Core Java Task

1. Complete customer dashboard
2. Create customer table in database with following columns and all varchar  
   email, custname, city, phone, password;
3. Add some dummy customers in customer table
4. Add DbConnection class in CMS java project and make sure it is connected to Db

Hibernate

<https://docs.jboss.org/hibernate/orm/6.1/userguide/html_single/Hibernate_User_Guide.html#architecture>

1. ORM framework => object relational mapping
   1. Mapping of java objects to database table
   2. Mapping of data types -> automatic type conversion [ XXXDialect ]
   3. Mapping of different types of relationships
      1. Java -> isa hasa, equals
      2. DB -> PK, FK, CK, UK, Joins
   4. Creation of sql queries is taken care
   5. DB connection, loading driver and executing queries
   6. Handing exceptions
   7. By default commit false
2. Creating a simple hibernate maven project
   1. Create a quickstart Maven project
   2. Add dependencies in pom.xml file
      1. Hibernate-core
      2. MYSQL [ DB driver ]
   3. Hibernate configuration file => hibernate.cfg.xml [src/main/resources]
      1. Connection parameters
      2. Dialect
      3. Other DB related configurations
   4. Create entity classes
   5. To map java class with database tables
      1. XML based configuration
      2. Annotation based configurations  
         <mapping class=”” or resources=”.xml”>
   6. Load the configuration file
   7. Tables were created automatically by hibernate => No  
      DB first approach
   8. CRUD operations
      1. Save or persist [IMP] => insert
      2. Merge[IMP] or update => update
      3. Delete or remove [IMP] =>delete
      4. Get(<name of the class>, id) or load=> Select by id
      5. createQuery(<string>, return type) => Select all
   9. HQl queries => where, order by, group by, having or aggregate functions

Or palceholders :id

1. OneToOne => 1 User has 1 Vehicle
   1. Unidirectional
   2. Bidirectional => mappedBy
   3. Can change the column name : @JoinColumn
2. Embeddable (on the class) and Embedded (on the property)
3. ElementCollection => can be used on collections of type primitives
   1. @JoinTable => to specify the name of the joined table
4. OneToMany => can be used on collections of type user defined data types [ class ]
   1. By default it created a 3rd table
   2. @JoinColumn -> will not create the 3rd table
5. ManyToMany
6. Inheritance

**SPRING CORE**

1. 3 Core principles
   1. DI – Dependency Injection
   2. AOP – Aspect Oriented Programming
   3. Abstraction layer
2. Steps to create a spring project – Quickstart Maven
   1. Add dependencies => spring-context
   2. Create a class with getters/setters
   3. Create spring configuration file -> spring.xml
      1. Configure the beans using <bean>  
         id is optional -> when there is no ambiguity [ only 1 instance of the bean is configured ]
      2. By default all beans are eagerly loaded ->   
         lazy-init => to lazily load the beans
      3. For constructor injection  
         <constructor-args name=”propname” value=””> - for primitives value attribute   
         <constructor-args name=”propname” ref=””> - for object types use ref
      4. For setter injection  
         <property name=”propname” value=””> - for primitives value attribute   
         <property name=”propname” ref=””> - for object types use ref
      5. Scope =”singleton” prototype(for factory objects)
   4. Load the configuration file
      1. ApplicationContext context = new ClassPathXmlApplicationContext(“spring.xml”)
   5. To access the instances created by spring
      1. Context.getBean(“classname.class”)
      2. Context.getBean(“idvlaue”)
3. Annotation based configurations
   1. Property[setter]/ constructor/ field
      1. @Value – which injects values of type primitives
      2. @Autowired – which injects dependencies of reference type
   2. @Component – which is used on the class which tells spring to load this class and instantiate it
   3. @Primary – the class that is annotated with this annotation will be the default reference to be autowired
   4. @Qualifier – if there are more than 1 bean of same type then to resolve the ambiguity use this annotation and it takes precedence over @Primary  
      By default the ids of the classes/ beans are created with camel casing of the bean  
      you can override the default by specifying the name within @Component
   5. @Configuration – that provides java based configuration and it is annotated on the class that provides with configurations to your project  
      DB configuration  
      Web configuration

Security configuration

* 1. @ComponentScan – is used on the configuration class to tell spring to scan the classes with @Component annotation  
     It looks for the classes in the root package of the configuration class and the sub packages

1. Lifecycle of spring bean
   1. Beans are loaded
   2. Instantiated -> constructor is invoked
   3. ApplicationCOntext is set
   4. BeanNameAware -> that assigna an id to the bean
   5. BeanFactory -> where all the beans are registered
   6. Set some initial properties for the bean
      1. XML approach init-method, destroy-method
      2. Interfaces to implement
         1. DisposableBean
         2. InitializingBean
      3. Annotation based approach :  
         <dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

