile("profile-name")*** annotation.

Set Active Profiles

You can set active profiles in one of the following ways:

1. Java-System Properties

   ```
   -Dspring.profiles.active=dev
   ```

2. Environment Variables

   ```
   export spring_profiles_active=dev
   ```

3. Properties Files

   ```
   spring.profiles.active=dev
   ```

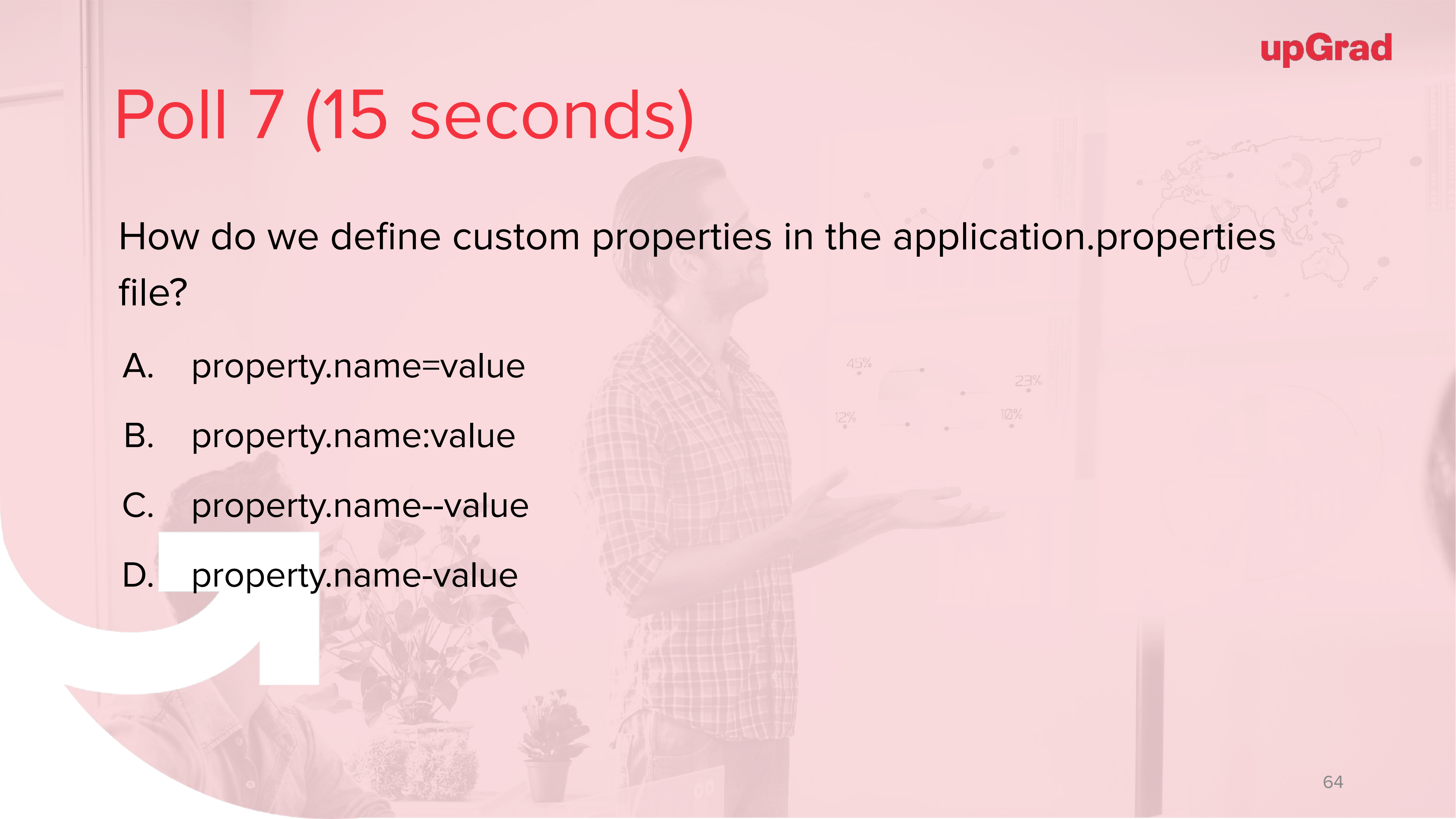
Spring Boot Lazy Initialisation

upGrad

- By default, when Spring Boot starts, it auto-configures all the dependencies and creates beans for them (recall ApplicationContext vs BeanFactory).
- We can ask Spring Boot to create beans only when they are required, as shown below:

```
spring.main.lazy-initialization=true
```

