upGrad

#LifeKoKaroLift

Spring Core

Course: Developing RESTful Web APIs Using Spring Boot

Lecture On: Spring Core

Instructor: Vishwa Mohan

Learning Objective

The purpose of this course:

- This course will help you build the backend of any web application.
- Some websites that use Spring Boot for the back end are as follows:
  - Trivago
  - Yatra
  - MIT
  - Intuit

Course Map

Spring Core and Spring Boot

1. Learn about Spring basics
2. Introduction to Spring Boot

Spring Boot JPA and Unit Testing

1. Learn about Spring Data JPA
2. Build the data access layer and service layer for a Movie Booking application
3. Learn about TDD using JUnit and Mockito

REST APIs in Spring Boot

1. Learn about RESTful APIs
2. Build RESTful web APIs for the Movie Booking application

Spring Security

1. Learn about Spring Security
2. Build the authorisation layer of the Movie Booking application

Today’s Agenda

- **Develop a loosely coupled application using the Spring Framework**
    - Introduction to Coupling, IoC and DI
    - Spring IoC Container
    - Different ways to inject dependencies
    - @Component, @Autowired and @Qualifier annotations
    - Scopes of Spring Bean
    - Other features provided by the Spring framework

- **Develop a Spring application using the Maven Build Tool**

Why Spring?

Why Spring?

- Helps in creating loosely coupled Java applications

Coupling

upGrad

Loose Coupling

- It is easy to test and debug if any component malfunctions.
- Components are easily replaceable and reusable with other components.

VS

Tight Coupling

- It is difficult to test and debug if any component stops working.
- It is difficult to replace or use components with other components.

Tight Coupling

- When multiple classes are **highly dependent** on each other, it is called tight coupling between classes.

- In software development, normally, an object of one class needs an object of the another class to execute its functionality. For example, a car needs an engine to execute its functionality.

- However, when you refer a dependency class directly inside the dependent class, it introduces tight coupling.

```
class Car {                              class Engine {
    Engine engine;
}                                        }
```

Here, a car (dependent object) is tightly coupled with the engine (dependency).

Tight Coupling

Main Class

```
public class Main {

   public static void main(String[] args) {
       GreetingService greetingService =
new GreetingService();
       greetingService.greet("John");
   }
}
```

Dependent Class

GreetingService Class

```
public class GreetingService {
   public void greet(String name) {
       System.out.println("Hello, " +
name);
   }
}
```

Dependency

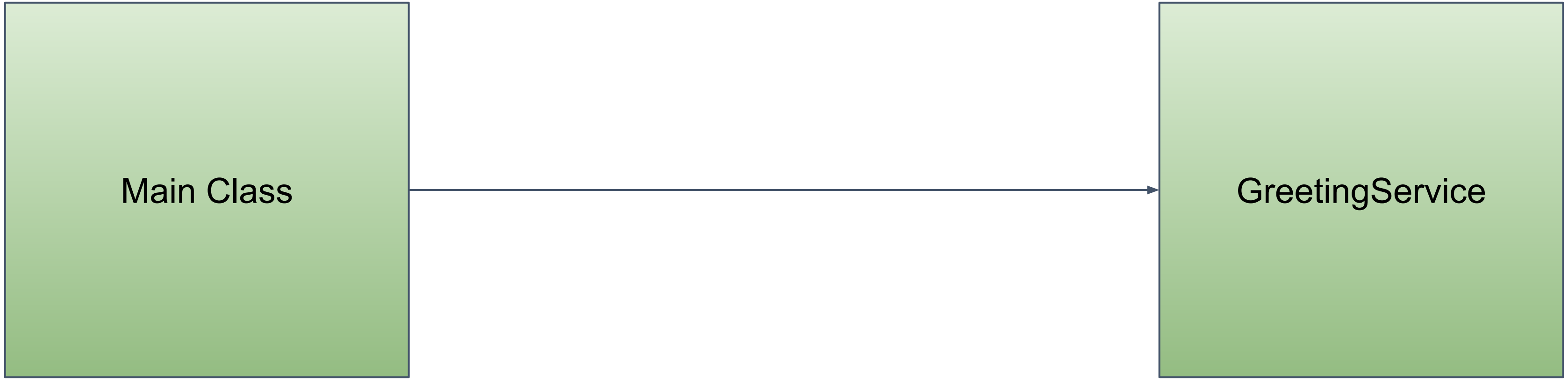
Code Reference

Tight Coupling

- In the previous example, you could observe that the **Main** class was referring to the **GreetingService** class directly. Hence, the Main class was tightly coupled with the GreetingService class.

- In the future, if you want to change the greeting service to some other service that greets in French, for example, you would need to make changes in the Main class as well.

Tight Coupling



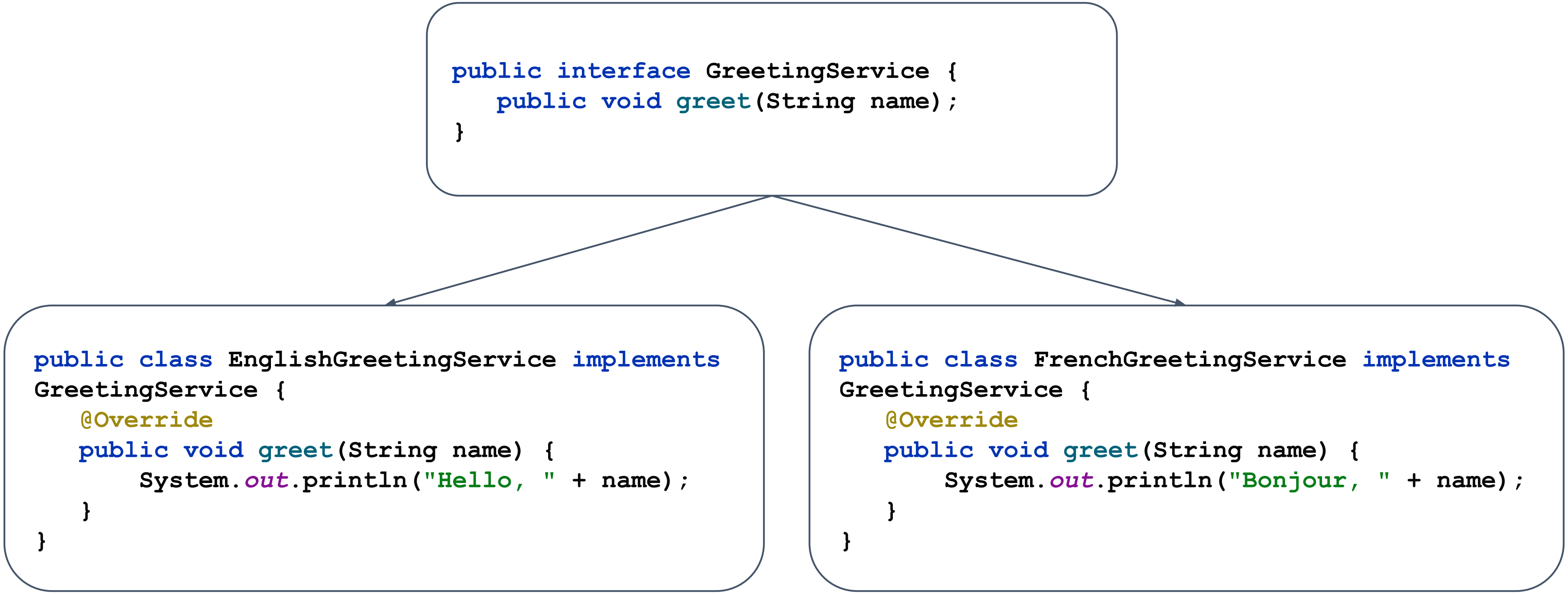
Main Class

GreetingService

Loose Coupling

- We can make applications loosely coupled by introducing the following:
    - Interfaces
    - Factory pattern (Can you recall the factory pattern that you studied in the Java course?)

Loose Coupling



```java
public interface GreetingService {
   public void greet(String name);
}
```

```java
public class EnglishGreetingService implements
GreetingService {
   @Override
   public void greet(String name) {
       System.out.println("Hello, " + name);
   }
}
```

```java
public class FrenchGreetingService implements
GreetingService {
   @Override
   public void greet(String name) {
       System.out.println("Bonjour, " + name);
   }
}
```

Loose Coupling

```java
public class GreetingServiceFactory {
   public GreetingService getGreetingService(String language) {
       if (language.equals("english")) {
           return new EnglishGreetingService();
       } else if (language.equals("french")) {
           return new FrenchGreetingService();
       } else {
           throw new RuntimeException("No greeting Service exist for " + language + " language.");
       }
   }
}
```

Loose Coupling

```java
public class Main {

   public static void main(String[] args) {
       GreetingServiceFactory greetingServiceFactory = new GreetingServiceFactory();
       GreetingService greetingService = greetingServiceFactory.getGreetingService("french");
       greetingService.greet("John");
   }
}
```

Code Reference

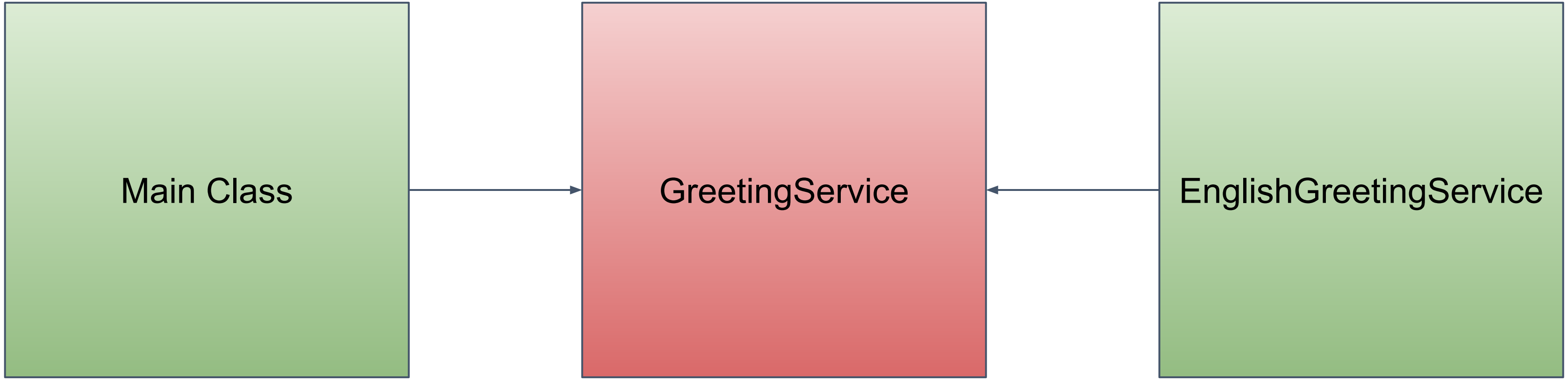
Loose Coupling

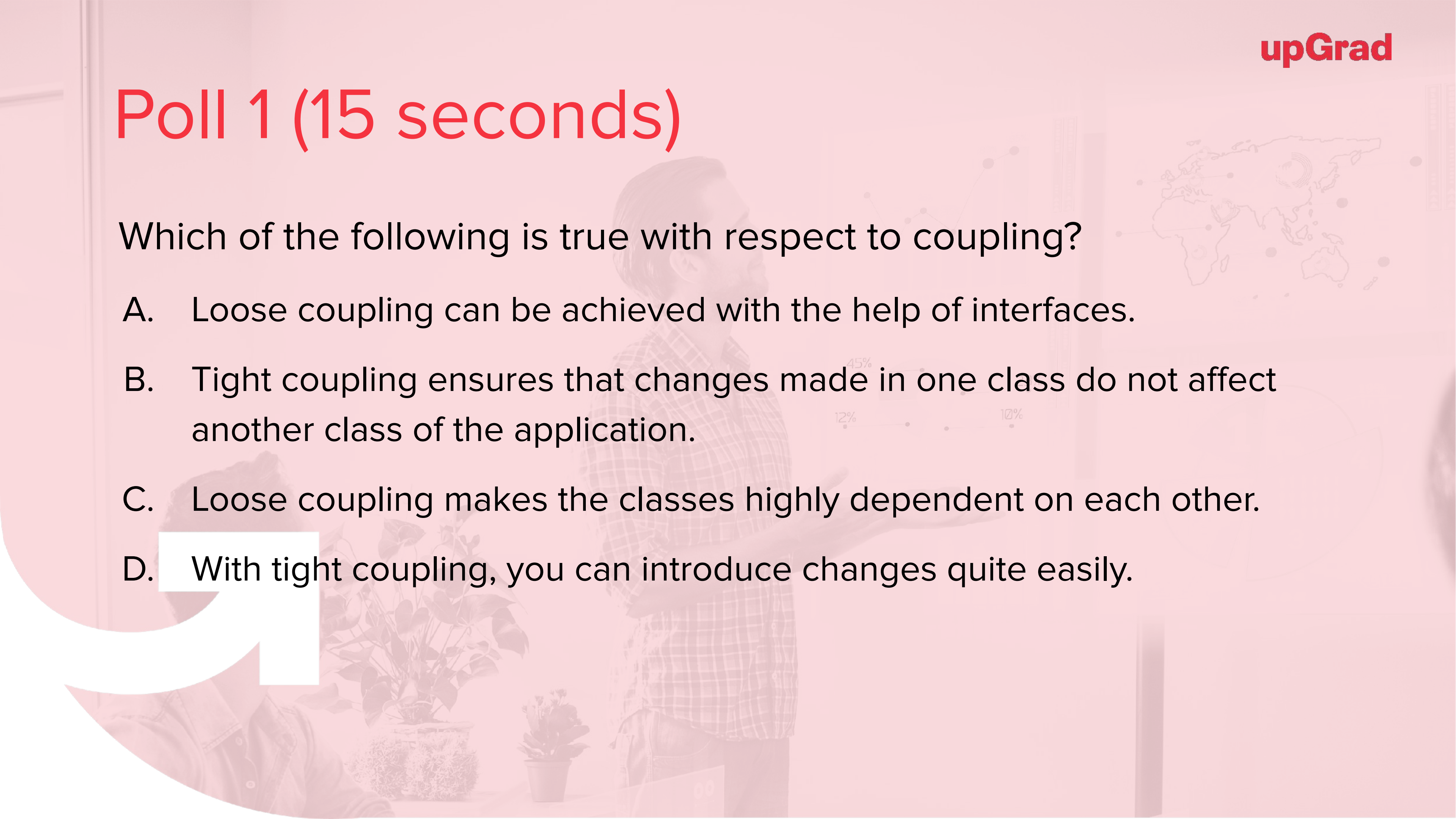
- Now, the **Main** class is **NOT** tightly coupled with its dependencies, the **EnglishGreetingService** class or the **FrenchGreetingService** class.

