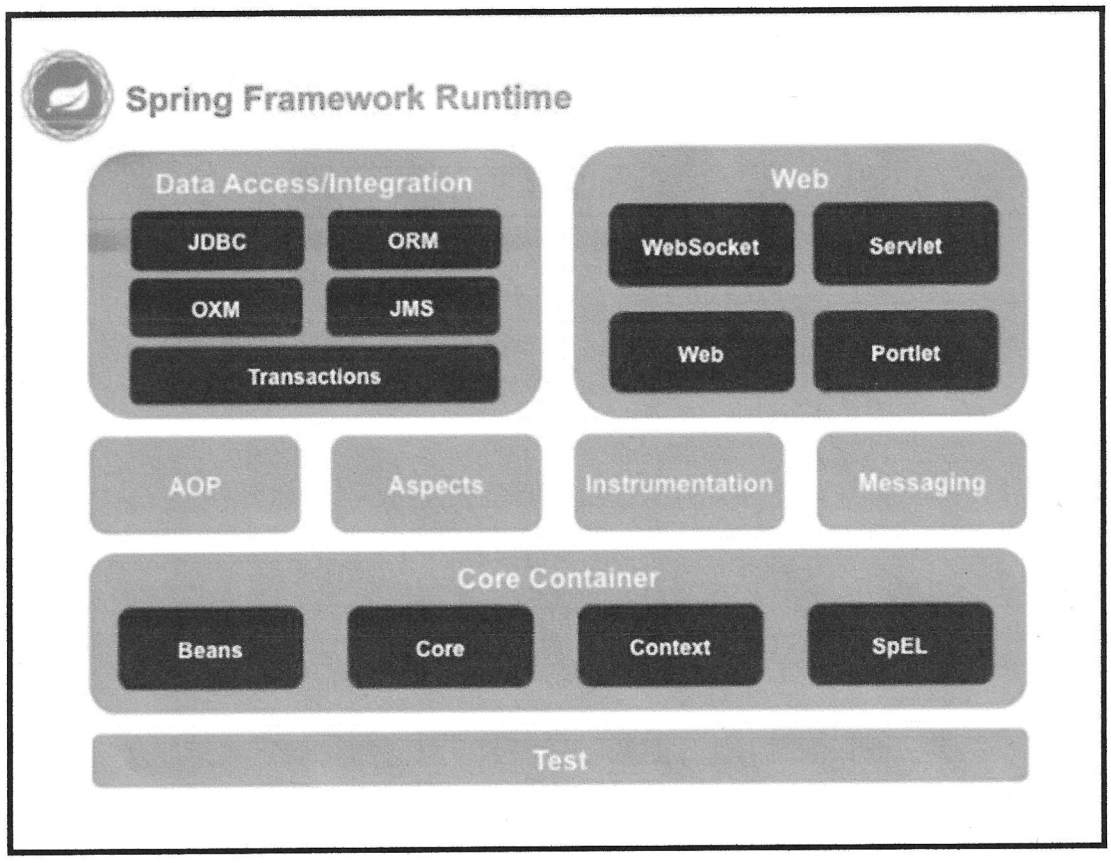
*** This study guide covers the following materials: REST, Web Services, and Spring

Spring Core

1) What are **Spring Projects** and **Spring Modules**?
   a) The **Spring Project Ecosystem** is a collection of projects that complement and extend the **Spring Framework** (aka a Java platform that provides infrastructure support for developing Java applications so that the developer can focus more on the business logic part of the application).
   b) The Spring Framework is a lightweight solution for building enterprise-level applications that supports your business logic.
   c) The Spring Framework:
      (1) Is modular = allowing you to use only the modules from the framework that you need for your application, without depending on the whole framework (aka can plug in & plug out what you need for your app)
      (1) Its 20+ modules:
         (a) All Spring Modules share the same release version as the Spring Framework.
         (b) They are part of the same project (thus the idea of Spring being an ecosystem)
   d) The Main Idea of the framework:
      (1) Includes an **Inversion of Control (IoC)** container that manages your Java objects (aka the zookeeper of our application "ecosystem" that conducts these Spring modules + the rest of our application to function harmoniously together)



2) What is IOC and what does the IOC Container do?
   a) Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework.
   b) Main goal of Spring's IoC Container:
      i) Separation of Concerns!
         (1) Aka decoupling the execution of a task from its implementation
         (2) When achieved, the application code becomes more modular, loosely coupled (to prevent spaghetti code), with less focus on how code is implemented (and abstracts implementation away from the infrastructure--focusing more on business logic)
   c) The IoC container is responsible for instantiating, configuring, and assembling the beans of the application
   d) This container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata (stored in a XML/YML/properties file)
   e) The advantages of IoC:
      i) decoupling the program from the infrastructure

   ii. making it easier to switch between different implementations
   iii. greater modularity of a program
   iv. greater ease in testing a program by isolating a component or mocking its dependencies and allowing components to communicate through contracts
f. Mention a little about Martin Fowler's IoC paper!
   i. Link: https://martinfowler.com/bliki/InversionOfControl.html
   ii. He mentions that this phenomenon as the **Hollywood Principle - "Don't call us, we'll call you"**
   iii. Inversion of Control is a key part of what makes a framework different to a library.
      1. A library is essentially a set of functions that you can call, these days usually organized into classes. Each call does some work and returns control to the client.
      2. A framework embodies some **abstract** design, with more behavior built in.
      3. **MAIN POINT**: In order to use a framework in your application, you need to insert your behavior into various places in the framework either by subclassing or by plugging in your own classes. The framework's code then calls your code at these points.

3. What is dependency injection and what are some of the benefits of using dependency injection?
   a. According to Martin Fowler's article, dependency is a specific style of inversion of control.
      i. DI = more specific of IoC
      ii. IoC = a generic pattern of control flow between app implementation and app infrastructure
   b. Dependency Injection → implementations are passed into an object through constructors/setters/service lookups, which the object will 'depend' on in order to behave correctly
   c. Benefits of DI:
      i. Reduced Dependencies & Reduced Dependency Carrying
         1. Reduced dependencies = by reducing unnecessary dependencies, the object becomes less vulnerable to changes that might occur from a dependency changing.
         2. Reduced dependency carrying = is when an object takes a parameter in one of its methods that it doesn't need itself, but is needed by one of the objects it calls to carry out its work. This can make for harder to maintain or test code.

