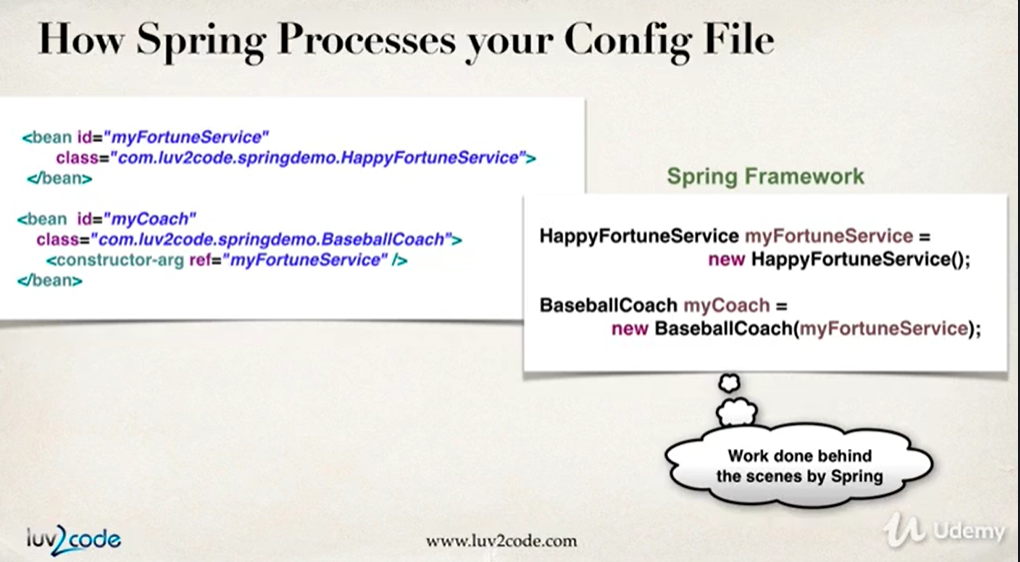
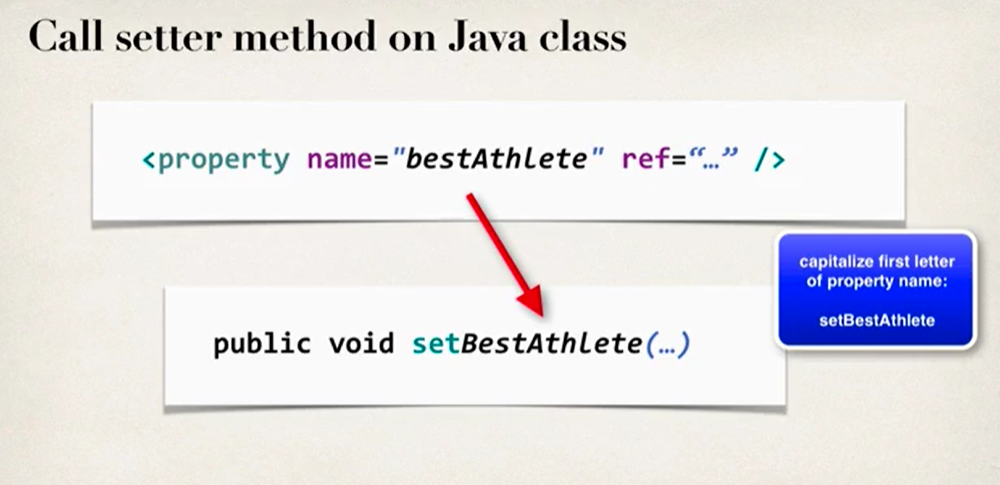
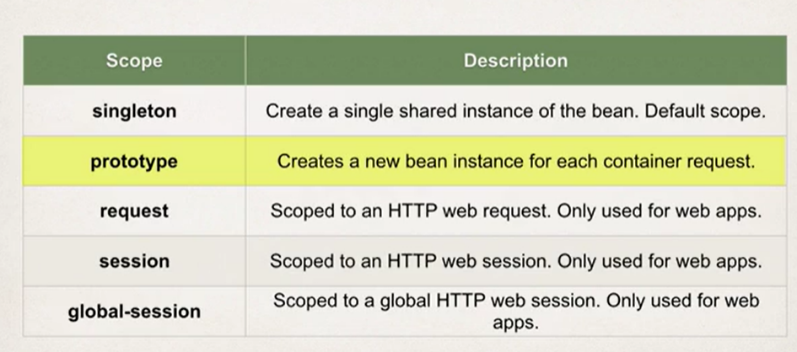
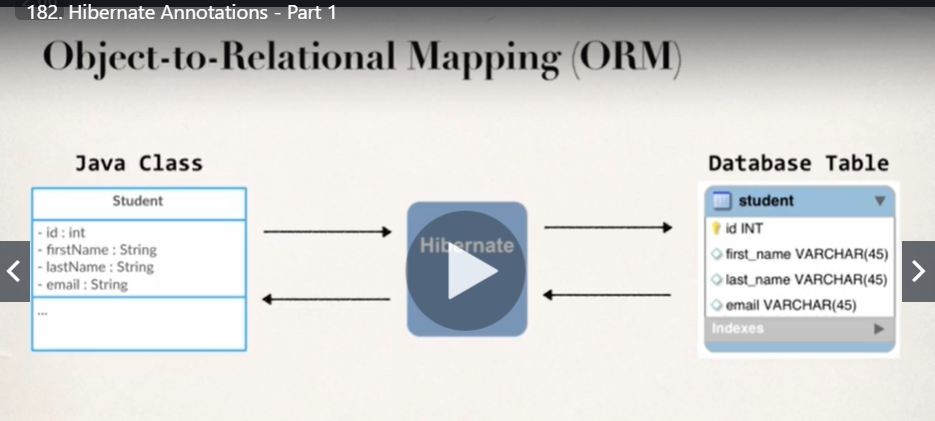
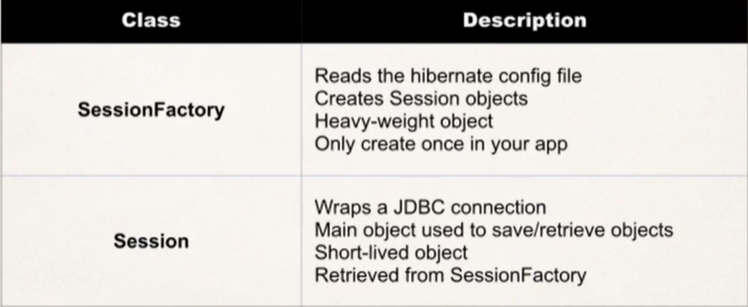
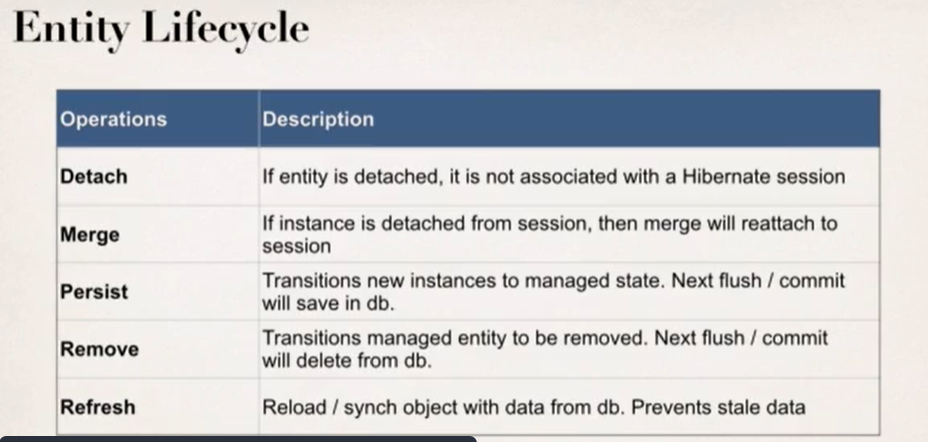
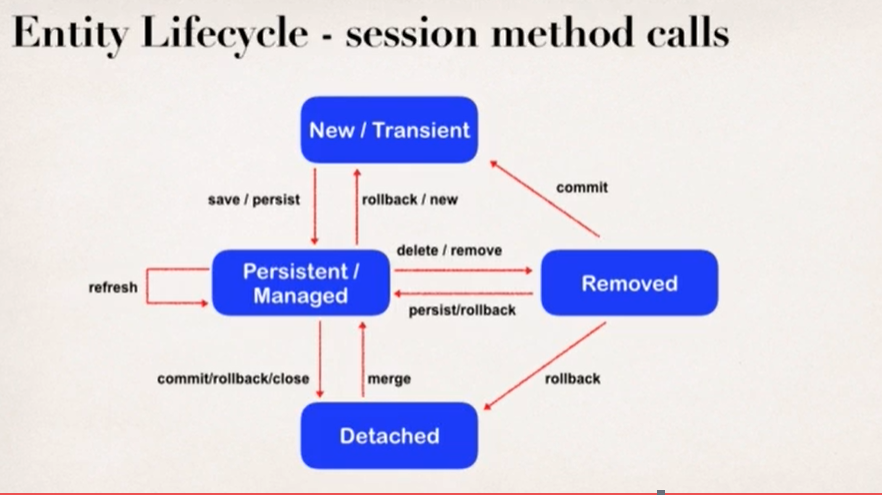
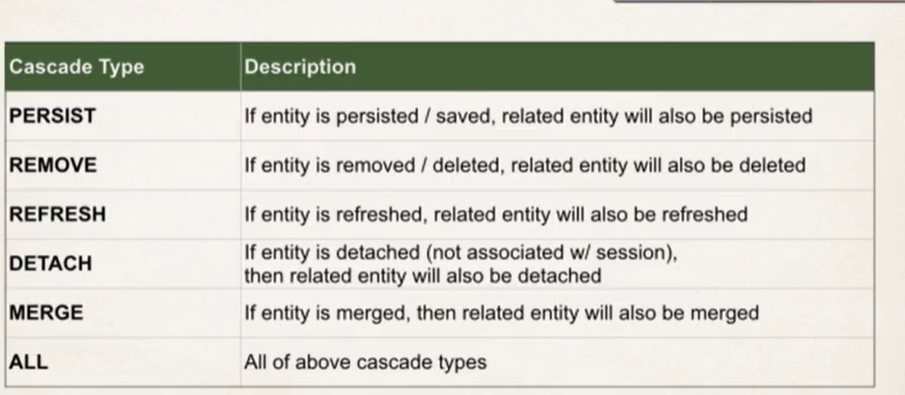
1. **Spring and Java**
   1. <https://spring.io/>
   2. A lightweight framework which is easy to use.
   3. Spring “Projects”: Additional Spring modules built on top of the core Spring Framework. These are optional to use. <https://spring.io/projects>
      1. Example: Spring Boot (Very popular)
   4. Requirements: Spring 5, JDK (Java 8 or higher), Tomcat Application Server, Eclipse IDE.
   5. Java
      1. JRE, JDK, Java SE, JAR files: <https://www.udemy.com/course/java-the-complete-java-developer-course/learn/lecture/3909250#questions/12987160>
      2. Naming convention:
         1. Package: <https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html>
         2. Class names: CamelCase, nouns, start with capital letter.
         3. Method names: mixedCase, verbs.
         4. Constants: All uppercase, with words separated by underscore. Declared using final keyword.
         5. Variable name: mixedCase, start with lowercase, no underscores, meaningful.
      3. <https://www.javatpoint.com/corejava-interview-questions>
      4. A java package is a group of similar types of classes, interfaces and sub-packages.
         1. Two types of packages in java:
            1. Built-in package: java, lang, awt etc.
            2. User-defined package.
         2. Advantage:
            1. Access protection.
            2. Used to categorize the classes and interfaces for easily maintability.
         3. If you import a package, subpackages won’t be imported. Classes and interface of that package (not of subpackage) will be imported.