- You can make changes in the dependency classes without requiring change in the **Main** class.

- You can also create more greeting services and use them into your Main class through GreetingServiceFactory without requiring much code changes in the Main class.

Tight Coupling





Poll 1 (15 seconds)

Which of the following is true with respect to coupling?

A. Loose coupling can be achieved with the help of interfaces.

B. Tight coupling ensures that changes made in one class do not affect another class of the application.

C. Loose coupling makes the classes highly dependent on each other.

D. With tight coupling, you can introduce changes quite easily.

Poll 1 (15 seconds)

Which of the following is true with respect to coupling?

A. Loose coupling can be achieved with the help of interfaces.

B. Tight coupling ensures that changes made in one class do not affect another class of the application.

C. Loose coupling makes the classes highly dependent on each other.

D. With tight coupling, you can introduce changes quite easily.

Coding Exercise

upGrad

Greeting App

Demonstration of tight coupling and loose coupling

Hands-On Experience

Now, let’s create another service for our Greeting App – time service. This service would be used to obtain the current time of the day.

TODO

- Use the following command to check out to the current state. (You can clone the project from here.)

  ```
  git checkout 8f22006
  ```

- Create a **TimeService** interface inside the services package. This interface should contain a method with the following signature:

  ```
  public int getCurrentTime();
  ```

- Create a **TimeService12HourFormat** class inside the services package to return the current hour in the 12-hour format. This class should implement the TimeService interface.

Hands-On Experience

TODO

- You can use the *LocalDateTime.now().getHour()* method that returns the current hour in the 24-hour format.
- Create a **TimeService24HourFormat** class inside the services package to return the current hour in the 24-hour format. This class should implement the TimeService interface.
- Create a **TimeServiceFactory** class inside the factories package to get the required time service.
- Obtain different **TimeService** instances using the **TimeServiceFactory** class inside the main() method of the Main class.
- Execute the application.

Code Reference

Spring Framework

Spring Framework

- In the previous example, you learnt how to make applications loosely coupled by writing extra codes in the form of the factory pattern.

- For big enterprise applications, this boilerplate code can get quite complex and divert us from the main goal of the application.

- This is where the Spring Framework helps you by providing this boilerplate code.

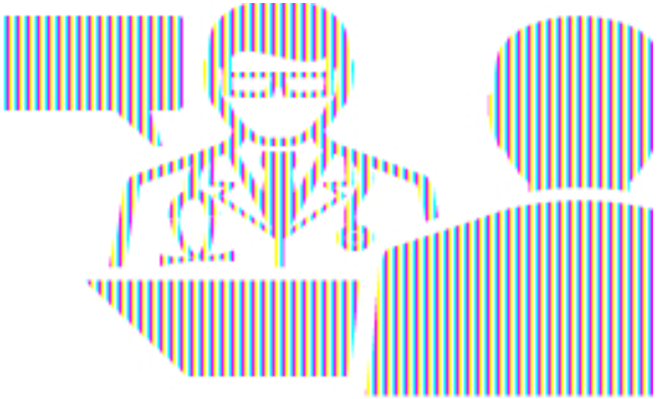
Spring Framework

- A framework, or a software framework, provides you with a well-tested generic boilerplate code that is written by some of the best software developers by following design principles and best practices.

- A framework will not only save time but will also provide the correct components in our applications. This is fast and easy.

- Almost all frameworks, including Spring, rely on the concept of **Inversion of Control** or IoC.

Inversion of Control (IoC)



Normal Flow

Doctor To Patient:

Do this, do that and then go here, etc.

The patient is in control because the doctor is only instructing the course of treatment, and the patient needs to take care of it.

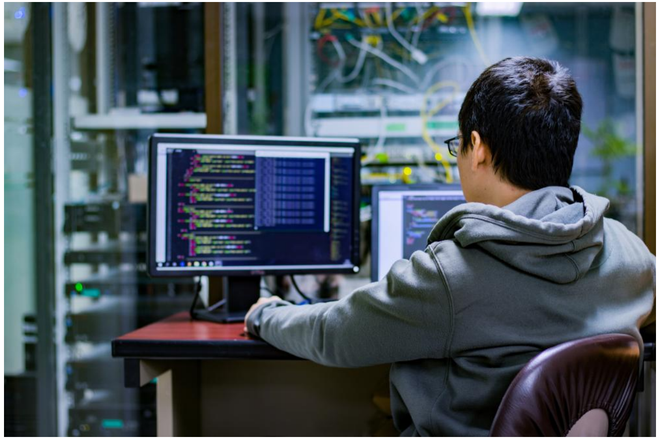
Inverted Flow

Patient To Doctor:

I do not want to know; just do whatever you want to do with me, and help me get rid of this disease!

The doctor is in control because the patient has asked him to take care of the entire course of treatment.

Inversion of Control (IoC)



Normal Flow

In a classical code, the person writing the software instructs the computer 'Do this, do that and then, go here, etc.'

The coder is in control!

Inverted Flow

The coder writing the software fills out a template for the framework, and the framework decides what it will do with it.

The framework is in control!

Inversion of Control (IoC)

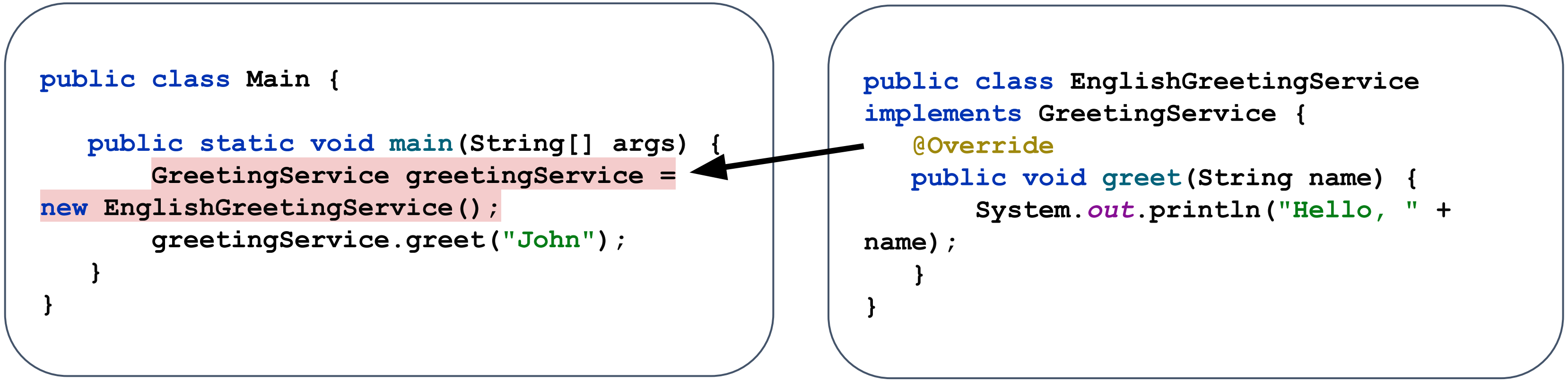
The normal flow of execution is as follows:

- The developer creates a class named ‘EnglishGreetingService’.
- The developer creates an instance of the class ‘EnglishGreetingService’ in the main class.
- The developer invokes a method that is defined in the ‘EnglishGreetingService’ class for execution in the main class.

Inversion of Control (IoC)

Main Class

EnglishGreetingService Class



```
public class Main {

   public static void main(String[] args) {
         GreetingService greetingService =
new EnglishGreetingService();
         greetingService.greet("John");
   }
}
```

```
public class EnglishGreetingService
implements GreetingService {
   @Override
   public void greet(String name) {
         System.out.println("Hello, " +
name);
   }
}
```

Dependent Class

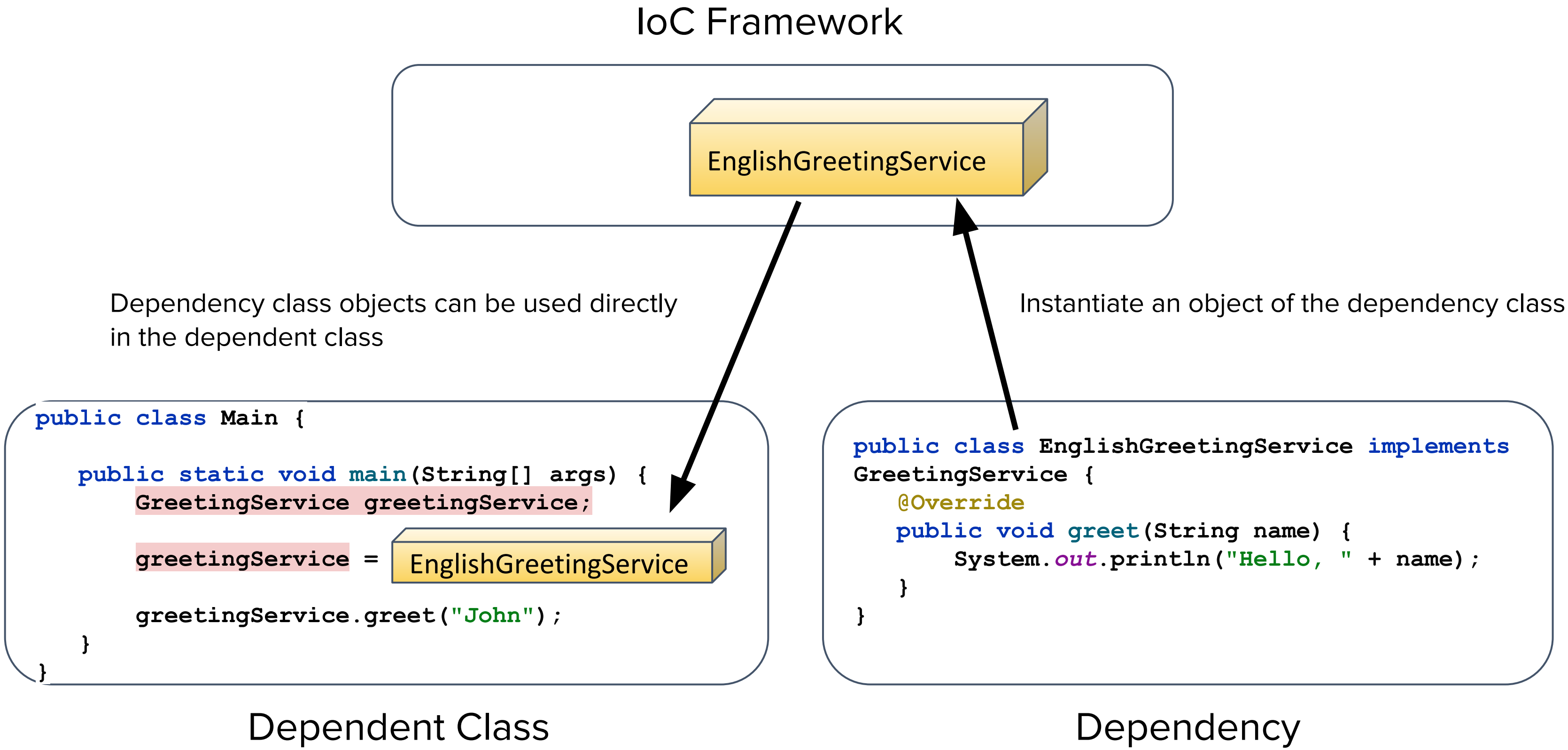
Dependency

Inversion of Control (IoC)

Flow of execution when using the IoC Framework

- The developer creates a class named ‘EnglishGreetingService’.
- **The IoC Framework** creates an instance of ‘EnglishGreetingService’ that can be used inside the main class.
- The developer invokes a method that is defined in the ‘EnglishGreetingService’ class for execution in the main class.