- Lazy initialisation improves the startup time as most of the beans are not getting created until they are required.
- But this also causes latency at runtime, as beans would get created.
- Therefore, the choice of lazy initialisation should be made based on the type of application.



upGrad

Poll 7 (15 seconds)

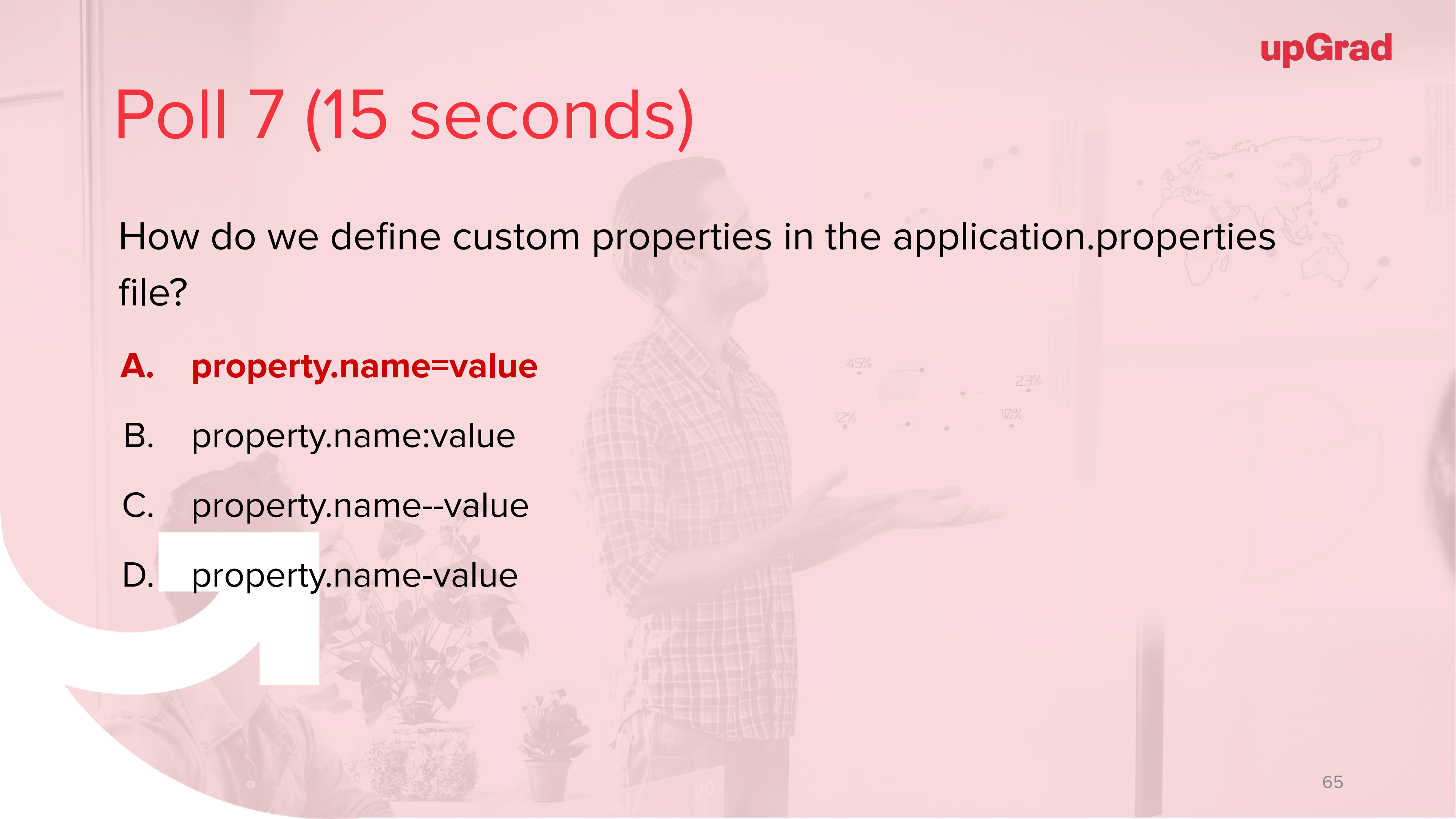
How do we define custom properties in the application.properties file?