* + - 1. @PostConstruct
      2. @PreDestroy

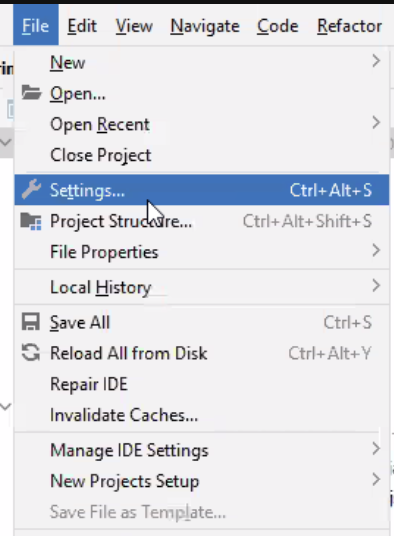
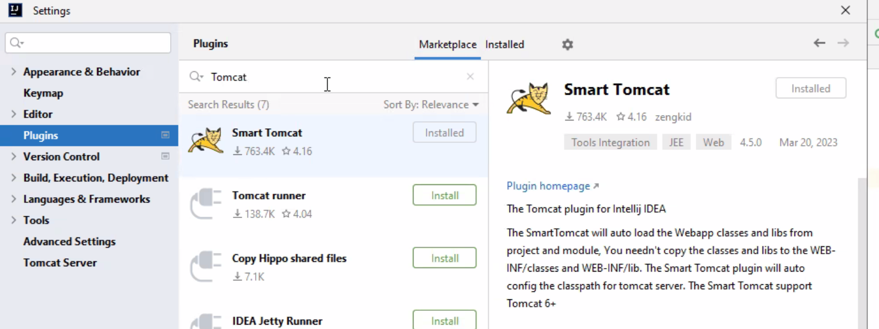
1. Event handling in spring beans
   1. Event -> extends ApplicationEvent  
      the source i.e the publisher that raised the event and any other extra info
   2. Listener -> implements ApplicationListener or can use @EventListener on the method
   3. Publisher -> implements ApplicationEventPublisherAware or can autowire the ApplicationEventPublisher object using which an event can be published and listener handles the event

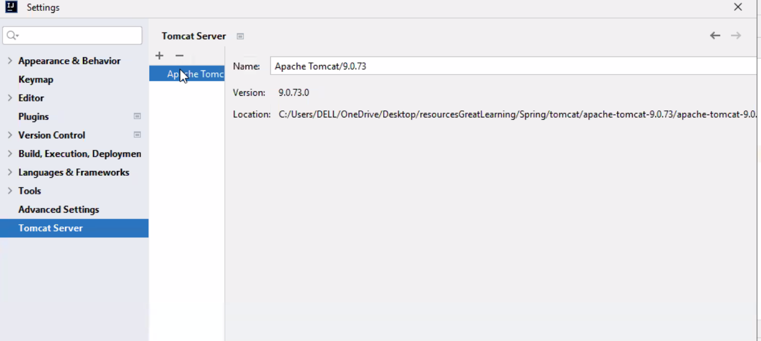
**SPRING JDBC**

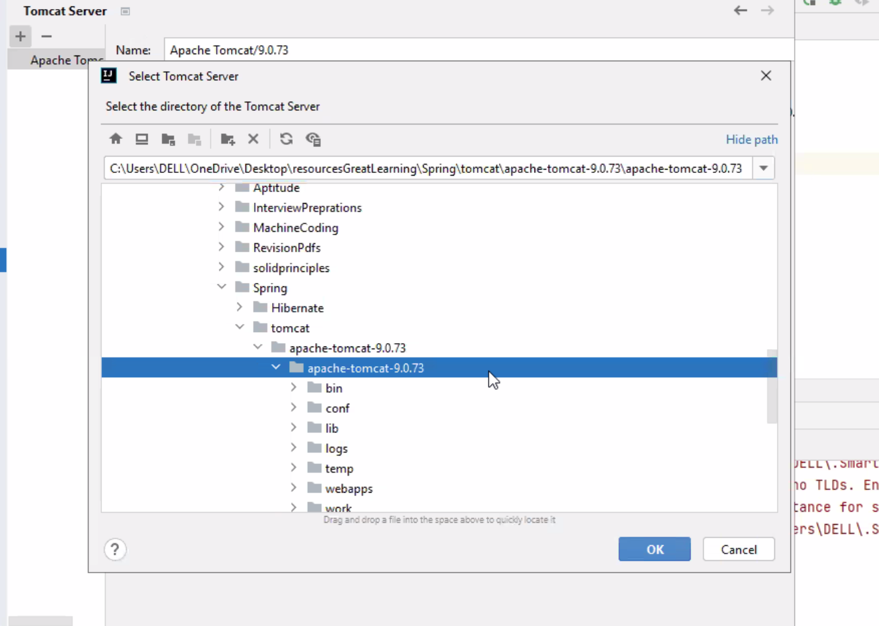
1. Create a quickstart maven project
2. Add dependencies
3. Create a configuration class
   1. Create a DataSource object and provide DB connection parameters
   2. Create a JdbcTemplate object
4. Create a bean that is mapped with DB table
5. Create a CustomerDB class and inject the JdbcTemplate object using which sql queries can be executed