Inversion of Control (IoC)



IoC Framework

Dependency class objects can be used directly in the dependent class

Instantiate an object of the dependency class

```java
public class Main {

   public static void main(String[] args) {
       GreetingService greetingService;

       greetingService = EnglishGreetingService

       greetingService.greet("John");
   }
}
```

```java
public class EnglishGreetingService implements
GreetingService {
   @Override
   public void greet(String name) {
       System.out.println("Hello, " + name);
   }
}
```

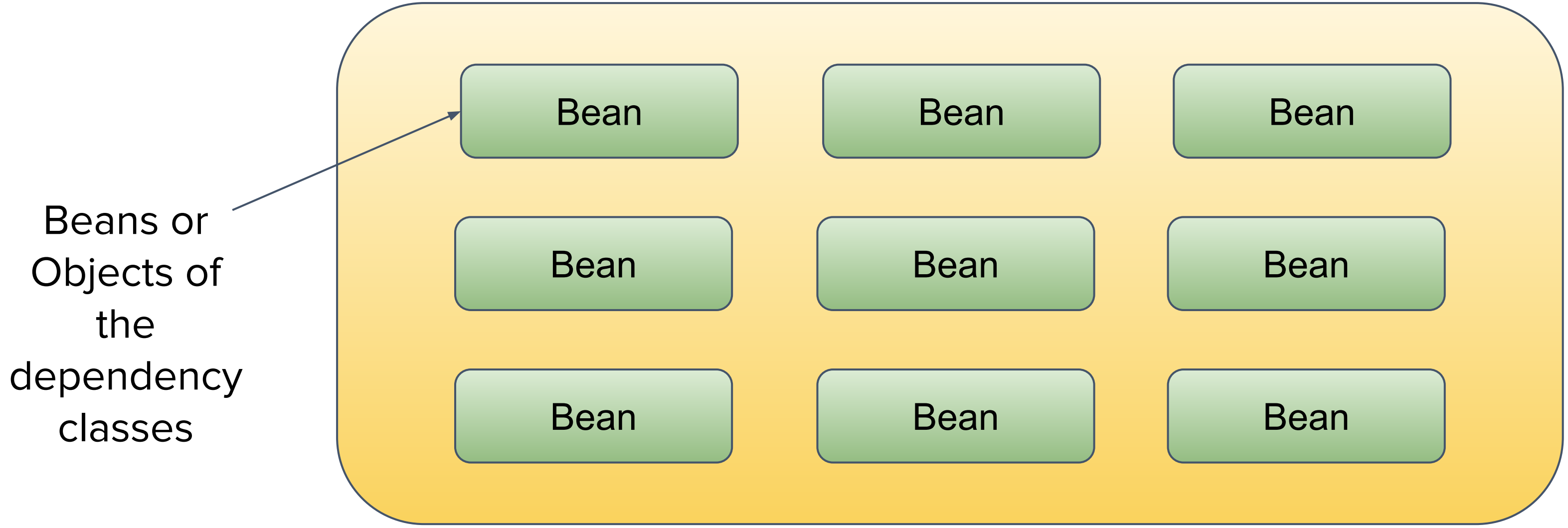
Dependent Class

Dependency

Spring Container

- As you saw in the previous example, the IoC Framework instantiates objects of the dependency classes that can be used by the dependent classes.
- The IoC Frameworks behave like containers for such objects, which is why IoC Frameworks are also called **IoC Containers.**
- The IoC Container of the Spring Framework is called, for obvious reasons, the **Spring Container.**
- In Spring, the dependency objects contained inside the Spring Container are called **Spring Beans** or, simply, beans.

Spring Container



Spring Container

Spring Container

- Spring provides the following two distinct types of containers:
    - Spring BeanFactory Container
    - Spring ApplicationContext Container

Spring Container (IoC)

BeanFactory

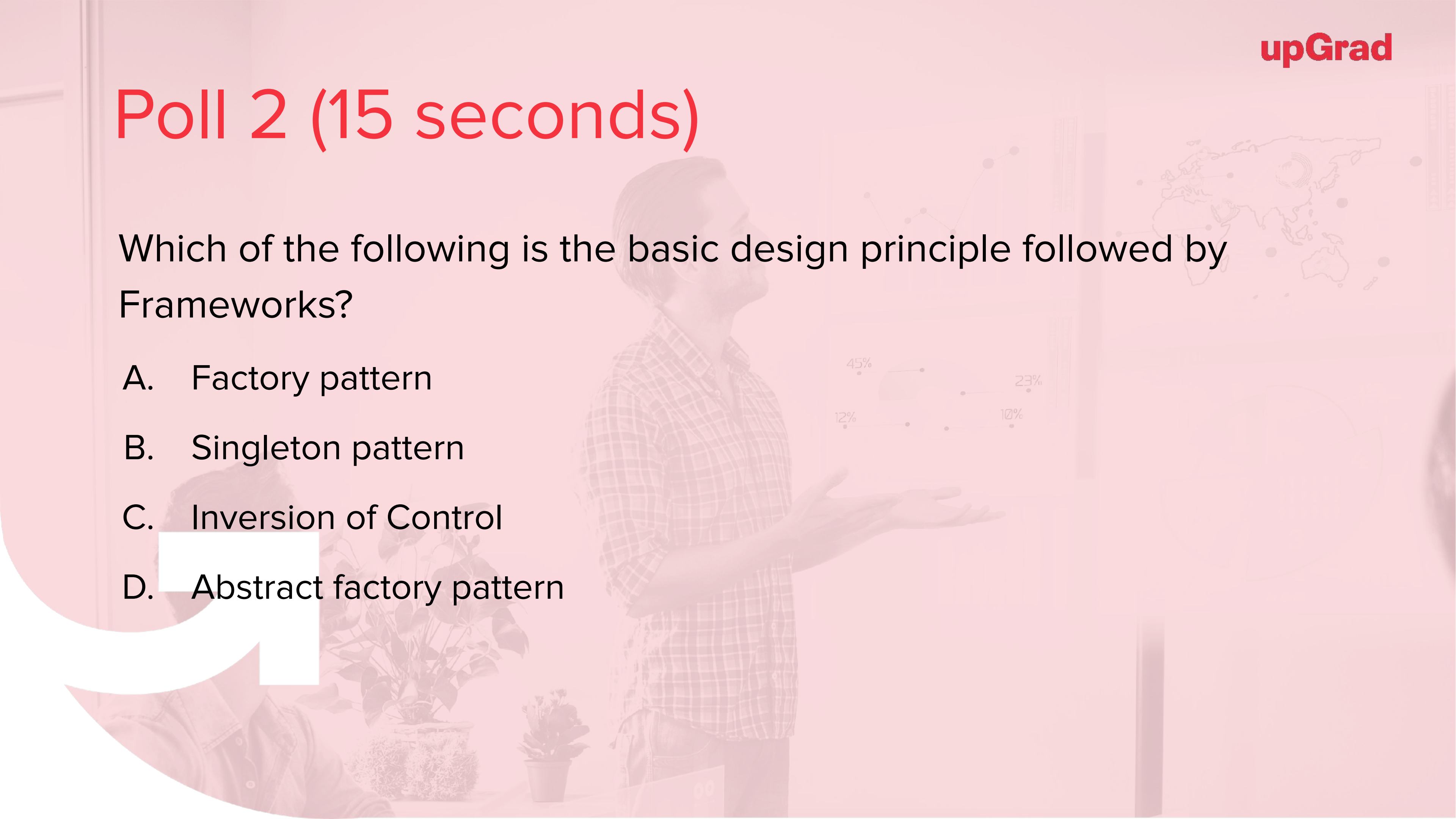
- It is similar to a factory class that contains a collection of methods to get beans from the Spring Container.
- It is lightweight, as it instantiates beans only when requested by the application instead of creating all the beans at the start of the application.
- Being lightweight, it is generally used for mobile device applications.

Spring Container (IoC)

ApplicationContext

- It is an interface that extends the BeanFactory interface.
- ApplicationContext is heavyweight compared with BeanFactory, as it loads all the beans at the startup.
- It provides some extra enterprise-specific functionality on top of the BeanFactory container.
- Since it includes all the functionalities of the BeanFactory Container and also provides some extra functionalities, it is generally recommended over BeanFactory wherein the startup time is not an issue, such as writing the server-side code.

Poll 2 (15 seconds)



Which of the following is the basic design principle followed by Frameworks?

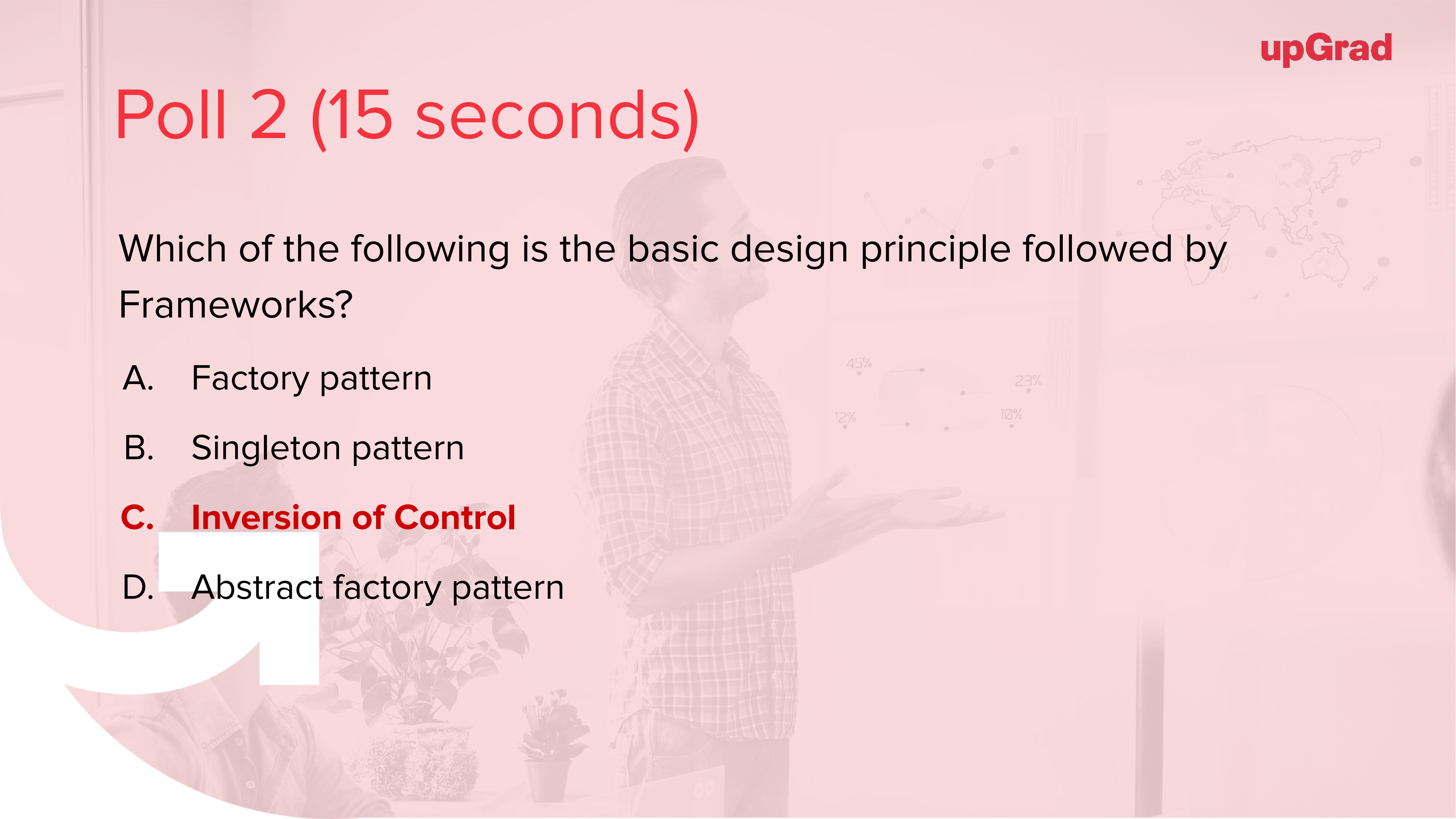
A. Factory pattern

B. Singleton pattern

C. Inversion of Control

D. Abstract factory pattern

Poll 2 (15 seconds)



Which of the following is the basic design principle followed by Frameworks?

A. Factory pattern

B. Singleton pattern

C. Inversion of Control

D. Abstract factory pattern

Bean Creation in Spring

Bean Creation

- So far, you learnt that Spring creates beans (or instantiates objects of the dependency classes) and maintains them inside the Spring Container.
- **How does Spring know which ones are dependency classes?**

Bean Creation

- Developers write dependency classes; so, they need to tell the Spring about the classes that should be treated as dependency classes.

- This information can be provided to the Spring in the form of metadata (information about your code).

- There are three ways to provide metadata to the Spring, which are as follows:
  a. **XML-based configuration**: We provide an XML file in which all the dependency classes listed.
  b. **Java-based configuration**: We provide a type of factory class that contains methods that return the object of the dependency classes. These methods are executed by the Spring Container to instantiate dependency classes. This factory class needs to be marked with the **@Configuration** annotation.

Is there any other way to provide metadata about your code? Recall your learnings from the Java course.

c. Annotation-Based Configuration

- Recall your learnings of Annotations in the Java course.
- Annotations are a way to provide metadata about your code.
- Annotations are similar to comments, not for other developers but for the tools that process your source code; here, that tool is the Spring Framework.
- In our application, we will use annotation-based configuration because it helps you provide shorter and more concise configuration.

@Component

upGrad

- To mark a class as a dependency class, it should be annotated with the @Component annotation.

- Let’s start using the Spring framework to build a loosely coupled application without writing the boilerplate code.