A. property.name=value

B. property.name:value

C. property.name--value

D. property.name-value



Poll 7 (15 seconds)

How do we define custom properties in the application.properties file?

A. property.name=value

B. property.name:value

C. property.name--value

D. property.name-value

Poll 8 (15 seconds)

Which of the following statement(s) regarding Spring profiles is/are true? (Multiple options can be correct.)

A. Profiles are used to specify environment-specific properties.

B. You can set an active profile using the application.properties file only.

C. You cannot control bean creation based on the active profile.

D. You can have multiple profile-specific application.properties files.

Poll 8 (15 seconds)

Which of the following statement(s) regarding Spring profiles is/are true? (Multiple options can be correct.)

A. Profiles are used to specify environment-specific properties.

B. You can set an active profile using the application.properties file only.

C. You cannot control bean creation based on the active profile.

D. You can have multiple profile-specific application.properties files.

upGrad

Basic Spring Boot Annotations

Basic Spring Boot Annotations

upGrad

- Before we get our hands dirty and start developing a Spring Boot application, let’s take a look at some of the basic Spring Boot annotations that we should be familiar with:
    - @SpringBootApplication
    - @EnableAutoConfiguration
    - @SpringBootConfiguration
    - @SpringBootTest

$\leftarrow$ Already studied earlier in the session.

Basic Spring Boot Annotations

upGrad

@SpringBootTest

- This annotation is used to mark a test class that runs Spring-Boot-based tests.
- It looks for the class marked with @SpringBootConfiguration or @SpringBootApplication (usually the main class) to read the configurations and create an application context (Spring container) for tests.
- It is mostly useful when you want to test an entire application instead of creating the basic unit tests that are specific to a class.
- You will learn more about Unit or Integration testing in the upcoming sessions.

Spring Boot Deployment Options

Spring Boot Deployment Options

upGrad

- As you have learnt earlier, you can package a Spring Boot application in one of the following two ways before deploying:
    - Jar
    - War

Spring Boot Deployment Options - Jar

upGrad

- Jar is the default packaging for Spring Boot applications.
- When you are deploying your application as Jar, you also need to include spring-boot-maven-plugin in your classpath.
- When you package your application as Jar, an embedded server is also packaged with your application. It helps you deploy your applications as a standalone application.
- Therefore, you do not need to download and configure the web server separately.

Spring Boot Deployment Options - War

upGrad

- You can ask Maven to package to your Spring Boot application as a War file using the packaging tag:

```
<packaging>war</packaging>
```

- War files do not have a server embedded within them, and they have to deployed on external servers.
- The scope of the Tomcat dependency has to be set to *provided* as shown below:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
</dependency>
```

Spring Boot Deployment Options - War

- You also need to initialise the Servlet Context as shown below:

```
public class ServletInitializer extends SpringBootServletInitializer {

    @Override

    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

        return application.sources(DemoApplication.class);