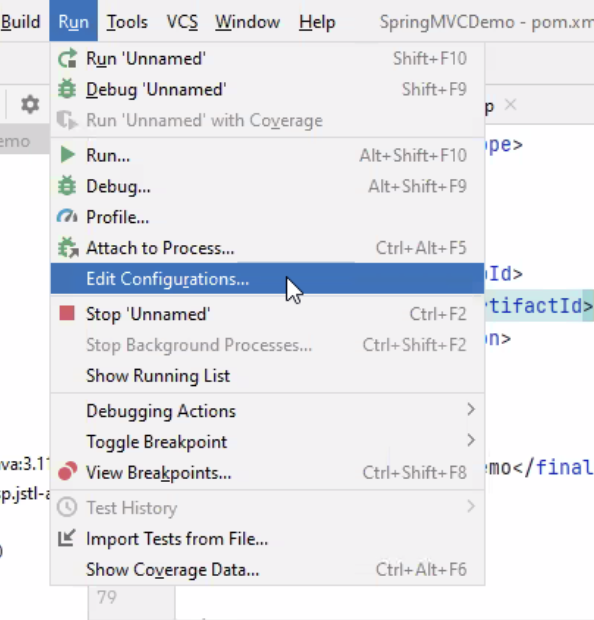
SPRING MVC

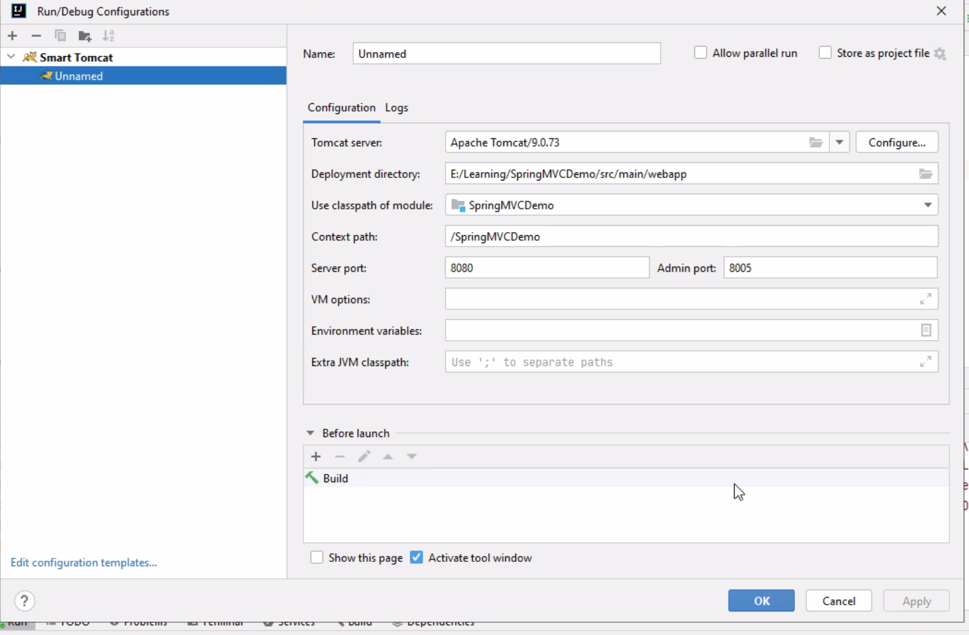
Tomcat setup for intellij

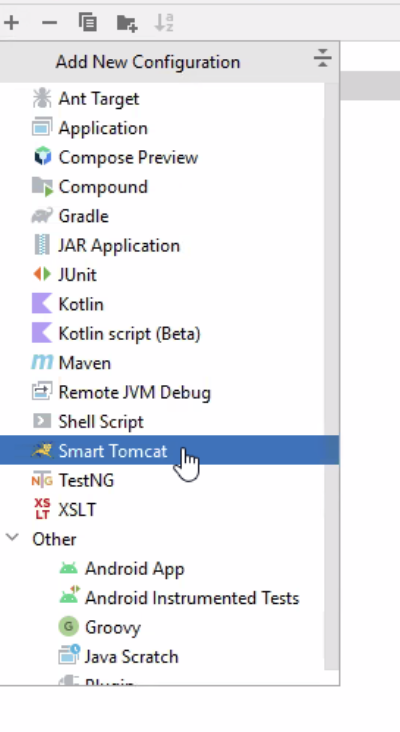
 











1. Create a maven webapp project
2. Add dependencies in pom.xml
3. Eclipse
   1. download tomcat server zip file and unzip it.
   2. Project Run As-> Run on server -> select tomcat and browse to the downloaded folder
4. Intellij : screenshots above
5. Create views folder under webapp/WEB-INF package and add hello.jsp within views folder
6. Create 2 configuration files to configure DispatcherServlet(ApplicationContextConfig.java) and ViewResolver(SpringWebAppInitializer)
7. Create a controller and configure it to return view
8. Add link in index.jsp page