   ii. More Reusable Code = by external configurations, it separates the code from the infrastructure within any given context, making for easier application maintenance
   iii. More Testable Code = DI makes it possible to inject mock implementations of these dependencies.
   iv. More Readable Code = As DI moves the dependencies to the interface of components, this makes it easier to see what dependencies a component has, making the code more readable (and abstracts the implementation to a interface)
4. What types of dependency injection does Spring support?
   a. Constructor Injection:
      i. Dependency Injection accomplished when the container invokes a **constructor with arguments** to instantiate a bean in which each argument of said constructor represents a dependency.
   b. Setter Injection:
      i. Dependency Injection accomplished when the container **calls setter methods** on a bean after invoking a no-argument constructor to instantiate a bean.
   c. Constructor vs Setter Injection:
      i. Constructor Injection is more secure, since dependencies are required to create an object, you are guaranteed to have each dependency populated
      ii. Consturctor Injection enables the implementation of immutable objects
      iii. Setter Injection allows for partial dependencies since Constructor injection requires all properties to be established upon bean instantiation.
      iv. Setter Injection occurs after constructor injection, essentially putting giving setter injection precedence over constructor injection
      v. Setter Injection can easily change values, and does not create new bean instances, making it more flexible than constructor injection.
      vi. Setter Injection can resolve circular references (i.e. if Object A and Object B are dependent on each other, setter injection can be used to resolve this, whereas Constructor injection would throw a BeanCurrentlyInCreationException).
   d. Fun fact: there is field injection
      i. This type of injection is possible only in the annotation based approach due to the fact that it is not really a new injection type — under the hood, Spring uses reflection to set these values.
      ii. Advantages

1. Easy to use, no constructors or setters required
2. Can be easily combined with the constructor and/or setter approach

iii. Disadvantages

1. Less control over object instantiation. In order to instantiate the object of a class for a test, you will need either a Spring container configured or mock library — depending on the test you are writing.
2. A number of dependencies can reach dozens until you notice that something went wrong in your design.
3. No immutability — the same as for setter injection.

e. Fun fact: there is also lookup method injection

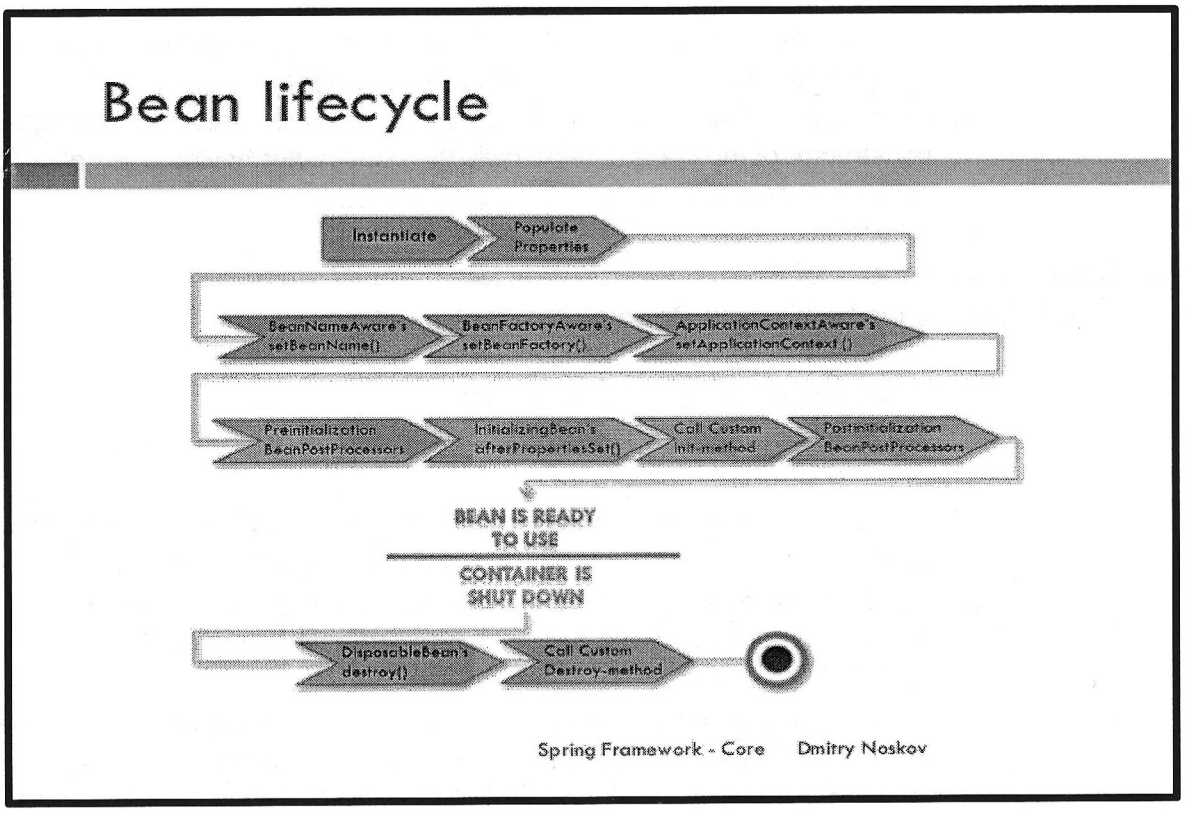
i. By lookup-method XML tag within a configuration file or by annotation of the bean class, we can indicate (or denote) the name of a service/method, which will inject the dependency.

1. totally different from others, used for injection dependency of smaller scope. Possible to configure in: XML, XML+Annotations, Java, Java + Annotations.

f. Source: https://medium.com/@ilyailin7777/all-dependency-injection-types-spring-336da7baf51b

5. What are some differences between BeanFactory and ApplicationContext? Which one eagerly instantiates your beans?
   a. The BeanFactory is the most basic version of IOC containers, and the ApplicationContext extends the features of BeanFactory.
   b. BeanFactory uses lazy instantiation (on-demand loading), but ApplicationContext uses eager instantiation (at-startup loading).
   c. ApplicationContext enhances BeanFactory in a more framework-oriented style. Provides:
      i. messaging functionality
      ii. event publication functionality
      iii. annotation-based dependency injection,
      iv. easy integration with Spring AOP (aspect-oriented programming)
   d. ApplicationContext supports all types of bean scopes, while BeanFactory only supports two scopes (Singleton and Prototype)
   e. The ApplicationContext automatically registers BeanFactoryPostProcessor and BeanPostProcessor at startup without any additional code. On the other hand, the BeanFactory does not register these interfaces automatically.
   f. NOTE: it's generally recommended to use the ApplicationContext but consider using BeanFactory only when memory consumption is critical.