    }

}
```

Poll 9 (15 seconds)

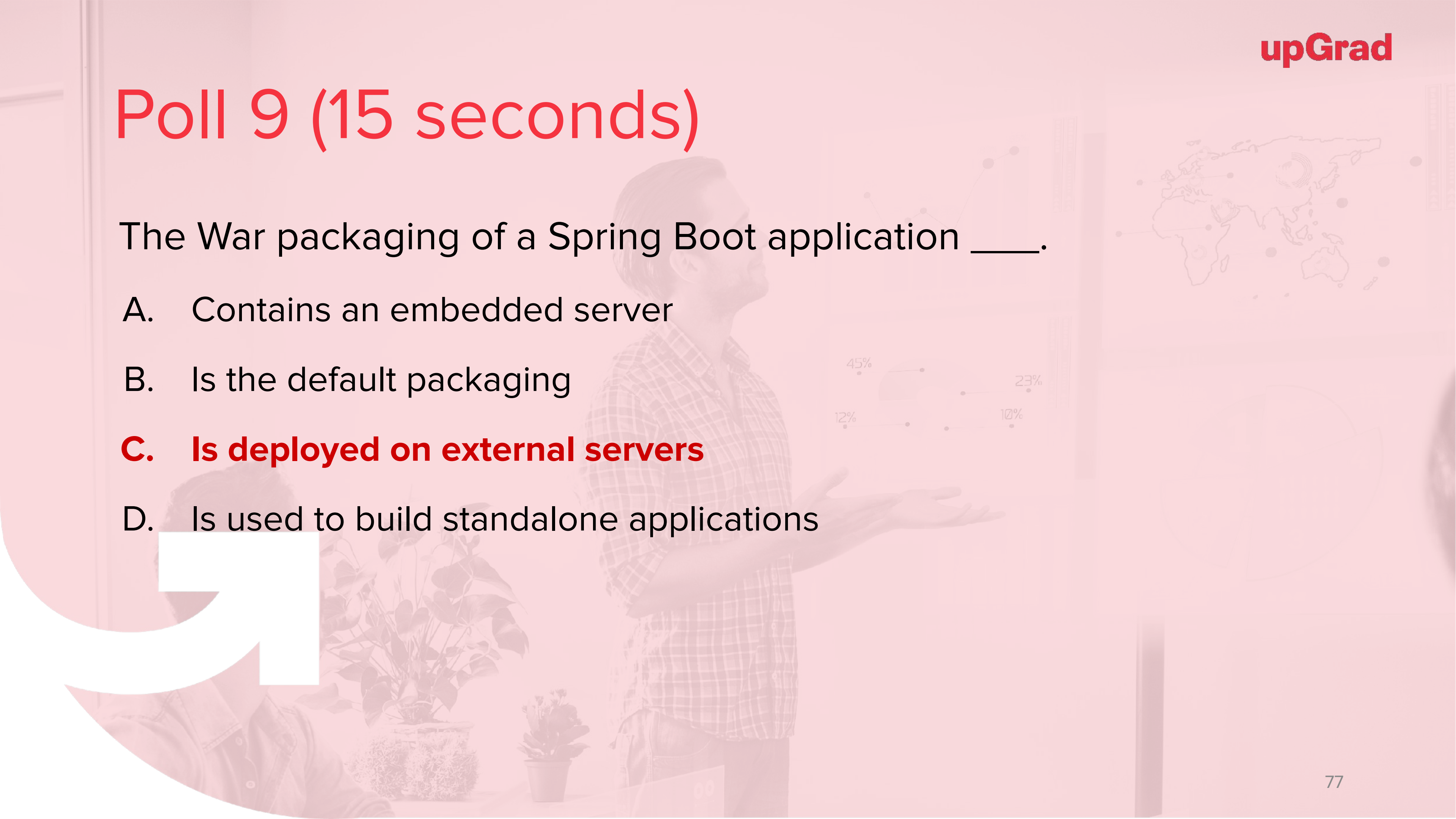
The War packaging of a Spring Boot application ____.

A. Contains an embedded server

B. Is the default packaging

C. Is deployed on external servers

D. Is used to build standalone applications



Poll 9 (15 seconds)

The War packaging of a Spring Boot application ____.

A. Contains an embedded server

B. Is the default packaging

[✓ **C. Is deployed on external servers**]

D. Is used to build standalone applications

Coding Exercise

upGrad

- Run the Movie Booking application as a Jar file
- Add different dependencies using the starter pom and check whether beans are getting created for those dependencies
- Change the embedded server from Tomcat to Jetty
- Show how to provide custom configurations, for example, configure the server port, in different ways, such as using command-line arguments, Java-system properties, environment variables, the application.properties file and YAML files
- Provide different application.properties files based on the profiles
- Set an active profile in different ways and control bean creation based on the active profiles.
- Run the same application as a War file

DevTools

Spring Boot DevTools

- When developers implement some feature, they constantly modify the source code, build/compile the application and start/deploy the application to check whether they are getting the desired result.
- It is cumbersome to hit the restart button every time, and it takes time to recompile/rebuild and restart the application.
- This is where DevTools (Developer tools) help you by automating this entire process.
- Whenever you make changes to the source code, it will auto-reload the application. It will detect the changes, compile the code and run the application with the new changes.

Spring Boot DevTools

- You can add DevTools to your application by adding the following dependency in your pom.xml file:

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-devtools</artifactId>
     <scope>runtime</scope>
     <optional>true</optional>
</dependency>
```