Spring Framework

1. Download the Spring jar (version 5.2.5) from the Spring repository present at the following link: https://repo.spring.io/release/org/springframework/spring/

2. Add the following jars to the classpath:
   a. *spring-core* and *spring-beans*: Provides IoC features to the Spring Container
   b. *spring-context*: Provides the Spring container
   c. *spring-aop* and *spring-expression*: Will discuss later in the course
3. Download the Apache Common Logging library from the following URL and add it to the classpath. https://commons.apache.org/proper/commons-logging/download_logging.cgi

Spring Framework

4. Mark the EnglishGreetingService class with the @Component annotation.

```
@Component
public class EnglishGreetingService implements GreetingService {
    @Override
    public void greet(String name) {
        System.out.println("Hello, " + name);
    }
}
```

The Spring Container will create a bean by instantiating the EnglishGreetingService class. The name of the bean would be the same as the class name, with the first letter in lower case. Thus, the bean name would be ‘englishGreetingService’.

Spring Framework

5. Replace the GreetingServiceFactory with ApplicationContext.

```
GreetingServiceFactory greetingServiceFactory =
       new GreetingServiceFactory();
GreetingService greetingService =
       greetingServiceFactory.getGreetingService("english");
```

```
ApplicationContext context =
       new AnnotationConfigApplicationContext("com.upgrad.greeting");
GreetingService greetingService =
       (GreetingService)context.getBean("englishGreetingService");
```

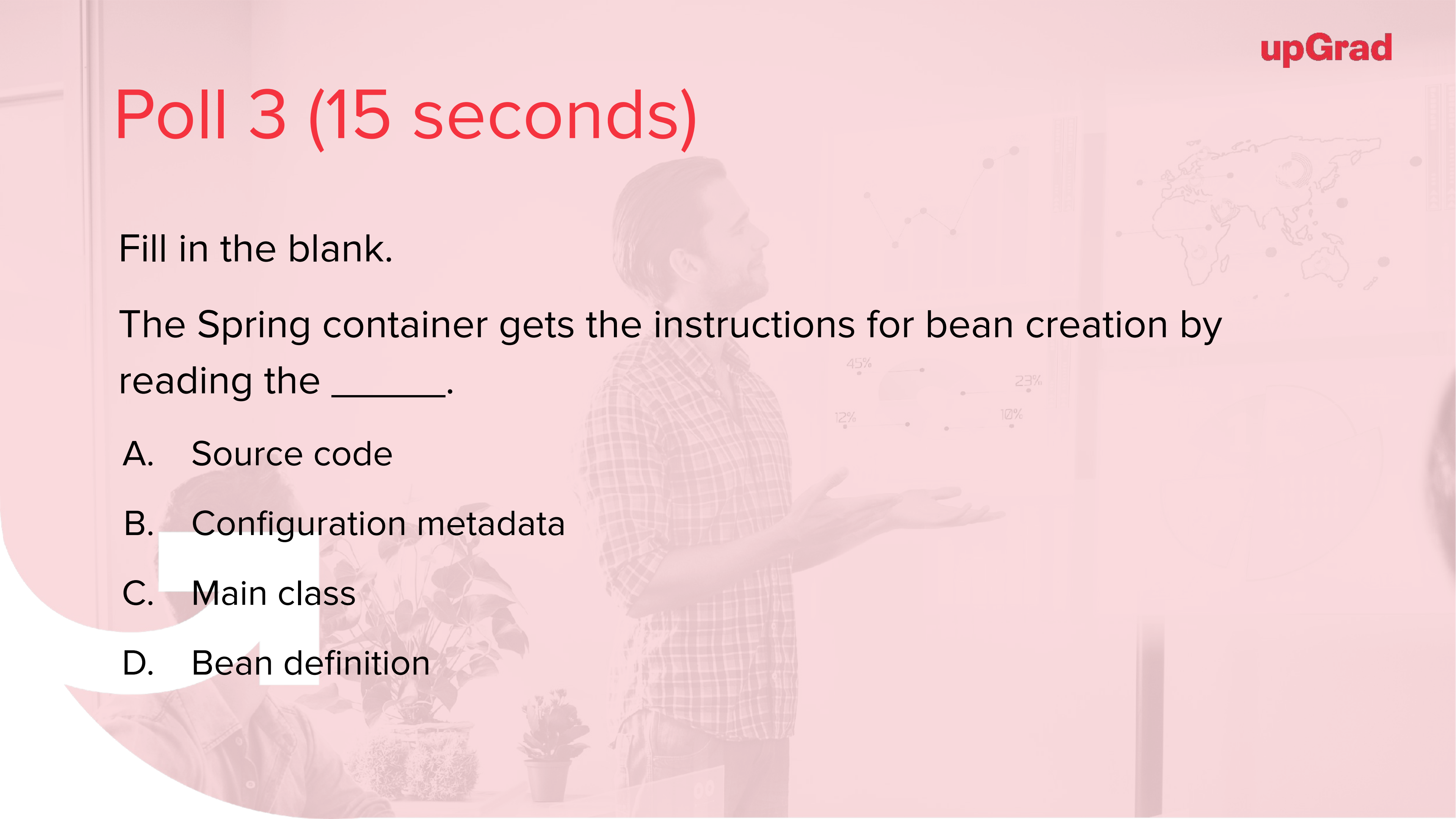
The name of the base package that, along with its subpackages, would search for the dependency classes (classes that are annotated with the @Component annotation)

Spring Framework

6. Run the code. You will notice that your application is working as it was before.
7. Mark the FrenchGreetingService with the @Component annotation, and use this dependency in the Main class for greeting.
8. Run the application. Now, your application would greet in French.
9. Now, there is no need to write the boilerplate code. You can delete the GreetingServiceFactory class.
10. You learnt how to create loosely coupled applications using the Spring framework.

Code Reference

Poll 3 (15 seconds)



Fill in the blank.

The Spring container gets the instructions for bean creation by reading the ______.

A. Source code

B. Configuration metadata

C. Main class

D. Bean definition

Poll 3 (15 seconds)

Fill in the blank.

The Spring container gets the instructions for bean creation by reading the ______.

A. Source code

B. Configuration metadata

C. Main class

D. Bean definition

Poll 4 (15 seconds)

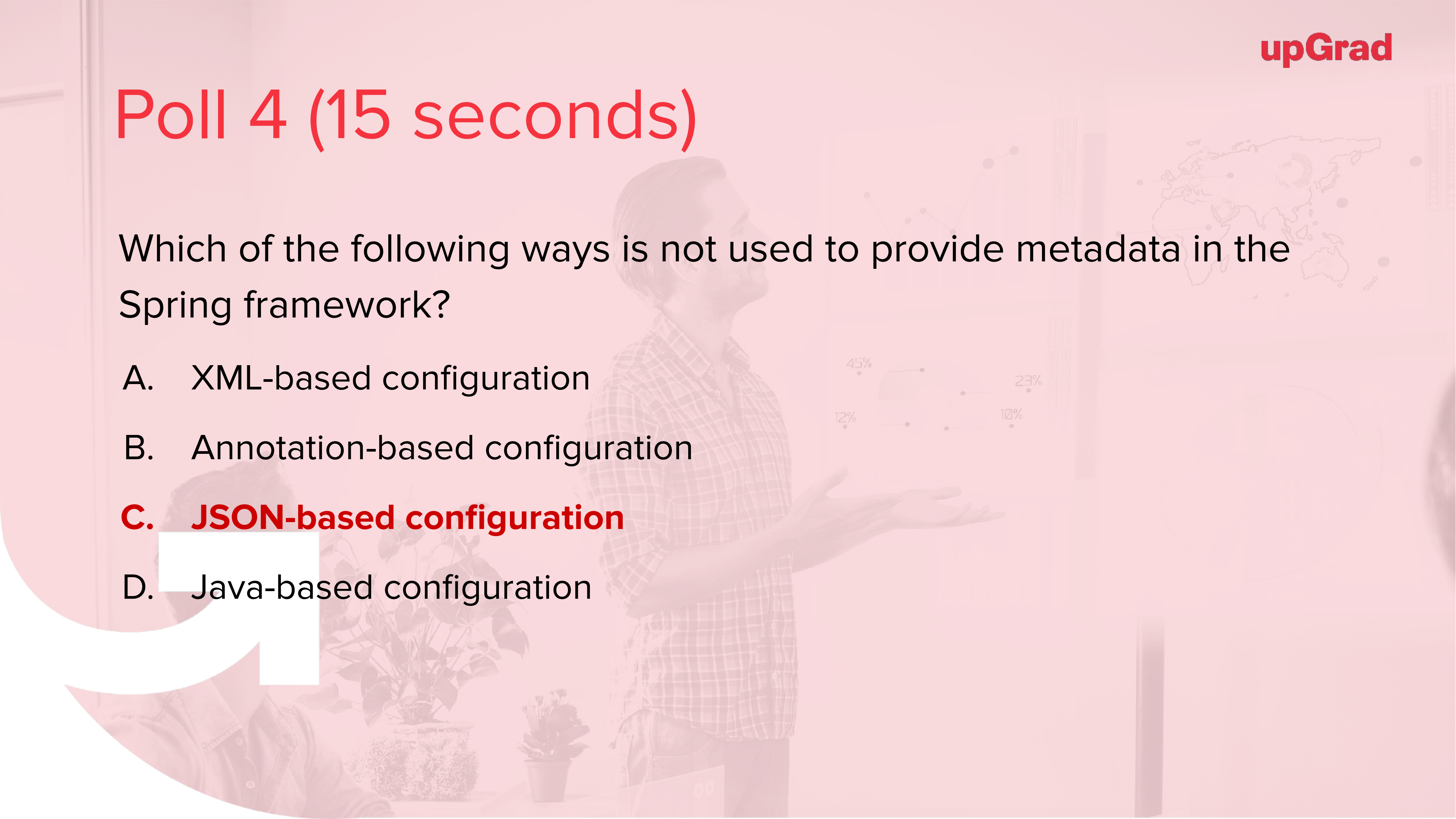
Which of the following ways is not used to provide metadata in the Spring framework?

A. XML-based configuration

B. Annotation-based configuration

C. JSON-based configuration

D. Java-based configuration



Poll 4 (15 seconds)

Which of the following ways is not used to provide metadata in the Spring framework?

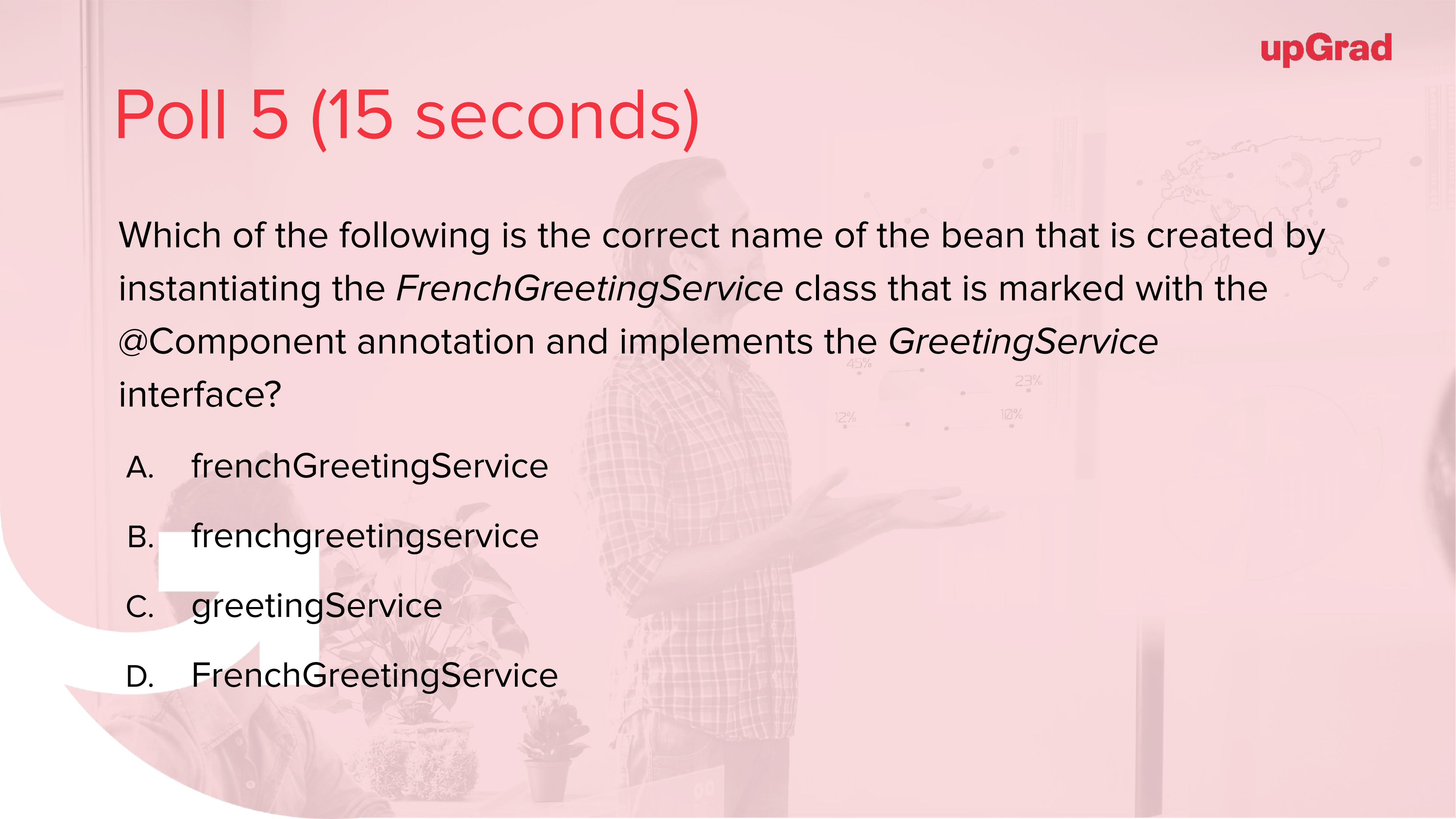
A. XML-based configuration

B. Annotation-based configuration

C. JSON-based configuration

D. Java-based configuration

Poll 5 (15 seconds)