6. What is the Spring Bean lifecycle?
    a. What is a Spring Bean?
        i. In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.
        ii. A Spring Bean therefore is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
    b. Spring provides the BeanPostProcessor interface that gives you the means to tap into the Spring context life cycle and interact with beans as they are processed.
    c. Spring Lifecycle:
        i. Instantiate
        ii. Populate Properties
        iii. Call setBeanName of BeanNameAware
        iv. Call setBeanFactory of BeanFactoryAware
        v. Call setApplicationContext of ApplicationContextAware
        vi. Pre-initialization (BeanPostProcessors)
        vii. afterPropertiesSet of Initializing Beans
        viii. Custom Init Method (SetUp)
        ix. Post Initialization (BeanPostProcessors)
        x. Bean Ready to Use
        xi. Container Shuts Down/Bean is being tore down manually
        xii. Call destroy of DisposableBean
        xiii. Custom Destroy Method (Teardown)
        xiv. End of Bean LifeCycle



7. What is bean wiring? What about autowiring?
    a. Bean wiring is the process of combining beans with Spring container thorough proper configurations (aka annotations or external property settings).
    b. The required beans are to be informed to the container and how the container should use dependency injection to tie them together, at the time of wiring the beans.
    c. There are two ways to wire a bean:
        i. Explicit wiring = The programmer needs to instruct the IoC container through tagging in a XML configuration (XML element tags of property, construct-args) how the container will do the dependency injection
        ii. Autowiring = Without any programmer-defined instructions, Spring automatically detects and injects dependencies by sniffing out any @Autowired annotations in a given application
8. What are the different ways that Spring can wire beans?
    a. The XML-configuration-based autowiring functionality has five modes:
        i. no (default in XML-config-based) = no autowiring at all; bean references are defined via ref element
        ii. byName = autowiring by property name; looks for bean named exactly the same as the property which needs to be autowired (setter dependency injection – aka having a @Autowired over the setter methods)
        iii. byType (*default in Java-config-based*) = autowiring by property class type; can only be wired in there is only one bean with that same type. If

there are multiple, an error will be thrown (setter dependency injection – aka having a @Autowired over the setter methods)

iv. constructor = The constructor mode injects the dependency by calling the constructor of the class. Spring calls the constructor having a large number of parameters.

v. autodetect (deprecated)

9. What are the scopes of Spring beans? Which is default?
    a. Determines how long a Spring Bean stays in scope
        i. singleton *(default)* = This scopes the bean definition to a single instance per Spring IoC container
        ii. prototype = This scopes a single bean definition to have any number of object instances
        iii. request = This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
        iv. session = This scopes a bean definition to an HTTP session (server-side). Only valid in the context of a web-aware Spring ApplicationContext
        v. global-session *(only available in Portlet applications)* = This scopes a bean definition to a global HTTP session (aka scoped to the server container). Only valid in the context of a web-aware Spring ApplicationContext.
10. What is the concept of component scanning and how would you set it up?
    a. Asking Spring to detect Spring-managed components using component scanning tells Spring what information to locate and register those Spring components with the application context when the application starts.
    b. Spring can auto scan, detect, and instantiate components from pre-defined project packages.
        i. Spring can auto scan all classes annotated with the stereotype annotations @Component, @Controller, @Service, and @Repository
        ii. The @ComponentScan annotation is used with the @Configuration annotation to tell Spring the packages to scan for annotated components.
            1. Through the use of sterotype annotations!
            2. @ComponentScan also used to specify base packages and base package classes using thebasePackageClasses or basePackages attributes of @ComponentScan.
11. What are the benefits and limitations of Java configuration?
    a. What is it → The central artifact in Spring's new Java-configuration support is the @Configuration-annotated class, consisting principally of @Bean-annotated methods that define instantiation, configuration, and initialization logic for objects that are managed by the Spring IoC container.
    b. Advantages:
        i. Java is type safe. Compiler will report issues if you are configuring right bean class qualifiers.

ii. XML based on configuration can quickly grow big, while the Java-based configuration allows for cleaner code with any annotation change.
iii. Search is much simpler, refactoring will be bliss. Finding a bean definition will be far easier.

c. Disadvantages:
i. Coupling between the POJO and its behavior
ii. Not centralized, spread throughout the application
iii. Integration of different configurations with the same application is harder to do when using annotations vs a XML file

12. What does the @Configuration and @Bean annotations do?
a. **@Configuration** annotation indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.
b. **@Bean** is a method-level annotation and a direct analog of the XML <bean/> element.
i. When JavaConfig encounters such a method, it will execute that method and register the return value as a bean within a BeanFactory.
ii. By default, the bean name will be the same as the method name (see bean naming for details on how to customize this behavior).

13. What is @Value used for?
a. @Value is a Java annotation that is used at the field or method/constructor parameter level and it indicates a default value for the affected argument.
b. It is commonly used for injecting values (explicitly or by interpolation) into configuration variables

14. What is Spring Expression Language? What can you reference using SpEL? What is the difference between $ and # in @value expressions?

a. For more information: https://stackabuse.com/the-value-annotation-in-spring/#:~:text=%40Value%20is%20a%20Java%20annotation,next%20part%20of%20the%20article.

b. The Spring Expression Language (SpEL) is an expression language which serves as the foundation for expression evaluation within the Spring portfolio.

c. Basically, when using SpEL together with the @Value annotation, we are just changing the way we tell Spring what we need at runtime in the form #{ *<expression string>* }.

d. Examples:

```
@Value("#{systemProperties['user.name']}")
private String userName;
```

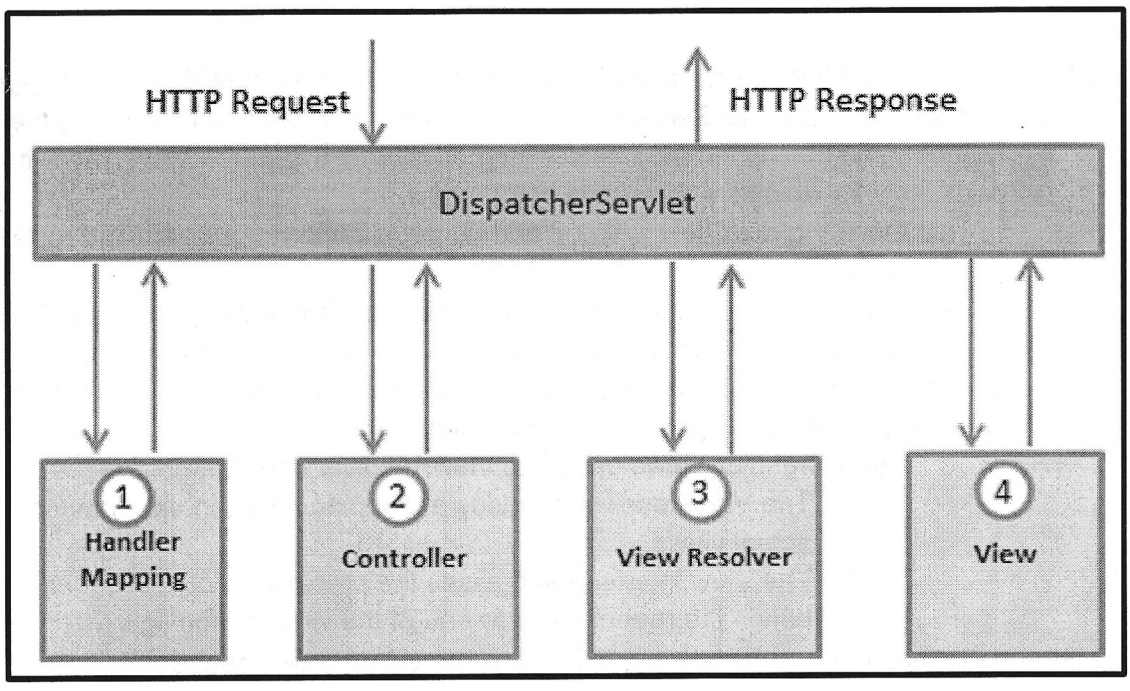
This is how you'd inject a specific system property. On the other hand, we could inject all properties by:

```
@Value("#{systemProperties}")
private Map<String, String> properties;
```