         4. If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. (One public top-level class per .java file!)
         5. Multiple classes in a java file (not recommended): <https://www.udemy.com/course/java-the-complete-java-developer-course/learn/quiz/4448802#questions/13038176>
         6. Saurabh Shukla packages (explains command so that \*.class file is saved in the package folder): <https://www.youtube.com/watch?v=uIyLyqmhzbM&list=PLX9Zi6XTqOKQ7TdRz0QynGIKuMV9Q2H8E&index=10&ab_channel=JavabySaurabhShuklaSir>
         7. Some packages like java.lang are automatically imported.
      5. Project directory structure: <https://stackoverflow.com/questions/10992028/what-is-the-project-directory-structure-for-a-standalone-java-se-application>
      6. Access Modifiers Best Practices: https://www.codejava.net/java-core/the-java-language/java-access-modifiers-examples-public-protected-private-and-default
2. **Spring Container’s primary functions**
   1. **Spring Inversion of Control – XML Configuration**
      1. For creating and managing objects.
      2. It makes your code loosely coupled.
      3. It also makes it easy for the programmer to write effective unit tests.
      4. Spring Container’s primary functions-
         1. Create and manage objects (Inversion of Control).
         2. Inject object’s dependencies (Dependency Injection).
      5. Configuring Spring Container-
         1. XML configuration (legacy)
         2. Java Annotations (modern)
         3. Java Source Code (modern)
      6. Spring Development Process
         1. Configure your Spring Beans.
         2. Create a Spring Container.
         3. Retrieve Beans from Spring Container.
      7. Spring Container is generally known as **ApplicationContext**.
      8. A **Spring Bean** is simply a Java object. When Java objects are created by the Spring Container, then Spring refers to them as "Spring Beans".
      9. Our app is now configurable. We can easily change the coach for another sport.
   2. **Spring Dependency Injection**
      1. Dependency is same thing as helper object.
      2. Inject object’s dependencies.
      3. Outsourcing the construction and injection of your object to an external entity (like Spring Object Factory).
      4. Two most common injection with Spring
         1. Constructor Injection
         2. Setter Injection
      5. **Constructor Injection Development Process**
         1. Define the dependency interface and class.
         2. Create a constructor in your class for injections.
         3. Configure the dependency injection in Spring config file.
      6. Config file behind the scenes
      7. A shortcut to get Qualified name for a class to use in config file.
      8. A shortcut to generate constructor, getters, setters in a class.
      9. When you don’t define any constructor in your class, compiler defines default one for you, however when you declare any constructor, compiler doesn’t do it for you.
      10. **Setter Injection Development Process**
          1. Create setter method(s) in your class injections.
          2. Configure the dependency injection in Spring config file.
      11. **Injecting literal values.**
          1. Create setter method(s) in your class for injections.
          2. Configure the injection in Spring config file.
      12. Interface name vs Class name when retrieving bean from Spring Container: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5663746#content>
      13. Constructor based dependency injection (multiple arguments): <https://www.tutorialspoint.com/spring/constructor_based_dependency_injection.htm>
      14. If you want spring's IoC container to use the no-arg constructor to instantiate an object for the class, you don't need to write one because it is provided by the JVM.
      15. **Injecting values from properties file.**
          1. Create properties file.
          2. Load properties file in spring config file.
          3. Reference values from properties file.
3. **Bean Scopes and Lifecycle**
   1. Scope refers to the lifecycle of a bean.
      1. Default Scope: Singleton
   2. How long does the bean live?
   3. How many instances are created?
      1. Spring Container creates only one instance of the bean by default. It is cached in memory.
   4. How is the bean shared in the Spring environment?
      1. All requests for that bean will return a shared reference to the same bean.
   5. Some custom methods during bean initialization and bean destroy. They have a void return type and no arguments.
4. **Spring Configuration with Java Annotations**
   1. **Inversion of Control**
      1. Java Annotations: Special labels/markers added to Java classes. Annotations are used to provide supplement information about a program. Start with ‘@’. Processed at compile-time or runtime.