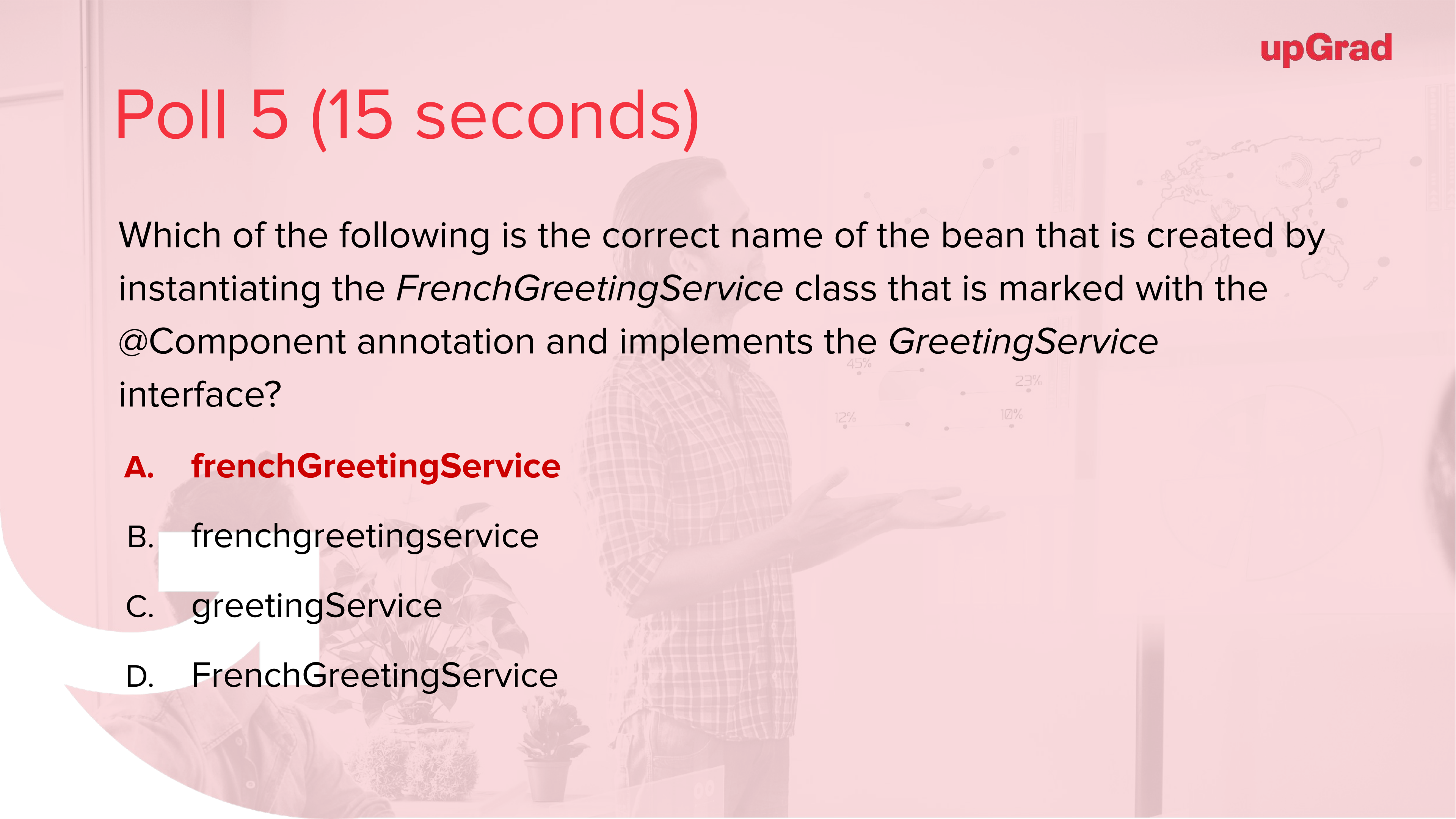
Which of the following is the correct name of the bean that is created by instantiating the *FrenchGreetingService* class that is marked with the @Component annotation and implements the *GreetingService* interface?

A. frenchGreetingService

B. frenchgreetingservice

C. greetingService

D. FrenchGreetingService



Poll 5 (15 seconds)

Which of the following is the correct name of the bean that is created by instantiating the *FrenchGreetingService* class that is marked with the @Component annotation and implements the *GreetingService* interface?

A. frenchGreetingService

B. frenchgreetingservice

C. greetingService

D. FrenchGreetingService

Coding Exercise

GreetingApp

Demonstration of Bean Creation Using the @Component Annotation and ApplicationContext

Hands-On Experience

Now, let's mark time services with the @Component annotation so that we can use these services inside our Greeting App in a loosely coupled manner using the Spring Framework. We will also delete the boilerplate code that we wrote.

TODO

- Use the following command to check out to the current state:

  ```
  git checkout 2f7ef2c
  ```

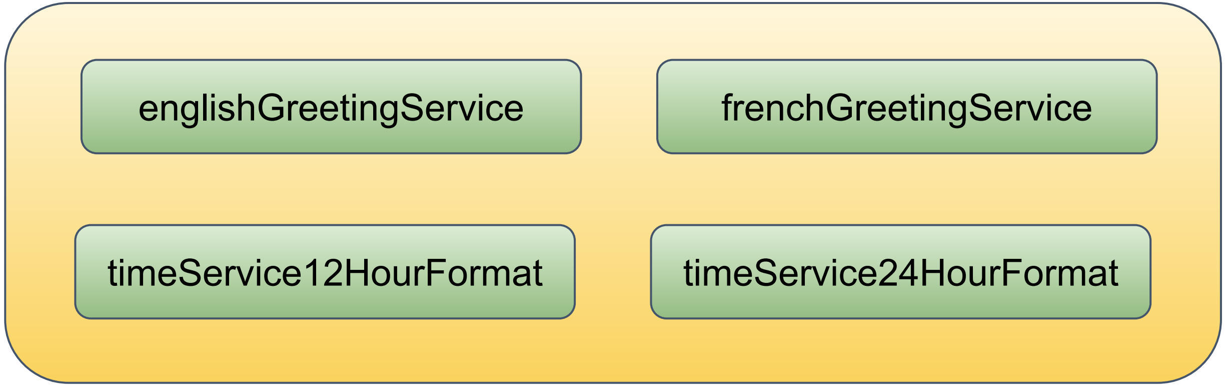
- Download the required jars and put them into the classpath.
- Annotate both time service dependency classes (TimeService12HourFormat and TimeService24HourFormat) with the @Component annotation.
- Get the beans of these services inside the Main class and run your application.
- Delete the TimeServiceFactory class along with the 'factories' package.

Code Reference

Dependency Injection

Dependency Injection

- So far, you have marked all four services with the @Component annotation.
- When the Spring Container loads at the start of the application, it will contain four beans as shown below.



Spring Container

Dependency Injection

- What if beans are themselves dependent on other beans?
- For example, suppose you want to greet people based on the time of the day.
- In that case, greeting service would depend on the time service.

Note: *For now, let's not have the TimeService12HourFormat as a Spring bean because having two beans of the same type (implementing the same interface) as a dependency for other beans requires understanding of some more annotations, which will be covered later in the session.*

Dependency Injection

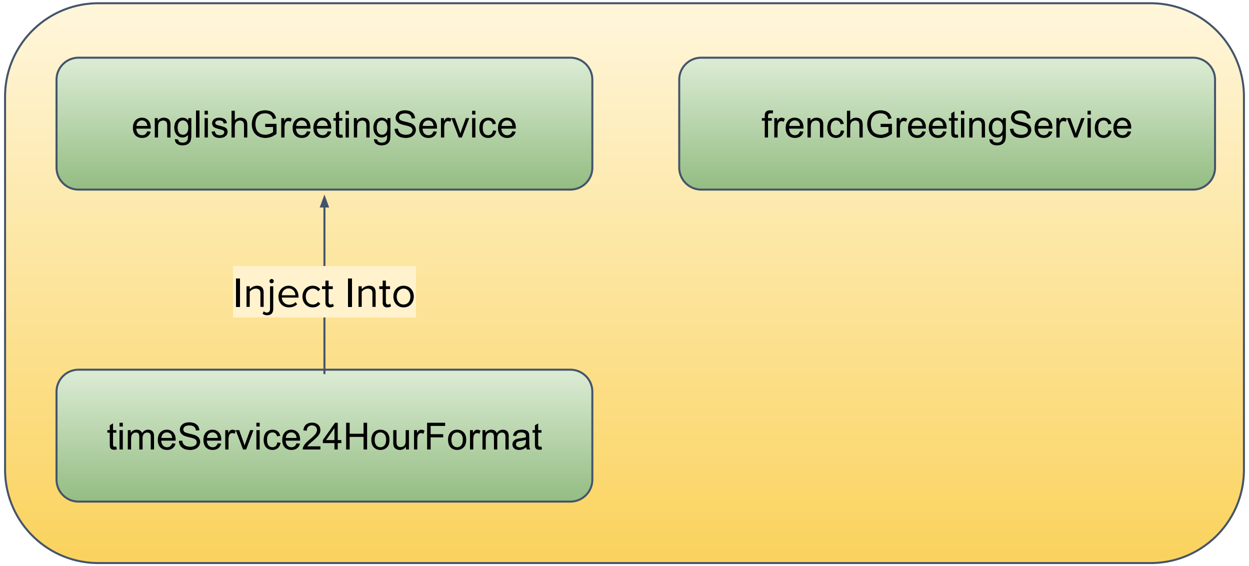
```java
@Component
public class EnglishGreetingService implements GreetingService {

   private TimeService timeService;

   @Override
   public void greet(String name) {
       //implementation goes here
   }
}
```

```java
@Component
public class TimeService24HourFormat implements TimeService {
   @Override
   public int getCurrentTime() {
       return LocalDateTime.now().getHour();
   }
}
```

Dependency Injection

- As you can see in the previous example, one bean is dependent on the other.
- So, when the Spring Container loads, there would be three beans, where one bean, EnglishGreetingService, is dependent on another one, TimeService24HourFormat.
- **How to instruct the Spring Container to inject one bean into another?**

Dependency Injection



Spring Container

Dependency Injection

- This is where the dependency injection (DI) helps you to inject dependencies into an object.

- IoC is a generic programming principle, whereas DI is an implementation of IoC.

- **IoC** states that dependent classes should not create objects of the dependency classes. Dependency objects would be created by the IoC container or Framework and provided to the dependent object.

- **IoC** does not answer how dependency objects will be provided to the dependent objects.

Dependency Injection

- **DI** is one of the implementation or a subtype of IoC and instructs how to provide dependency objects to the dependent objects.

- **DI** states that dependencies would be injected into the dependent objects via the setter method or a constructor.

- In Spring, you can also inject dependencies via a field or a property injection. So, in Spring, you can perform DI in the following three ways:
    - Field or property injection
    - Setter injection
    - Constructor injection

@Autowired Annotation

- In Spring, DI is performed using the @Autowired annotation.

- Since DI can be done in three ways in Spring, the @Autowired annotation can be applied over the following:
  - Fields or properties
  - Setter methods
  - Constructors

- Let’s learn how to perform DI using the @Autowired annotation to inject a bean of TimeService24HourFormat into EnglishGreetingService.

Field/Property Injection

```
@Component
public class EnglishGreetingService implements GreetingService {

   @Autowired
   private TimeService timeService;

   @Override
   public void greet(String name) {
       //Implementation goes here
   }
}
```

Setter Injection

```java
@Component
public class EnglishGreetingService implements GreetingService {

    private TimeService timeService;

    @Autowired
    public void setTimeService(TimeService timeService) {
        this.timeService = timeService;
    }

    @Override
    public void greet(String name) {
        //Implementation goes here
    }
}
```

Constructor Injection

```
@Component
public class EnglishGreetingService implements GreetingService {

   private TimeService timeService;

   @Autowired
   public EnglishGreetingService(TimeService timeService) {
       this.timeService = timeService;
   }

   @Override
   public void greet(String name) {
       //Implementation goes here
   }
}
```

Code Reference

@Autowired Annotation

Which method of DI should be preferred for our application?

- Property and setter injection
  - To inject those dependencies that are not critical for the dependent object to perform its functionalities
  - To inject those dependencies that may change over the course of time
- Constructor injection
  - As you cannot instantiate a class without providing all the arguments to the constructor, the constructor injection is used to inject those dependencies that are critical for the dependent object to perform its functionalities.

Poll 6 (15 seconds)

Suppose you want to inject a bean of ServiceX into ServiceY. Which of the following steps needs to be followed to achieve this? (Note: More than one option may be correct.)