Coding Exercise

upGrad

- Add DevTools to the Movie Booking application to reduce development time and effort

All the codes used in today's session can be found at the link provided below (branch session2-demo):

https://github.com/ishwar-soni/tm-spring-movie-booking-application/tree/session2-demo

Important Concepts and Questions

Important Questions

upGrad

1. Differentiate between an embedded container and War.
2. What are the dependencies that are needed in Spring Boot to build a web application?
3. Can you name some common Spring Boot Starter POMs?
4. Can you explain what happens in the background when a Spring Boot application is ‘run as a Java application’?
5. Would we be able to use Spring Boot with applications that do not use Spring?
6. Can you explain how to deploy to a different server with Spring Boot?
7. Can you create a non-web application in Spring Boot?
8. Explain how to register a custom auto-configuration.
9. Do you think you can use Jetty instead of Tomcat in spring-boot-starter-web?
10. Can you change the port of the embedded Tomcat server in Spring Boot? If yes, then how?

upGrad

Doubt Clearance Window

upGrad

In this class, you learnt that:

1. You should use Spring Boot while developing Spring applications, since Spring Boot provides auto-configurations, which significantly reduces development time.
2. Spring Boot auto-configuration works on Opinionated defaults.
3. There are different methods to create Spring Boot applications, of which Spring Initializr is the one that is used most frequently.
4. You can add maven dependencies using the starter pom to download all related dependencies for implementing a feature.
5. You can also disable the auto-configuration entirely or do it for some specific auto-configurations.
6. There are multiple ways to provide custom properties, of which application.properties is the one that is used most commonly.
7. You can use DevTools to reduce development time and effort significantly.

Homework

upGrad

You have already built a Spring Boot application during the session. Now, try out the following TODOs:

1. Run the Movie Booking application as a Jar file (you should have added spring-boot-starter-web in your classpath; otherwise, your application would run as a console app, and not as a web app)
2. Add various dependencies using the starter pom and check whether beans are getting created for those dependencies. For example, add spring-boot-starter-data-jpa and check whether the DataSource bean is getting created by the Spring Boot (you also need to add a dependency for the h2 database)
3. Change the embedded server from Tomcat to Jetty
4. Provide custom configurations for the dependencies added to the classpath using application.properties. For example, you can change the server port from 8080 to 8082.

Homework

upGrad

5. Provide different application.properties files based on the profiles. For example, for dev, testing and prod
6. Set an active profile using application.properties and control bean creation for the spring-boot-starter-data-jpa dependency
7. Run the same application as a War file
8. Add a DevTools dependency to your application (you can refer to this article if IntelliJ is not working with DevTools)

Code Reference

Note: The solution code (code reference) contains the solution for each TODO in separate commits. You can check the different commits, or check the difference between various commits to see the solution for each TODO.

upGrad

Tasks to Complete After Today's Session

MCQs
Coding Questions
Homework
Project Checkpoint 1

Next Steps

Check this code (**session3-stub** branch) before proceeding to the next session.

upGrad

#RahoAmbitious

Thank You!