      2. Why Spring Configuration with Annotations? Coz for a large project, XML configuration can be verbose. Annotations minimizes the XML configuration.
      3. Creating a new project: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5306076#overview>
      4. Development Process
         1. Enable component scanning in Spring config file.
         2. Add the @Component Annotation to your Java classes.
         3. Retrieve bean from the Spring container.
   2. **Spring Dependency Injection with Annotations and Autowiring**
      1. **Spring Autowiring**
         1. For dependency injection, Spring can use autowiring. Spring will look for a class that matches the property and will inject it automatically, hence it is autowired.
         2. Autowiring Injection Types
            1. Constructor Injection
            2. Setter Injection (Can use any method name)
            3. Field Injection
      2. When a class implements an interface, you can instantiate an object for the class in the following ways. (Similar in case of inheritance: <https://www.youtube.com/watch?v=QM0iW8fFSXs&list=PLX9Zi6XTqOKQ7TdRz0QynGIKuMV9Q2H8E&index=19&ab_channel=JavabySaurabhShuklaSir>)
         1. ImplementationClass implementationClassReference = new ImplementationClass();
            1. All (well, not all) the methods defined in the interface and overridden in the Implementation class and the methods which are newly defined in the implementation class are accessible with the implementationClassReference.
         2. InterfaceName interfaceReference = new ImplementationClass();
            1. All (well, not all) the methods defined in the interface and overridden in the Implementation class are only accessible (but not the methods which are newly defined in the implementation class) with the interfaceReference.
      3. **Very Important!** The constructors of the parent classes are called, all the way up the class hierarchy through Object, before the child class's constructor is called. Constructors are not inherited.
         1. Saurabh Shukla, Constructor in Inheritance: <https://www.youtube.com/watch?v=CmffeGETyk4&list=PLX9Zi6XTqOKQ7TdRz0QynGIKuMV9Q2H8E&index=21&ab_channel=JavabySaurabhShuklaSir>
         2. Constructor Chaining: <https://www.youtube.com/watch?v=jYu91dljs4s&list=PLX9Zi6XTqOKQ7TdRz0QynGIKuMV9Q2H8E&index=22&ab_channel=JavabySaurabhShuklaSir>
         3. <https://stackoverflow.com/questions/34488484/why-is-super-class-constructor-always-called>
      4. Use @Qualifier when autowiring in case multiple implementations of an interface.
         1. Default bean names are, class name with first letter lowercase. But if in a class, first two letters are uppercase, then the name is not clonverted. We can also give explicit names.
   3. **Bean Scopes and Lifecycle**
      1. JAR files: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/9120288#questions>
      2. For "prototype" scoped beans, Spring does not call the @PreDestroy method.
5. **Spring Configuration with Java Code (no XML)**
6. **Spring MVC- Building Spring Web Apps**
   1. It is a framework for building Web Apps in Java.
   2. Based on Model-View-Controller design pattern.
      1. Model: Contains your data.
      2. View Template: JSP + JSTL (a lot more options available).
      3. Controller: Business Logic.
   3. Debugging Tomcat links provided.
   4. JAR files in WEB-INF folder don’t require build path.
7. **Spring MVC- Creating Controllers and Views**
   1. IMPORTANT: Component Scan while package names are different in MVC- <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5740736#questions/13094592>
   2. How to use CSS, Images and JS files? <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5608584#content>
   3. Deploying to Tomcat using WAR file: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5633776#content>
   4. @RequestMapping will add '/' if we have not added it.
   5. Very Important, Controller Level Request Mapping, Relative Path: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/7218630#questions/11592154>
8. **Spring MVC- Form Tags and Data Binding**
   1. Spring MVC Form Tags- a support for form tags.
      1. Plus they support data binding!
      2. Like the trick Colt explained, storing multiple attributes in an object: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5653308#questions>
      3. It would automatically call the set methods for that object.
      4. So Spring will populate the object automatically.
   2. Spring MVC form tag has a method attribute. You can set the value to get or post. In recent 5.x versions of Spring, default value is post.