A. ServiceX has to be marked with the @Component annotation.

B. ServiceY has to be marked with the @Component annotation.

C. ServiceX will have a property of the type ServiceY with the @Autowired annotation.

D. ServiceY will have a property of the type ServiceX with the @Autowired annotation.

Poll 6 (15 seconds)

Suppose you want to inject a bean of ServiceX into ServiceY. Which of the following steps needs to be followed to achieve this? (Note: More than one option may be correct.)

A. ServiceX has to be marked with the @Component annotation.

B. ServiceY has to be marked with the @Component annotation.

C. ServiceX will have a property of the type ServiceY with the @Autowired annotation.

D. ServiceY will have a property of the type ServiceX with the @Autowired annotation.

Coding Exercise

GreetingApp

Demonstration of the @Autowired Annotation
for the Constructor, Field and Setter

Hands-On Experience

Now, let’s inject time service inside the FrenchGreetingService to greet in French based on the current time. In French, good morning and good afternoon is ‘bonjour’, and good evening is ‘bonsoir’.

TODO

- Use the following command to check out to the current state:

  ```
  git checkout 3b99ab6
  ```

- Inject the time service into FrenchGreetingService using the @Autowired annotation.
- Get a bean of FrenchGreetingService inside the Main class and greet people in French.
- Try different ways of DI while injecting the time service into FrenchGreetingService.

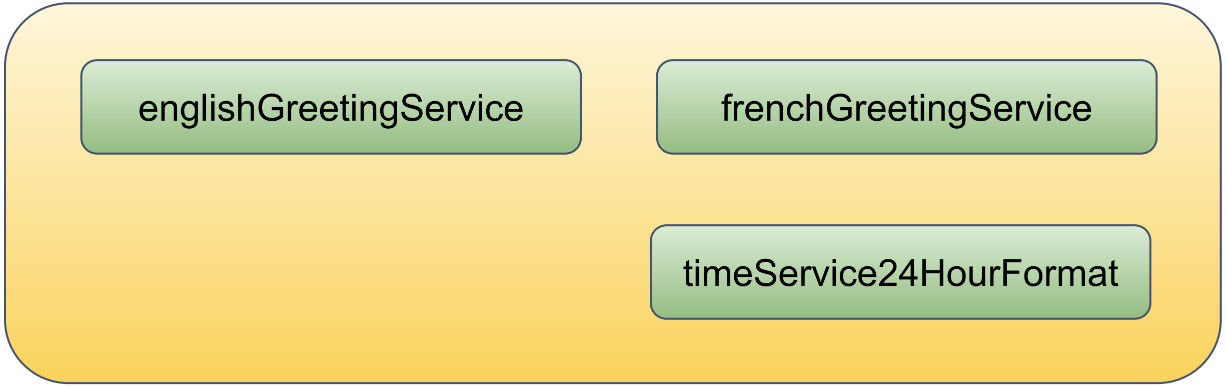
Code Reference

Dependency Injection Using @Qualifier Annotation

DI Using the @Qualifier Annotation

upGrad

- When you try to inject a dependency using the @Autowired annotation, the Spring Container checks all the beans inside it and injects that bean whose type matches with the type of the requested bean.
- For example, consider that the Spring container contains the following three beans:



DI Using the @Qualifier Annotation

- Now, suppose you try to inject the time service bean into the greeting service using the @Autowired annotation.

```
@Autowired
private TimeService timeService;
```

- In this case, Spring will check all the beans inside the Spring Container and inject the timeService24HourFormat bean into the greeting service because this is the only bean whose type matches with that of the requested bean (as the timeService24HourFormat implements the TimeService interface). This is called the **byType** injection.

- What if there is more than one bean of the requested type? Let's mark TimeService12HourFormat with the @Component annotation and observe what happens.

DI Using the @Qualifier Annotation

- When there is more than one bean present of the requested type, the Spring Container would throw the following exception:

  org.springframework.beans.factory.**NoUniqueBeanDefinitionException**: No qualifying bean of type 'com.upgrad.greeting.services.TimeService' available: expected single matching bean but found 2: timeService12HourFormat,timeService24HourFormat

- How to instruct the Spring the bean that is to be injected into the greeting service?
- This is where the @Qualifier annotation helps to specify the bean that is to be injected using the bean name. This is called the **byName** injection.

DI Using the @Qualifier Annotation

- Code snippet for property injection:

```
@Autowired
@Qualifier("timeService24HourFormat")
private TimeService timeService;
```

DI Using @Qualifier Annotation

- Code snippet for setter injection:

```
@Autowired
@Qualifier("timeService24HourFormat")
public void setTimeService(TimeService timeService) {
   this.timeService = timeService;
}
```

DI Using @Qualifier Annotation

- Code snippet for constructor injection:

```
@Autowired
public EnglishGreetingService(@Qualifier("timeService24HourFormat")
TimeService timeService) {
   this.timeService = timeService;
}
```

Code Reference

Poll 7 (15 seconds)

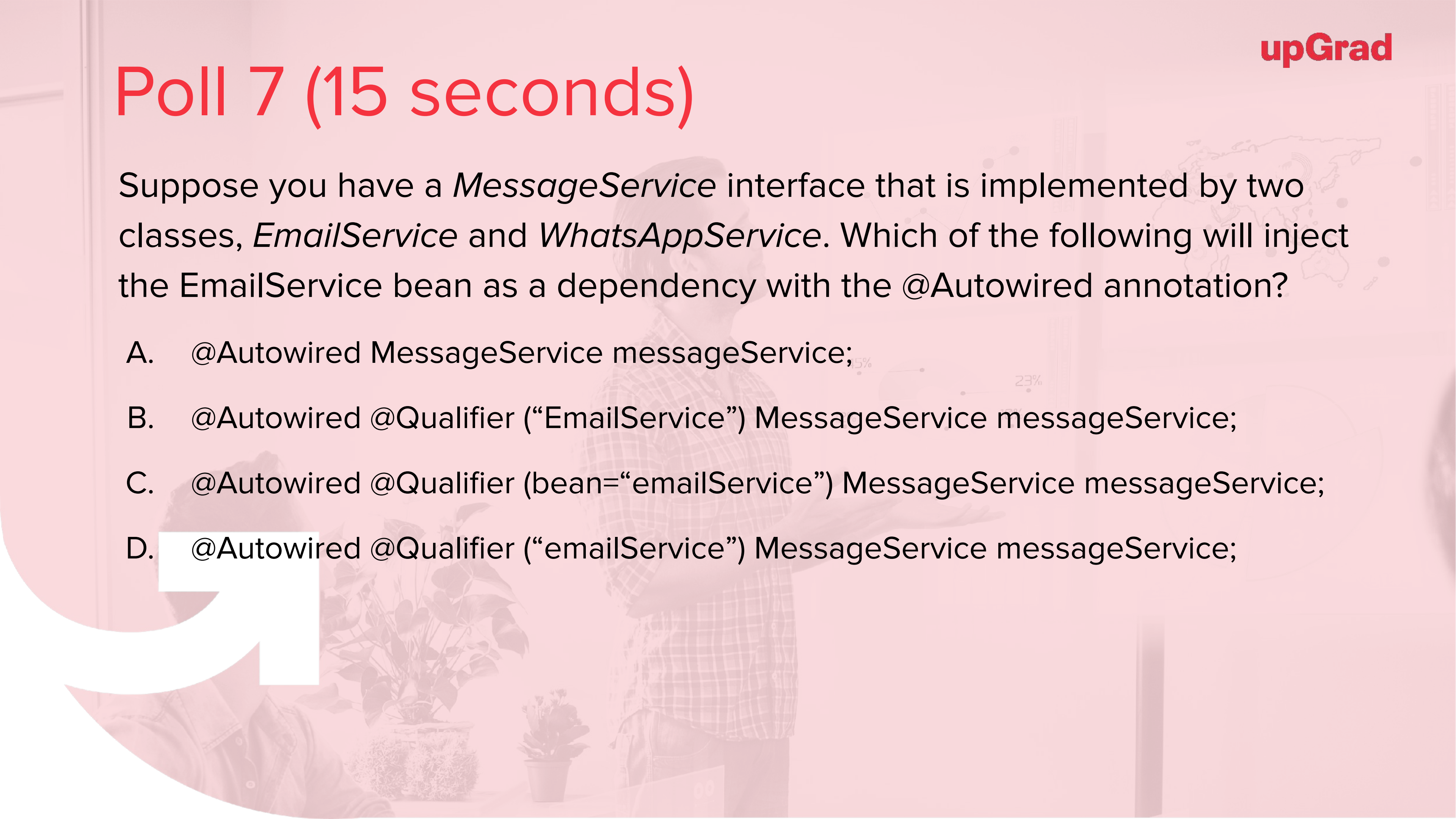
Suppose you have a *MessageService* interface that is implemented by two classes, *EmailService* and *WhatsAppService*. Which of the following will inject the EmailService bean as a dependency with the @Autowired annotation?

A. @Autowired MessageService messageService;

B. @Autowired @Qualifier (“EmailService”) MessageService messageService;

C. @Autowired @Qualifier (bean=“emailService”) MessageService messageService;

D. @Autowired @Qualifier (“emailService”) MessageService messageService;



upGrad

Poll 7 (15 seconds)

Suppose you have a *MessageService* interface that is implemented by two classes, *EmailService* and *WhatsAppService*. Which of the following will inject the EmailService bean as a dependency with the @Autowired annotation?

A. @Autowired MessageService messageService;

B. @Autowired @Qualifier ("EmailService") MessageService messageService;

C. @Autowired @Qualifier (bean="emailService") MessageService messageService;

D. @Autowired @Qualifier ("emailService") MessageService messageService;

Hands-On Experience

Now, let’s inject the time service inside the FrenchGreetingService using the @Qualifier annotation.

TODO

- Use the following command to check out to the current state:

  git checkout dac9c3e

- Inject the TimeService24HourFormat into FrenchGreetingService using the @Autowired and @Qualifier annotations.

- Get the bean of FrenchGreetingService inside the Main class and greet people in French.

- Try different ways of DI while injecting the time service into FrenchGreetingService.

Code Reference

Spring Bean Scope

Spring Bean Scope

- When you mark a class with the @Component annotation, the Spring Container will treat that class as a dependency and instantiate that class to create a bean.
- By default, Spring creates one bean per dependency class, but you can control that using the @Scope annotation.

Spring Bean Scope

- The Spring framework supports the following five types of bean scopes. Out of these, three are applicable only in the case of a web-aware ApplicationContext:
    - *Singleton*
    - *Prototype*
    - *Request*
    - *Session*
    - *Global-session*

  Note: In most of the cases, you will only deal with a singleton or a prototype bean scope.

Spring Bean Scope - Singleton

- Is the **default** scope of Spring bean inside the Spring IoC container
- Is best described as **One Bean Per @Component Class Per IoC container**
- The Spring IoC container gives the **same bean reference** every time when a request is made for a bean using the getBean() method or when there arises a need to inject a bean using the @Autowired annotation.
- This scope is used for stateless beans (which do not maintain any state).

Spring Bean Scope - Singleton

```
@Component
@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
public class EnglishGreetingService implements GreetingService {

}


ApplicationContext context = new AnnotationConfigApplicationContext("com.upgrad.greeting");
GreetingService greetingService1 = (GreetingService)context.getBean("englishGreetingService");
GreetingService greetingService2 = (GreetingService)context.getBean("englishGreetingService");

System.out.println(greetingService1 == greetingService2);
```

The output of the program would be “true”.

Code Reference

Spring Bean Scope - Prototype

- The Spring IoC container gives a **new bean instance reference** every time a request is made for bean using the getBean() method or when there arises a need to inject a bean using the @Autowired annotation.
- Thus, a single bean definition of the Prototype scope can have any number of object instances.
- This scope is used for stateful beans (that maintain the state).

Spring Bean Scope - Singleton

```java
@Component
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class EnglishGreetingService implements GreetingService {

}


ApplicationContext context = new AnnotationConfigApplicationContext("com.upgrad.greeting");
GreetingService greetingService1 = (GreetingService)context.getBean("englishGreetingService");
GreetingService greetingService2 = (GreetingService)context.getBean("englishGreetingService");

System.out.println(greetingService1 == greetingService2);
```

The output of the program would be “false”.

Code Reference

Spring Bean Scope

The other three scopes are available only for web-based applications, which are as follows:

- **request:** The Spring Container gives a single bean instance per HTTP request.
- **session:** The Spring Container gives a single bean instance per HTTP session (user-level session).
- **global-session:** The Spring Container gives a single bean instance per global HTTP session (application-level session).