We know now, that we can use the `@Value` annotation for methods as a global value or as a parameter value.

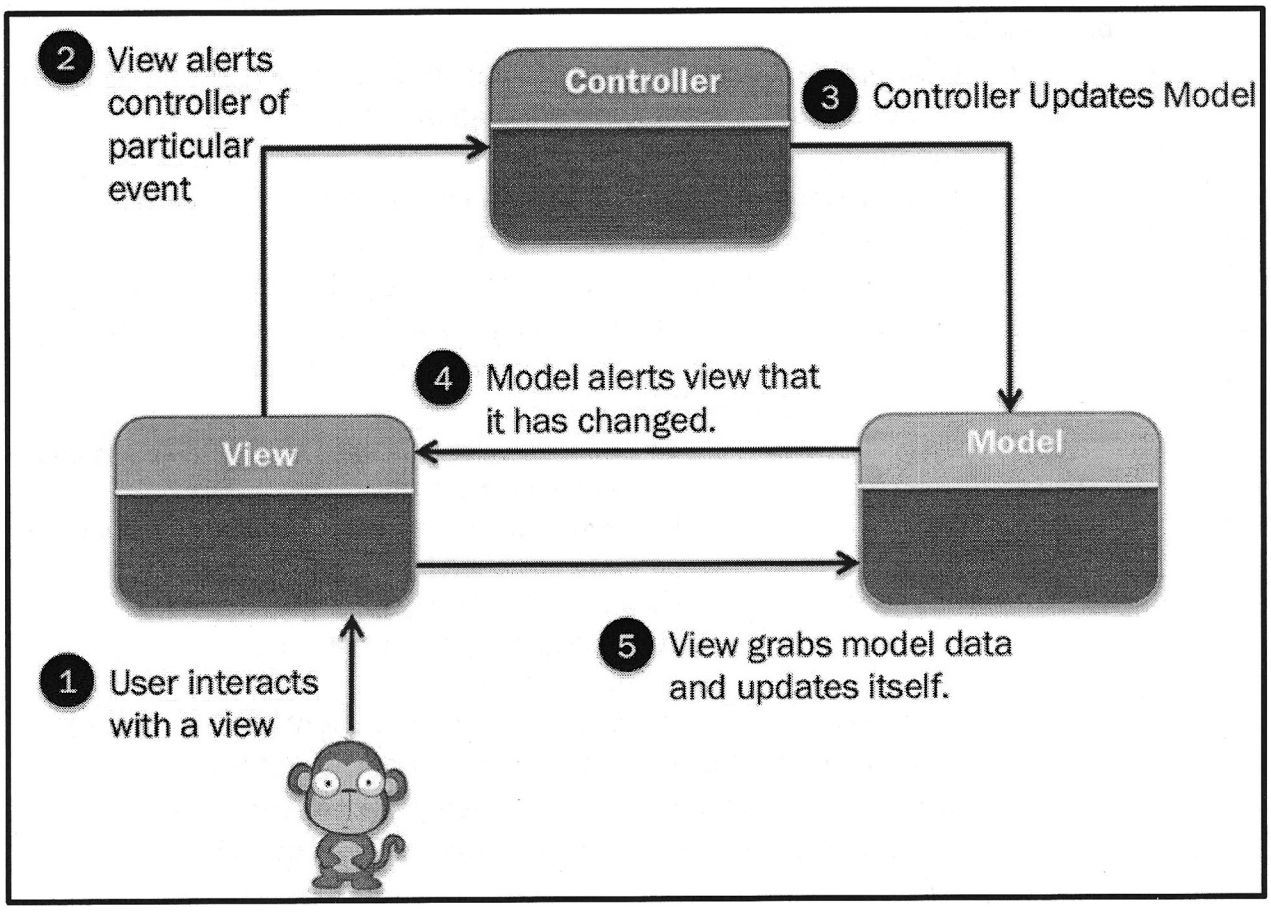
Spring MVC

15. Explain the MVC architecture and how HTTP requests are processed in the architecture
    a. The Model-View-Controller (MVC) framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller.
    b. MVC separates the business logic and presentation layer from each other.
    c. Three important MVC components:
        i. Model: It includes all the data and its related logic
        ii. View: Present data to the user or handles user interaction
        iii. Controller: An interface between Model and View components
    d. How HTTP requests are processed in Spring MVC architecture:
        i. Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet −
            1. After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller.
            2. The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
            3. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
            4. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.



e. More information: https://dzone.com/articles/how-spring-mvc-really-works

f. Generic flow of MVC:



16. What is the role of the DispatcherServlet? What about the ViewResolver?
    a. The job of **DispatcherServlet** is to find the right Controller for processing the request, and then when the handler method returns the logical view name, it also consults view resolvers to see the actual View.
        i. Once the real View is found and output is rendered, it sends the response back to the client.
        ii. The DispatchServlet is an implementation of the front controller pattern (aka the emergency dispatcher or tour guide for our application)
    b. Spring provides view resolvers, which enable you to render models in a browser without tying you to a specific view technology.
        i. Spring handles views using the ViewResolver and View interfaces
            1. The ViewResolver provides a mapping between view names and actual views.
            2. The View interface addresses the preparation of the request and hands the request over to one of the view technologies.
17. List some stereotype annotations. What are the differences between these?
    a. Stereotype Annotations: Annotations denoting the roles of types or methods in the overall architecture (at a conceptual, rather than implementation, level).
    b. Annotations:
        i. @Component = lets the ApplicationContext know to build this class as a bean (basic stereotype annotation) – thus a class level annotation
        ii. @Controller = a class level annotation which tells the Spring Framework that this class serves as a controller in Spring MVC
            a. allows the class to use the @RequestMapping annotation (for mapping web requests onto methods in request-handling classes with flexible method signatures)
                i. This annotation can be used both at the class and at the method level.
                ii. In most cases, at the method level applications will prefer to use one of the HTTP method specific variants @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, or @PatchMapping.
        iii. @Indexed = Indicate that the annotated element or field represents a stereotype for the index (useful in MongoDB and Spring Data)
        iv. @Repository = stereotype for the persistence layer in your application(or any other class that encapulates the CRUD

operations that the applications needs to get data from the database)

v. @Service = stereotype for the service layer (or any other stateless singleton class that implements some logic on other objects)
    1. Not functionally different from the Component annotation

18. How would you declare which HTTP requests you'd like a controller to process?
    a. By declaring which handler that controller will have with the following annotations to map web requests to Spring Controller methods:
        i. The basic @RequestMapping annotation can hold metadata like:
            1. value = the URI endpoint
            2. method = the HTTP Request Method verb
                a. Note: this parameter has no default value! vs with a HTTP browser form's default method is a GET request when not specified
            3. headers = any header data from the request
        ii. @GetMapping = marks a method that is getting a resource
        iii. @PostMapping = marks a method that is creating a new resource
        iv. @PutMapping = marks a method that is updating or replacing every existing resource within the request collection
        v. @DeleteMapping = marks a method that is deleting a resource
        vi. @PatchMapping = marks a method that is modifying the resource
19. What is the difference between @RequestMapping and @GetMapping?
    a. @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET). Both
    b. GetMapping strictly handles HTTP GET requests whereas RequestMapping can deal with any type of HTTP request
20. How to declare the data format your controller expects from requests or will create in responses?
    a. Spring knows about the Accept and Content-Type headers, but it does not know how to convert between Java Objects and JSON. Or XML. Or YAML.
    b. It needs an appropriate 3rd-party library (like JAX-B and Jackson) to do the dirty work (also called marshalling/unmarshalling or serialization/deserialization), which is why the classes that integrate between Spring MVC and these 3rd-party libaries are called HttpMessageConverters
    c. Can also use Spring's built-in annotations. Example:
        i. @RequestBody
        ii. @RequestParam
        iii. @RequestHeader
        iv. The above is usually used in combination of the @RestController annotation (which signals Spring that you do not want to write HTML pages through the usual ModelAndView process and allows data to written directly into the HTTP response body)
21. What annotation would you use to bypass the ViewResolver?

a. @ResponseBody annotation!
b. @ResponseBody is a Spring annotation which binds a method return value to the web response body.
c. **It is not interpreted as a view name.** It uses HTTP Message converters to convert the return value to HTTP response body, based on the content-type in the request HTTP header.

22. How would you extract query and path parameters from a request URL in your controller?
    a. @RequestParam and @PathVariable can both be used to extract values from the request URI. The difference:
        i. @RequestParams extract values from the query string
        ii. @PathVariables extract values from the URI path given within the RequestMapping arguments
            1. "/foos/{id}" ← the request URI path
            2. "/foos?id=1" ← the query param passed within the request
23. What concerns is the controller layer supposed to handle vs the service layer?
    a. In practice:
        i. The Controller's job is to translate incoming requests into outgoing responses. In order to do this, the controller must take request data and pass it into the Service layer.
        ii. The service layer then returns data that the Controller injects into a View for rendering.
    b. By having these layers, it provides:
        i. Separation of concerns between business logic and request handling
            1. Service layer provides code modularity, the business logic and rules are specified in the service layer which in turn calls DAO layer.
            2. Therefore, DAO layer is then only responsible for interacting with DB, and the Controller layer is then only responsible for translating web requests from the client.
        ii. Database security
            1. By having that layer that encapsulates the data access layer, your application is less vulnerable to cyber-attacks because the DB cannot be accessed directly from the client except through your service.
        iii. Loose coupling
            1. Allows for more readability of application code as request handling and business logic/DB calls are separated into their own classes
24. How would you specify HTTP status codes to return from your controller?
    a. Spring provides a few primary ways to return custom status codes from its Controller classes:
        i. using a ResponseEntity
        ii. using the @ResponseStatus annotation on exception classes
        iii. ***using the @ControllerAdvice and @ExceptionHandler annotations.

b. Code examples:

i. via a ResponseEntity

```
@RequestMapping(value = "/controller", method = RequestMethod.GET)
@ResponseBody
public ResponseEntity sendViaResponseEntity() {
    return new ResponseEntity(HttpStatus.NOT_ACCEPTABLE);
}
```

ii. via Exception

```
@RequestMapping(value = "/exception", method = RequestMethod.GET)
@ResponseBody
public ResponseEntity sendViaException() {
    throw new ForbiddenException();
}
```

iii. via @ResponseStatus

```
@ResponseStatus(HttpStatus.FORBIDDEN)
public class ForbiddenException extends RuntimeException {}
```

```
@ResponseStatus(value = HttpStatus.FORBIDDEN, reason="To show an example of a custom message")
public class ForbiddenException extends RuntimeException {}
```

25. How do you handle exceptions thrown in your code from your controller? What happens if you don't set up any exception handling?
    a. You can add methods annotated with @ExceptionHandler to specifically handle exceptions thrown by request handing methods (aka methods marked with @RequestMapping) in the same controller.
    b. Can also have a separate utility class that handles your controller exceptions, annotating it with @ControllerAdvice
        i. A controller advice allows you to use exactly the same exception handling techniques but apply them across the whole application, not just to an individual controller. You can think of them as an annotation driven interceptor.
26. How would you consume an external web service using Spring?
    a. Spring supports calling one rest service to another rest service using the **RestTemplate** class.
    b. RestTemplate is a synchronized, client side class that is responsible for calling another rest service. RestTemplate supports all HTTP methods such as GET, POST, DELET, PUT, HEAD, etc.
27. What are the advantages of using RestTemplate?

a. Is thread-safe for each request - once constructed, you can use callbacks to customize its operations for client-side HTTP access

    i. Until each request is completed and response is sent back to user or service erred out, the thread will be blocked out for that user.

    ii. This blocking technique makes it difficult to make async requests with RestTemplate (which is why Spring plans to depreciate it in the future and go with Spring WebClient)

b. Abstraction

    i. If we want to make an HTTP Call, we need to create an HttpClient, pass request and form parameters, setup accept headers and perform unmarshalling of response, all by yourself.

    ii. Spring Rest Templates tries to take the pain away by abstracting all these details from you.