   3. @RequestMapping handler methods have a flexible signature and can choose from a range of supported controller method arguments and return values. That is Spring MVC handler methods are very flexible in terms of args passed.
   4. Taglibs in jsp file!
   5. <https://stackoverflow.com/questions/5322632/spring-expression-language-spel-with-value-dollar-vs-hash-vs>
9. **Spring MVC Form Validation- Applying Built-in Validation Rules** 
   1. Hibernate has a implementation of validation framework.
   2. <http://hibernate.org/validator/>
   3. When performing Spring MVC validation, the location of the BindingResult parameter is very important. In the method signature, the BindingResult parameter must appear immediately after the model attribute.
   4. @Valid performs validation rules on the passed in Customer object. And results of those validation in BindingResult.
   5. @InitBinder annotated custom method will be called for every web request. We can write custom code to remove leading and trailing whitespaces in the input data. Part of preprocessing.
   6. Model is a just an object whose scope is restricted to a request.
10. **Spring MVC Form Validation- Validating Number Ranges and Regular Expressions**
    1. A Java both int and Integer are used to store integer type data the major difference between both is type of int is primitive while Integer is of class type.
       1. In order to check for null we must use the appropriate wrapper class: Integer.
    2. How to override the errors and load a properties file containing those custom error messages.
11. **Spring MVC Form Validation- Creating Custom Validation Rules**
    1. Custom Validation: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/6724848#questions/11004158>
12. **Introduction to Hibernate**
    1. A framework for persisting/saving java objects in the database.
    2. <http://hibernate.org/>
    3. Use it for saving/retrieving data from the database.
    4. It provides ORM that is Object-to-Relational Mapping.
    5. Hibernate is on top of JDBC. It handles the lower level SQL code to help us. It uses JDBC in the background to communicate with the database.
13. **Setting up Hibernate Development Environment**
    1. Creating new in MySQL Workbench: <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5116978#questions/13202102>
14. **Hibernate Configuration with Annotations**
    1. Hibernate config file contains information like JDBC url, username, password so that Hibernate with the help of JDBC in background can connect to the database.
    2. Entity Class- Java class that is mapped to a database table.
    3. Object-to-Relational mapping or ORM- table is mapped to class and fields to columns.
    4. There are two ways of mapping-
       1. XML configuration (legacy)
       2. Java Annotations (modern, preferred)
15. **Hibernate CRUD Features**
    1. Primary key can’t be null.
    2. For every operation with Hibernate, it must run in the context of a transaction. This applies even for read-only queries. Hence always create a session, and end it with session.getTransaction().commit();
       1. How many CRUD operations in one transaction? <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/9060648#questions/13190026>
       2. <https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/7743980#questions/13248848>
       3. It’s always good to commit after one transaction and try to limit number of operations in a transaction to one! Even you perform multiple operations in a single transaction, that is not gonna hurt but not a great practice (straight from the hibernate docs).
    3. HQL (Hibernate Query Language) like SQL for querying the tables.
    4. A cool way to refactor code. Select a block of code, right click > refactor > extract method.
16. **Hibernate Advanced Mappings - @OneToOne**
    1. Entity lifecycle
    2. Entity lifecycle – session method calls
    3. Cascade delete means if two tables are linked by a foreign key, then deleting an entry in instructor (parent) table will also delete the corresponding entry in the instructor\_detail (child) table.
       1. But we can’t apply cascade delete everywhere for example, not in many-to-many relation.
       2. There are many other cascade types.
          1. @OneToOne cascade types
    4. We can get the database diagram to include in documentation through reverse engineer option on MySQL Workbench.
    5. Schema and Database are used interchangeably in MySQL Workbench.
17. **Hibernate Advanced Mappings – Eager vs Lazy Loading**
    1. Prefer lazy loading over eager loading for good app performance.
    2. In case of lazy loading, the dependent data is loaded only on demand. However, requires an open Hibernate session. Needs a connection to database to retrieve data.
    3. Setting breakpoints and then, Debug As > Java Application.