Poll 8 (15 seconds)

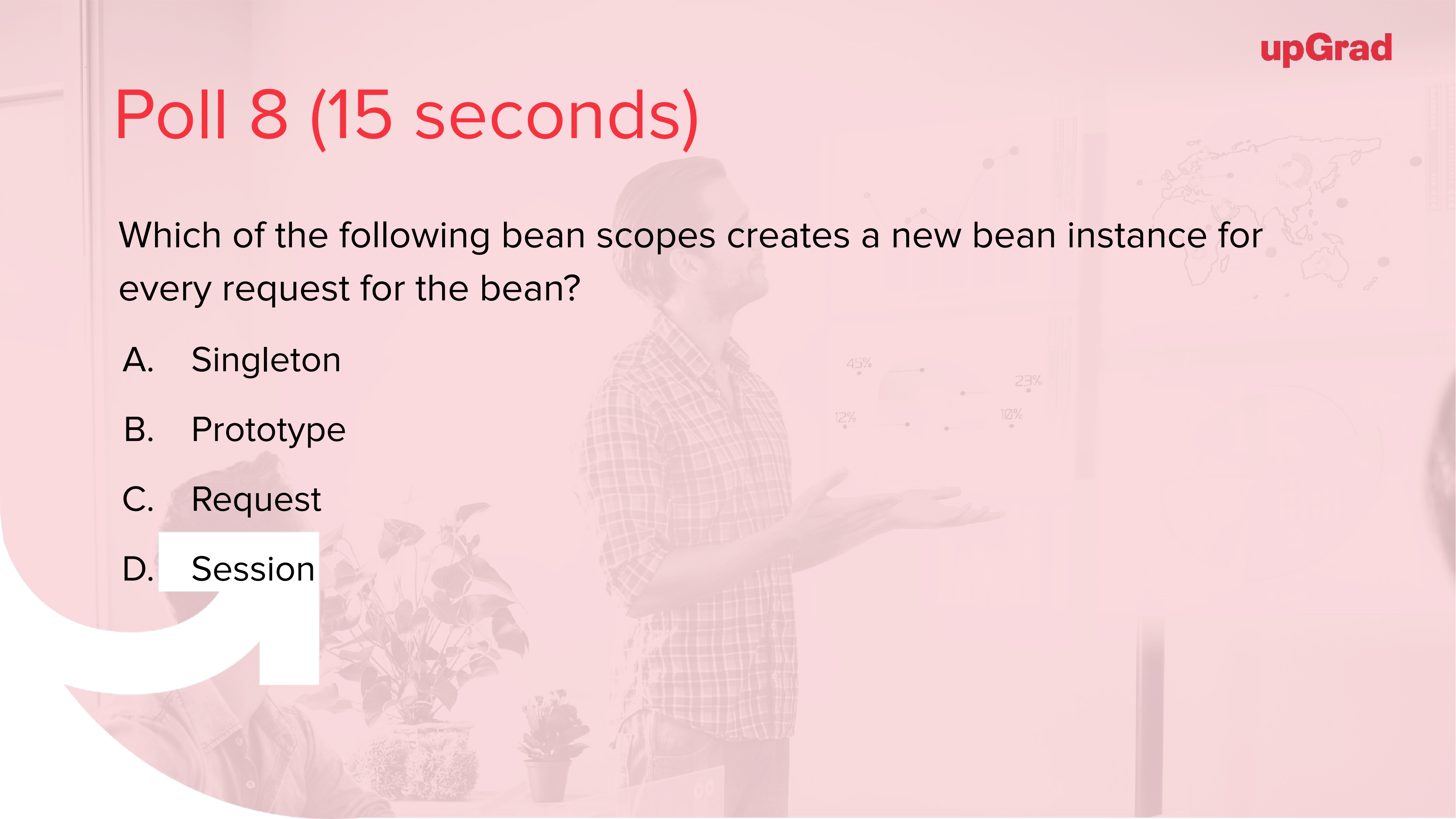
Which of the following bean scopes creates a new bean instance for every request for the bean?

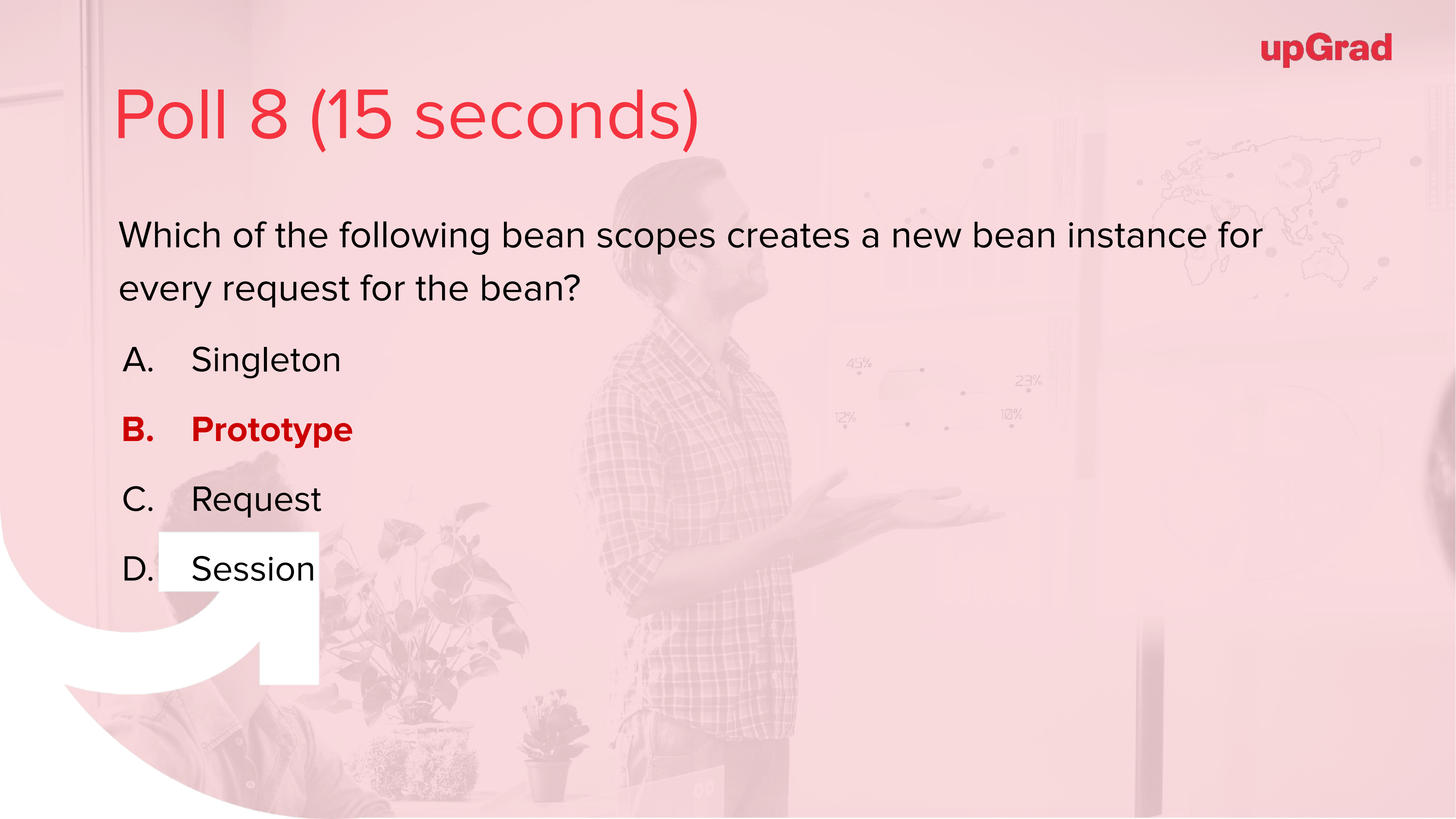
A. Singleton

B. Prototype

C. Request

D. Session





Poll 8 (15 seconds)

Which of the following bean scopes creates a new bean instance for every request for the bean?

A. Singleton

B. Prototype

C. Request

D. Session

upGrad

Maven Build Tool

Maven

- When you were developing our Greeting App, you had to download the Spring jars and Apache commons jar from their official websites and add them to the classpath of the project.
- For such small projects, we do not need more than 5–7 jars, but for big enterprise applications, there may arise a need for hundreds of jars.
- Downloading these hundreds of jars from different websites and adding them to the classpath can be a quite tedious task.
- This is where Maven helps you.
- You only need to provide a list of dependencies that are needed for your application to the Maven, and it will download and add them to the classpath.

Maven

upGrad

What is Maven?

- Maven is a software application build tool.
- It is primarily used to download jar files.
- Other uses of Maven are as follows:
  - It provides a standard project structure.
  - It helps you in compiling, building and deploying a software.
  - It also helps you in generating the documentation for the source code.

API Plugin

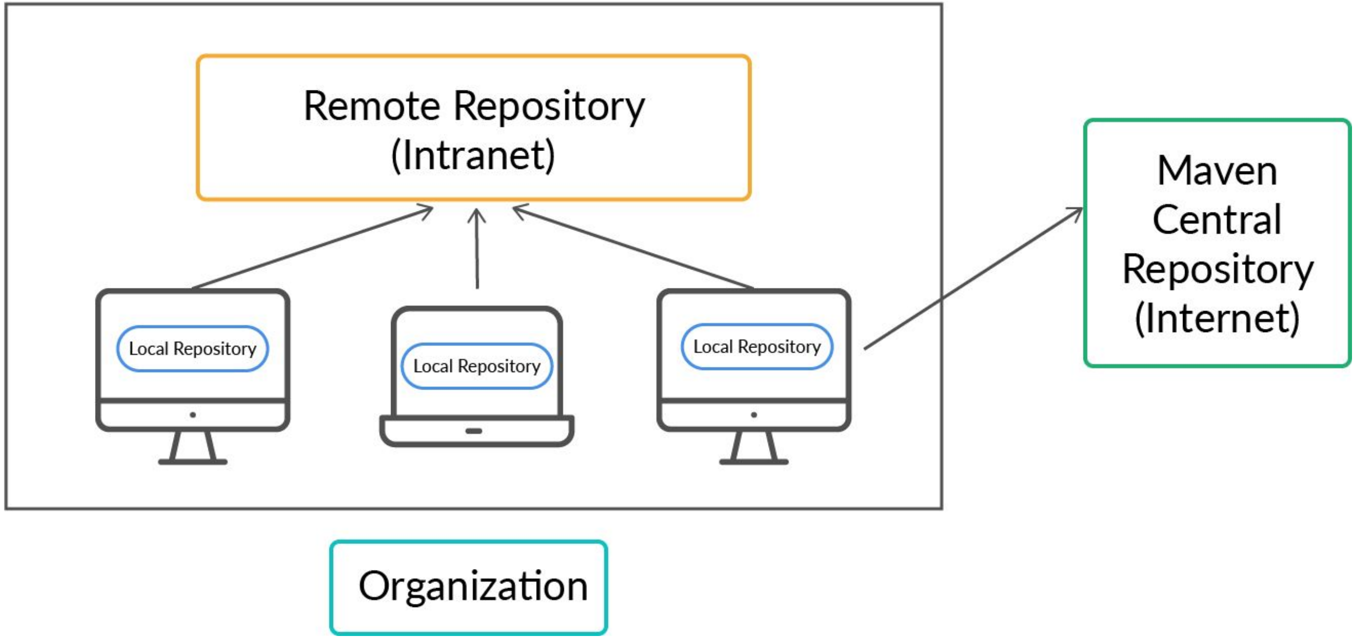
- The external jars are also called API plugins, a software component that adds additional features to existing applications.
- A jar is set of Java classes, interface, enums and annotations zipped into a .jar file.
- Jar files are mostly provided by third-party vendor companies.
- It needs to be added separately into the application classpath.
- Examples of API plugins are as follows:
    - The Spring Framework
    - JDBC Type 4 Driver

Maven Repository

- A location where all Maven dependencies (jar files) are stored.
- It is of the following two types:
  - Local repository (c:\Users\username\.m2)
  - Remote repository (https://mvnrepository.com/)

Maven

MAVEN REPOSITORIES



Remote Repository (Intranet)

Local Repository

Local Repository

Local Repository

Maven Central Repository (Internet)

Organization

Maven

upGrad

pom.xml

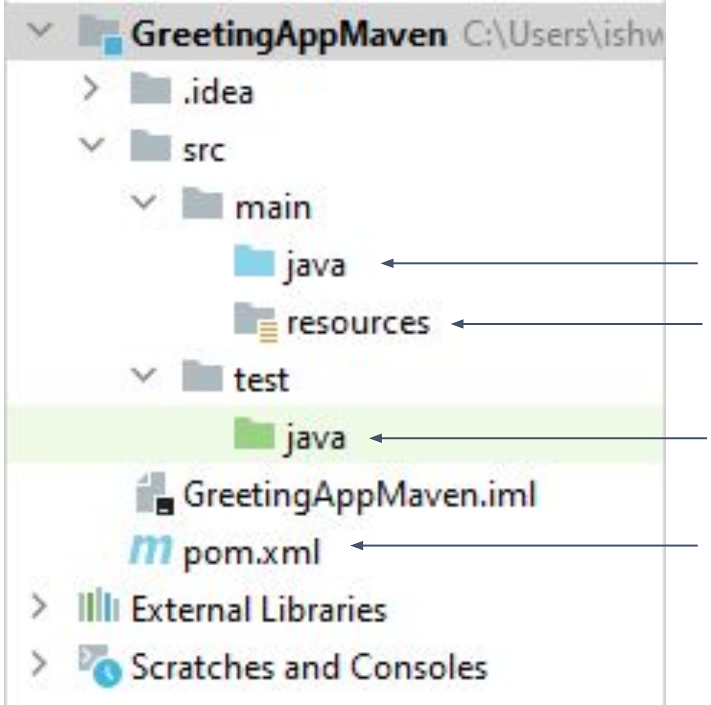
- It contains all the project-related configuration information that will be used by Maven to build the application.
- It also contains a list of all the API plugin jar dependencies that need to be downloaded.
- It resides in the base directory of a project.

Maven

Let’s create a new Maven project inside IntelliJ IDEA by following the steps given below.