Spring AOP

28. What is "aspect-oriented programming"? Define an aspect.
    a. AOP addresses the problem of **cross-cutting concerns**, which would be any kind of code that is repeated in different methods and can't normally be completely refactored into its own module.
    b. The aspect is the key unit of modularity.
        i. An aspect is the part of the application which cross-cuts the core concerns of multiple objects. Example: transaction management, security
29. Give an example of a cross-cutting concern you could use AOP to address
    a. Aspects enable the implementation of crosscutting concerns (such as transaction, logging, security, caching) that are not central to business logic without cluttering the code core to its functionality.
    b. It does so by adding additional behavior that is the advice to the existing code.
    c. Annotate our class with @Aspect and @Component
        i. @Aspect is an annotation we get from Aspect J which is a full AOP framework; we just use the AspectJ annotations rather than xml config
30. Define the following:

- Join point = a point during the execution of a program, which in Spring AOP's case is the execution of a method
    - where you **can** inject
- Pointcut = a predicate (or search param) that matches join points
    - Where you **will** inject
- Advice = action taken by an aspect at a particular join point

- the code to be injected at a join point
- **what** you will inject

31. What are the different types of advice? What is special about the @Around advice? How is the ProceedingJoinPoint used?
    a. 3 types:
        i. Before advice (@Before):
            1. Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
        ii. After advice (@After):
            1. After returning advice (@AfterReturning): Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
            2. After throwing advice (@AfterThrowing): Advice to be executed if a method exits by throwing an exception.
            3. After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
        iii. Around advice (@Around): Advice that surrounds a join point such as a method invocation.
            1. **This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation.**
            2. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.
    b. **JoinPoint** interface can be passed into any aspect method
        i. Gives access to the target object (the object being advised; this is a Spring AOP proxy object)
        ii. Allows us to get the JP method signature, arguments, etc.
    c. **ProceedingJoinPoint** is a subinterface of JoinPoint that can be passed into a method which handles Around advice only
        i. Allows you to control if/when the join point is actually executed relative to the surrounding advice

32. Explain the pointcut expression syntax
    a. A pointcut expression language is a way of describing pointcuts programmatically.
    b. Example:

A pointcut expression can appear as a value of the *@Pointcut* annotation:

```
@Pointcut("within(@org.springframework.stereotype.Repository *)")
public void repositoryClassMethods() {}
```

The method declaration is called the **pointcut signature**. It provides a name that can be used by advice annotations to refer to that pointcut.

```
@Around("repositoryClassMethods()")
public Object measureMethodExecutionTime(ProceedingJoinPoint pjp) throws Throwable {
    ...
}
```

A pointcut expression could also appear as the value of the *expression* property of an *aop:pointcut* tag:

```
<aop:config>
    <aop:pointcut id="anyDaoMethod"
      expression="@target(org.springframework.stereotype.Repository)"/>
</aop:config>
```

c. Pointcut Designators:
   i. A pointcut expression starts with a **pointcut designator** (PCD), which is a keyword telling Spring AOP what to match.
   ii. execution: most common, method execution
      1. "execution([type] [methodSignature](..))
      2. "execution(* doSomething())"
      3. "execution(* set*(..))
   iii. within: limits to method execution within certain classes
      1. "within(com.revature.beans.*)"
   iv. this: limits matching to jp's where the AOP proxy being advised
   v. target: limits matching to jp's where the target object is being proxied
   vi. args: limits matching to jp's where arguments are instances of the given types

33. What visibility must Spring bean methods have to be proxied using Spring AOP?

a. Only **public** methods of Spring beans will be proxied

b. Additionally the call to the public method must originate from outside of the Spring bean.

****Complete Spring AOP study guide link: https://mossgreen.github.io/Spring-Certification-Spring-AOP/

Spring Data

34. What is Spring ORM and Spring Data?
   a. Spring-ORM is an umbrella module that covers many persistence technologies, namely JPA, JDO, Hibernate and iBatis, each providing smooth integration with Spring transaction management.

i. For each technology, the configuration basically consists in injecting a DataSource bean into some kind of SessionFactory or EntityManagerFactory etc. bean.

b. Spring-Data is an umbrella project that provides a common API to define how to access data (DAO + annotations) in a more generic way, covering both SQL (like JPA) and NOSQL (like Redis and Hadoop) data sources.

35. What is the Template design pattern and what is the JDBC template?
    a. In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods.
        i. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.
        ii. Base class declares algorithm 'placeholders', and derived classes implement the placeholders.
    b. The JDBC template is the main API in Spring JDBC module that will focus on:
        i. creation and closing of connections
        ii. executing statements and stored procedure calls
        iii. iterating over the ResultSet and returning results
36. What does @Transactional do? What is the PlatformTransactionManager?
    a. @Transactional defines the scope of a single database transaction.
    b. PlatformTransactionManager is the Spring transaction client API that provides the classic transaction client operations: begin, commit and rollback. In other words, this interface provides the essential methods for controlling transactions at runtime.
37. What is a PersistenceContext?
    a. The persistence context is just a synchronizer object that tracks the state of a limited set of Java objects and makes sure that changes on those objects are eventually persisted back into the database.
38. Explain how to integrate Spring and Hibernate using ContextualSession
    a. The DAO implementation uses Hibernate's SessionFactory to query against the database, instead of using JdbcTemplate.
    b. Basically, in order to support Hibernate integration, Spring provides two key beans available in the org.springframework.orm.hibernate4 package:
        i. LocalSessionFactoryBean: creates a Hibernate's SessionFactory which is injected into Hibernate-based DAO classes.
        ii. HibernateTransactionManager: provides transaction support code for a SessionFactory. Programmers can use @Transactional annotation in DAO methods to avoid writing boiler-plate transaction code explicitly.
39. What interfaces are available in Spring Data JPA?
    a. @JpaRepository = provides CRUD operations for our entities
    b. @EnableJpaRepositories = Will scan the package of the annotated configuration class for Spring Data repositories (@JpaRepository)
40. What is the difference between JPARepository and CrudRepository?

a. CrudRepository and JPA repository both are the interface of the spring data repository library.
b. JpaRepository extends PagingAndSortingRepository which in turn extends CrudRepository.
    i. CrudRepository mainly provides CRUD functions.
    ii. PagingAndSortingRepository provides methods to do pagination and sorting records.
    iii. JpaRepository provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.
c. Other differences charted below:

Sr. No.	Key	JPARepository	CrudRepository
1	Hierarchy	JPA extend crudRepository and PagingAndSorting repository	Crud Repository is the base interface and it acts as a marker interface.
2	Batch support	JPA also provides some extra methods related to JPA such as delete records in batch and flushing data directly to a database.	It provides only CRUD functions like findOne, saves, etc.
3	Pagination support	JPA repository also extends the PagingAndSorting repository. It provides all the method for which are useful for implementing pagination.	Crud Repository doesn't provide methods for implementing pagination and sorting.
4	Use Case	JpaRepository ties your repositories to the JPA persistence technology so it should be avoided.	We should use CrudRepository or PagingAndSortingRepository depending on whether you need sorting and paging or not.