- Click on File -> New -> Project
- Select Maven from the left-side pane and click on Next
- Provide the name and artifact coordinates for your Maven project
- It create a new Maven project with the project structure shown on the next page.

Maven



```
GreetingAppMaven C:\Users\ishw
  .idea
  src
    main
      java
      resources
    test
      java
  GreetingAppMaven.iml
  pom.xml
External Libraries
Scratches and Consoles
```

Contains the source code of your application

Contains the resources related to your source code, such as metadata, images or other files

Contains test cases for your application

The pom.xml file that contains configuration information for your project

Maven

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
```

Metadata about the POM file

```
   <groupId>org.upgrad</groupId>
   <artifactId>GreetingAppMaven</artifactId>
   <version>1.0-SNAPSHOT</version>
```

Artifact coordinates

```
</project>
```

Maven

Now, let’s download the jars that are required for the project.

- Go to https://mvnrepository.com/ and search for the following Maven dependencies
  - Spring core
  - Spring context
  - Apache Commons Logging
- Select the appropriate version and copy the dependency into the pom.xml file.
- When you rebuild the project, Maven will download all the jars corresponding to the specified dependencies and add them to the classpath.

Poll 9 (15 seconds)

Which of the following statements about Maven is true? (Note: More than one option may be correct.)

A. Maven is a project building and configuration tool.

B. Maven manages the downloading of the dependencies from the remote repository.

C. Maven is a framework.

D. Maven helps in achieving loose coupling.

Poll 9 (15 seconds)

Which of the following statements about Maven is true? (Note: More than one option may be correct.)

A. Maven is a project building and configuration tool.

B. Maven manages the downloading of the dependencies from the remote repository.

C. Maven is a framework.

D. Maven helps in achieving loose coupling.

Other Features of the Spring Framework

Spring Framework

Here are some of the other features of Spring that we can benefit from:

- It is **open source.**
- It is **lightweight** in terms of execution, as it does not require an external container support to run applications, such as the Servlet Container that is used to run the Servlet/JSP application, and because of its POJO implementation.
- **Modular:** It is developed in different modules. So, you can import only those Spring modules as per the application's requirements and ignore the rest.
- **The integration with other frameworks is in a loosely coupled manner,** such as integration with the Hibernate framework.

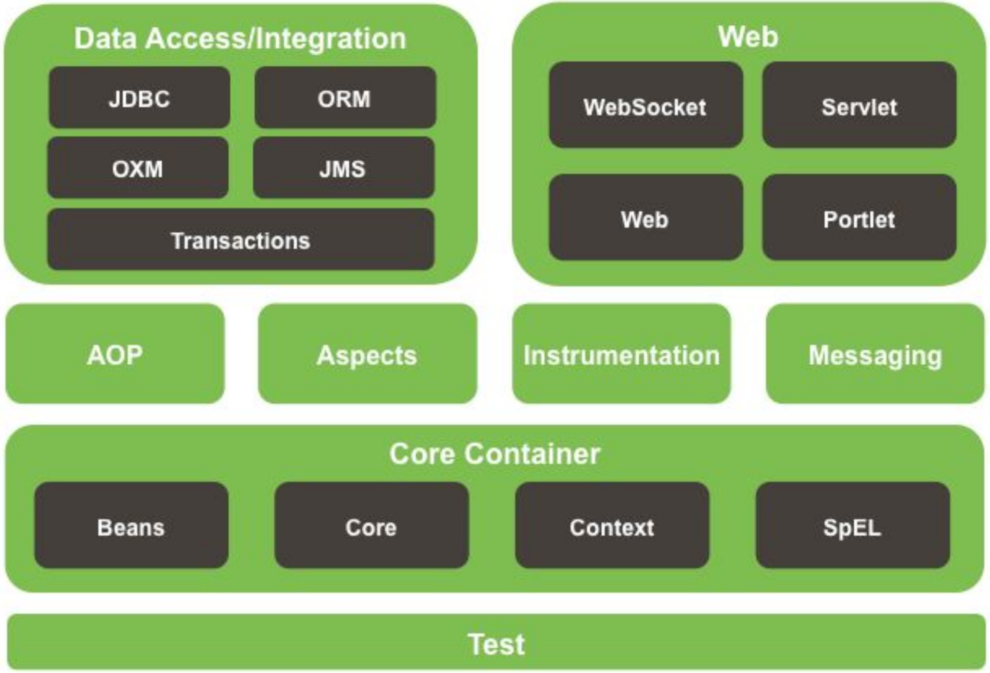
Spring Framework

Here are some of the other features of Spring that we can benefit from:

- **Aspect-oriented programming:** It uses AOP to separate cross-cutting concerns (logging, security, exception handling, etc) from the business logic.
- **Dependency injection:** It helps in developing loosely coupled applications and, hence, easier unit testing.
- **IoC containers:** It creates and manages the lifecycle of the Spring beans.
- **Transaction management:** It is used to maintain transactions and is mostly used to maintain database transactions.

Spring Architecture

The Spring Framework provides approximately 20 modules that can be used according to the application's requirement.



https://docs.spring.io/spring/docs/5.0.0.RC3/spring-framework-reference/overview.html

Spring Sub-Projects

In addition to the modules, Spring also provides some other frameworks that can be used for other purposes. Here is a list of some of them:

- Spring Boot: Helps you build Spring web applications faster
- Spring Batch: Simplifies and optimises the processing of high-level batch operations
- Spring Security: Provides authentication and authorisation support
- Spring Data: Provides a consistent approach to data access
- Spring Cloud: Is useful for building and deploying microservices

Poll 10 (15 seconds)

Which of the following is a feature of the Spring Framework? (Note: More than one option may be correct.)

A. IoC Container

B. Dependency injection

C. Build tool

D. Lightweight

Poll 10 (15 seconds)

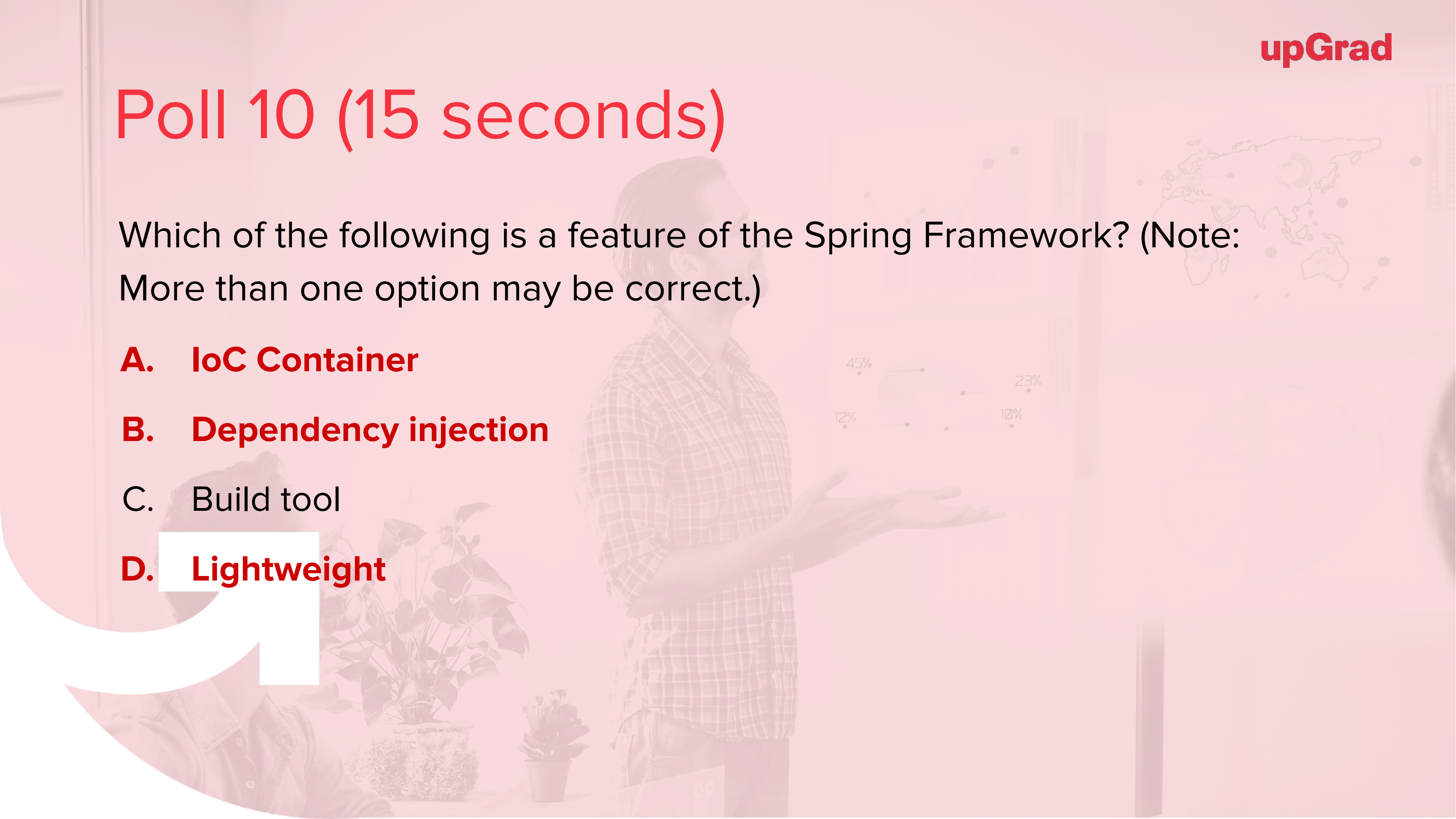
Which of the following is a feature of the Spring Framework? (Note: More than one option may be correct.)

A. IoC Container

B. Dependency injection

C. Build tool

D. Lightweight



All the codes used in today’s session can be found at the link provided below:

https://github.com/ishwar-soni/tm-spring-greeting-app

Important Concepts and Questions

Important Questions

upGrad

1. What is the difference between ApplicationContext and BeanFactory?
2. What problems does Spring solve?
3. What if the outer bean has a singleton scope and the inner bean has a prototype scope?
4. Explain all the scopes of Spring.
   a. Singleton vs prototype
   b. Prototype vs request vs session
5. How does Spring integrate the dependencies internally?
6. In how many ways can dependencies be injected? Which one of them is better?
7. What are the different ways to autowire bean in Spring?
8. What is the internal working of bean creation in Spring? Does it use a new keyword to create the bean? Can you write your own logic to create a bean that is similar to Spring's logic?

Doubt Clearance Window

Today, you learnt about the following:

1. Applications must be developed in a loosely coupled manner to reduce complexity between modules.
2. The Spring Framework works on the software engineering principles of Inversion of Control (IoC) and dependency injection (DI).
3. Spring creates beans of those classes that are marked with the @Component annotations.
4. You can perform DI using the @Autowired and @Qualifier annotations.
5. Spring supports five types of bean scopes. Among them, ‘singleton’ is the default scope.
6. Maven takes care of all the dependencies.
7. You also learnt about other features, modules and sub projects of the Spring Framework.

Homework - Calculator

As part of the homework, you need to build a loosely coupled calculator with the help of Spring. You can follow the given TODOs to achieve it.

- Create a new Maven project and name it Calculator.
- Provide Maven dependencies for Spring in the pom.xml file. You need to provide the following three Maven dependencies:
  - spring-core
  - spring-context
  - commons-logging
- Create a new package inside src/main/java and name that package ‘com.upgrad.calculator’.
- Create a new package inside ‘com.upgrad.calculator’ and name it ‘com.upgrad.calculator.services’.

Homework - Calculator

- Create an interface inside the ‘com.upgrad.calculator.service’ packages and name it ‘MathService’.
  - Provide a method inside this interface with the following signature

    ```
    public int operate(int x, int y);
    ```
- Create two classes inside the ‘com.upgrad.calculator.service’ package and name them ‘AdditionService’ and ‘SubtractionService’.
  - These two classes should implement the ‘MathService’ interface.
  - They should be annotated with the @Component annotation.
  - You should implement the operate() method based on the class name.

Homework - Calculator

- Create a ‘Calculator’ class inside the ‘com.upgrad.calculator’ package.
  - This class should be marked with the @Component annotation.
  - This class should contain two properties of the type ‘MathService’, one for ‘AdditionService’ and another for ‘SubtractionService’.
  - These two dependencies should be injected using the @Autowired and @Qualifier annotation.
  - This method should also contain a method with the following signature:

    ```
    public int compute(String op, int x, int y)
    ```

    where op represents the type of operation that needs to be performed on x and y
  - If op is ‘add’, then use the AdditionService dependency to operate on the x and y arguments. If it is ‘sub’, then use SubtractionService.

Homework - Calculator

- Create a ‘Main’ class inside the ‘com.upgrad.calculator’ package.
- This package should contain the main() method.
  - Inside the main() method, use ApplicationContext and AnnotationConfigApplicationContext to get the bean of ‘Calculator’.
  - Use ‘Scanner’ to get the user input for the type of operation that needs to be performed and get the arguments.
  - Call the compute() method of the calculator bean and print the result on the console.

Code Reference

Note: While running your project, if an error ‘java: error: release version 5 not supported’ is thrown, then add the following tag in your pom.xml file.

```xml
<properties>
   <maven.compiler.source>1.8</maven.compiler.source>
   <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

Tasks to Complete After Today’s Session

MCQs

Coding Questions

Homework - Calculator App

upGrad

#RahoAmbitious

Thank You!