41. What are the naming conventions for methods in Spring Data repositories?
    a. camelCase!!
42. How are Spring repositories implemented by Spring at runtime?
    a. The fundamental approach is that a JDK proxy instance is created programmatically using Spring's ProxyFactory API to back the interface and a MethodInterceptor intercepts all calls to the instance and routes the method into the appropriate places
    b. This, in turn, helps at runtime, Spring Data JPA will create your repository implementations with the common CRUD methods. You can then perform CRUD operations without writing a single line of data access code.
43. What is @Query used for?

    a. You need to inform Spring Data JPA on what queries you need to execute by using the @Query annotation

    b. This annotation takes a custom query as a string.

c. Read more at: https://springframework.guru/spring-data-jpa-query/

d. Example:

```
01. package guru.springframework.customquery.repository;
02.
03. import guru.springframework.customquery.domain.Book;
04. import org.springframework.data.jpa.repository.Query;
05. import org.springframework.data.repository.CrudRepository;
06. import org.springframework.stereotype.Repository;
07.
08. @Repository
09. public interface BookRepository extends CrudRepository<Book, Integer> {
10.   @Query("SELECT b FROM Book b")
11. List<Book> findAllBooks();
12.
13. }
```

Spring Boot

44. How is Spring Boot different from legacy Spring applications? What does it mean that it is "opinionated"?
    a. Spring Boot is basically an extension of the Spring framework which eliminated the boilerplate configurations required for setting up a Spring application.
    b. It takes an opinionated view of the Spring platform which paved the way for a faster and more efficient development eco-system.
    c. Spring Boot just decides on a set of default configured beans which you can override if you want, providing an **opinionated** configuration and dependency starting point.
    d. Spring Boot also has an embedded server to avoid complexity in application deployment (aka no more setting up runtime environments manually)
    e. Other features and benefits of Spring Boot:
        i. Automatic config for Spring functionality – whenever possible and with ease
        ii. Necessary pre-configuration found in application.properties file
            1. No xml configuration for additional config
        iii. Spring Boot Actuator - provides endpoints for helpful metrics and project information
        iv. Spring Boot Development tools
45. What does "convention over configuration" mean?
    a. Spring has always favored *convention over configuration*, which means it takes up the majority of working uses cases into consideration rather than nit-picking an exact configuration and dependencies required for a specific application development.

46. What annotation would you use for Spring Boot apps? What does it do behind the scenes?
    a. Annotate the class with your main method with **@SpringBootApplication**
    b. @SpringBootApplication encapsulates these annotations with their default attributes:
        i. @Configuration = labels the class as an auto-configured bean
        ii. @EnableAutoConfiguration = enables auto-configuration, meaning that Spring Boot looks for auto-configuration beans on its classpath and automatically applies them.
        iii. @ComponentScan
            1. It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.
    c. In main method:
        i. SpringApplication.run(Driver.class, args) to run the entire Spring Boot app
47. How does Boot's autoconfiguration work?
    a. Autoconfiguration is a feature that allows developers to automatically configure beans in the Spring context based on different conditions of the application, such as the presence of certain classes in the classpath, the existence of a bean or the activation of some property.
    b. The goal was to provide default implementations for all of those infrastructure components (and they should fit perfectly well for most of our cases) but we also wanted to have the option to modify some of them if needed.
48. What is the advantage of having an embedded Tomcat server?
    a. Pros:
        i. You can create the entire applications in a fast track manner. Not very in-depth knowledge of Spring framework is required. It defaults a lot of settings or in other words convention over configuration.
        ii. Good for developing micro services in docker environment. Spring Cloud Netflix OSS is one such example.
        iii. It has embedded tomcat servlet container and you write code as if you are writing a standalone Java code with plain old main method. This way you are focusing more on the developing the business logic rather than plumbing aspects.
    b. Cons:
        i. As its rapid application development with convention over configuration and you really want to control each aspect of the application - then you need to know everything that is happening under the hood. For any further code changes, you just don't have to go thru the source code but each configuration that is done by Spring boot for you.
        ii. Lots of JAR files and settings are created for you so it might impact the overall footprint of the app as well as impact the performance. Yes, you can work on it but you need to know what's going on under the hood.
49. What is the significance of the Spring Boot starter POM?

a. Starter POMs are a set of convenient dependency descriptors that you can include in your application.
b. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy-paste loads of dependency descriptors.
c. Benefits:
    i. increase pom manageability
    ii. production-ready, tested & supported dependency configurations
    iii. decrease the overall configuration time for the project

50. What is the Spring Boot actuator? What information can it give you?
a. Spring Boot Actuator is a sub-project of the Spring Boot Framework.
b. It includes a number of additional features that help us to monitor and manage the Spring Boot application.
c. It contains the actuator endpoints (the place where the resources live).
    i. To access the actuator:
        1. http://<host-name>:<post>/actuator/<actuator-endpoint>.
d. Key information given by actuators:
    i. Endpoints = allows us to monitor and interact with the application.
        1. Key endpoints:

ENDPOINTS	USAGE
/metrics	To view the application metrics such as memory used, memory free, threads, classes, system uptime etc.
/env	To view the list of Environment variables used in the application.
/beans	To view the Spring beans and its types, scopes and dependency.
/health	To view the application health
/info	To view the information about the Spring Boot application.
/trace	To view the list of Traces of your Rest endpoints.

    ii. Metrics
        1. Spring Boot Actuator provides dimensional metrics by integrating with the micrometer.
        2. The micrometer is integrated into Spring Boot. It is the instrumentation library powering the delivery of application metrics from Spring. It provides vendor-neutral interfaces for timers, gauges, counters, distribution summaries, and long task timers with a dimensional data model.
    iii. Audit
        1. Spring Boot provides a flexible audit framework that publishes events to an **AuditEventRepository**, which automatically

publishes the authentication events if spring-security is in execution.

51. What files would you use to configure Spring Boot applications?
    a. Spring Boot lets you externalize your configuration so that you can work with the same application code in different environments.
    b. **You can use properties files, YAML files, environment variables, and command-line arguments** to externalize configuration.
52. What is the benefit of using Spring Boot profiles?
    a. Profile is nothing but a key to identify an environment for your application.

Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. Any `@Component` or `@Configuration` can be marked with `@Profile` to limit when it is loaded:

```
@Configuration
@Profile("production")
public class ProductionConfiguration {

    // ...

}
```

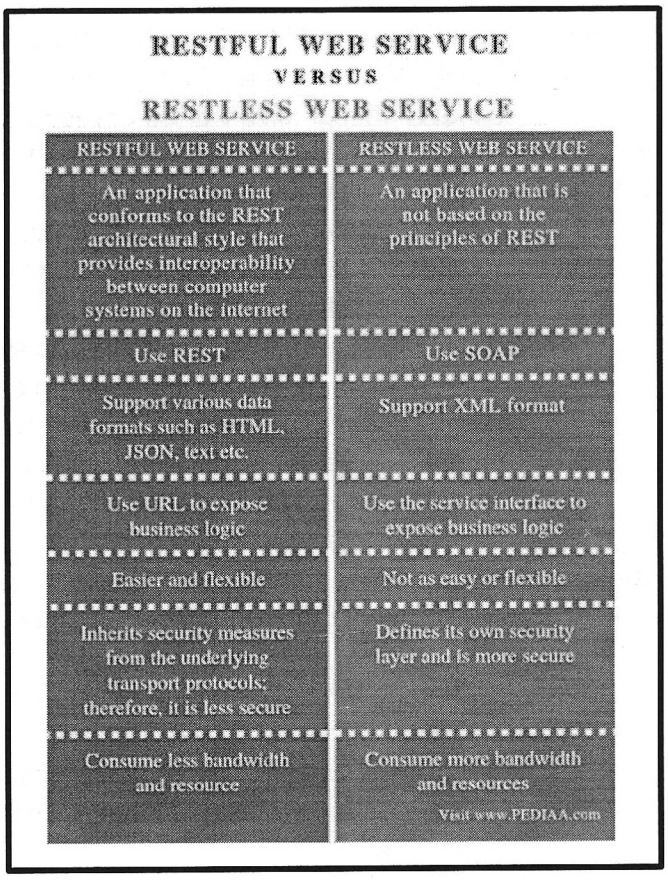
In the normal Spring way, you can use a `spring.profiles.active` `Environment` property to specify which profiles are active. You can specify the property in any of the usual ways, for example you could include it in your `application.properties`:

```
spring.profiles.active=dev,hsqldb
```

or specify on the command line using the switch `--spring.profiles.active=dev,hsqldb`.

REST & Web Services

53. What is a web service?
    a. A collection of open protocols and standards used for exchanging data between applications or systems.
    b. A web service is any service that:
        i. Is available over the Internet
        ii. Uses a standardized XML messaging system
        iii. Is not tied to any one operating system or programming language
        iv. Is self-describing via a common XML grammar
        v. Is discoverable via a simple find mechanism
54. How does a RESTful web service differ from a normal web service?
    a. A RESTful Web Service is a lightweight, maintainable, and scalable service that is built on the REST architecture
    b. Differences:
        i. A RESTful Web Service will expose an API from your application in a secure, uniform, stateless manner to the calling client.
        ii. The calling client can perform predefined operations using the RESTful service.
        iii. The underlying protocol for REST is HTTP.
        iv. Chart for comparsion between RESTful and RESTless web services:



55. What is REST? What are its constraints?
    a. REST stands for REpresentational State Transfer.
    b. REST is a way to access resources which lie in a particular environment.
    c. Roy Fielding describes how a well-designed web application is supposed to behave like with these 6 constraints:
        i. **Uniform interface**
            1. Having a uniform way of interacting with a given server regardless of device or type of application
            2. Guidelines:
                a. Resource-Based: Individual resources are identified in requests. For example: API/users.
                b. Manipulation of Resources Through Representations: Client has representation of resource and it contains enough information to modify or delete the resource on the server, provided it has permission to do so. Example: Usually user get a user id when user request for a list of users and then use that id to delete or modify that particular user.
                c. Self-descriptive Messages: Each message includes enough information to describe how to process the message so that server can easily analyses the request.

d. Hypermedia as the Engine of Application State (HATEOAS): It need to include links for each response so that client can discover other resources easily.

ii. Client-server
    1. Should have a client-server architecture
    2. Neither client nor server are concerned with what their counterpart is doing and thus each evolve independently
        a. REMEMBER:
            i. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server
            ii. A Server is someone who holds the resources and are not concerned with the user interface or user state.

iii. Stateless
    1. The state to handle the request is contained within the request itself
        a. Meaning the server would not store anything related to the session.
        b. Think of it as "Browser State Amnesia" as the browser will forget the current session state after the request is sent to the server.
        c. Advantage: enables greater availability since the server does not have to maintain, update or communicate that session state with every request
        d. Drawback: Reduction of the scope of network optimization the larger the client's request for data becomes

iv. Cacheable
    1. Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side.
    2. Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server.
    3. A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance. But sometime there are chances that user may receive stale data.

v. Layered system
    1. An application architecture needs to be composed of multiple layers.
    2. Each layer doesn't know anything about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server.

3. Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

vi. Code on demand (optional)

1. Servers can provide executable code (in the form of snippets or scripts) to the client.

56. What is SOAP? Name and describe its message elements.

a. SOAP = Simple Object Access Protocol
   i. A standards-based web services access protocol that been around longer than REST
   ii. Similar to REST, SOAP is used to "expose" and "consume" webservices
b. XML-based web service protocol originally developed by Microsoft
c. Platform and language independent
d. Legacy protocol: most companies switching to RESTful web services
e. Can be used in conjunction with any transport-layer protocol (HTTP, SMTP, FTP, etc)
   i. When used with HTTP, POST requests are used
   ii. HTTP must set that content type to XML
f. Uses a contract (WSDL)
   i. WSDL = Web service description language
      1. Web Service Description Language
      2. XML file that describes the service
      3. "Contract" or "Endpoint"
      4. WSDL elements are placed within the namespaces like this:
      5. xmlns:xsd, xmlns:soap
      6. WSDL specifies ports, messages, operations, services
   ii. Contract first vs contract last development
g. Built-in security
h. SOAP message elements
   i. envelope - (mandatory) defines the start and end of a message
   ii. header - (optional) specifies attributes)
   iii. body - (mandatory) this is the XML data
   iv. fault - (optional) describes error that may have occurred

57. What are the differences between SOAP and REST?

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API
ACID compliance	Has built-in ACID compliance to reduce anomalies	Lacks ACID compliance