ISC506 · WEB AND INTERNET PROGRAMMING

Enterprise Application is a software application typically used in large business organizations

Feature of Enterprise Application

1- Support for concurrent users and external systems.
2 - Support for synchronous and asynchronous communication using different protocols.
3. Ability to integrate with back-end systems and web services.
4- Support for scalability to handle future growth.
5 - Support for highly secure access control for different types of users.

Java Enterprise Edition (Java EE) is a specification for developing enterprise applications using Java. It is a platform-independent standard that is developed under the guidance of the Java Community process (JCP).

Java EE 7 specification consists of no of component application programming interfaces (API) that are implemented by an application server

- The Red Hat JBoss Enterprise Application platform (EAP) implements the Java EE standard.

Benefits of Java EE Enterprise Application

1- Application are portable across Java EE compliant application servers due to Java EE standard.
2- platform-independent applications can be developed and will run on many different types of OS (on small PCs as well as large mainframes).
3- The Java EE specification provides a large no of APIs typically used by enterprise applications such as web services, transactions, database connectivety, batching utilities, security and asynchronous messaging
4- A large no of sophisticated tools such as IDEs, monitoring systems, enterprise application Integration (EAI) frameworks, and performance measurement tools are available for Java EE applications from third-party vendors.

Comparing Java EE and Java SE

* Java Development Kit (JDK) - provides the compiler, debugger, tools and runtime environment for hosting java applications.
* Java Virtual Machine (JVM) and a large set of reusable component classes that are commonly used by application
* This API provides packages & classes for networking, I/O, Parsing, database Connectivity, developing GUI and many more. This API is commonly known as the Java Standard Edition (Java SE).

* Java SE is generally used to develop stand-alone programs, tools, and utilities that are mainly run from the command line, GUI programs and server processes that need to run as daemons.
* The Java EE specification is a set of APIs built on top of Java SE. it provides a runtime environment for running multi-threaded, transactional secure and scalable enterprise applications.

* Java is mainly a set of standard specifications for an API, and runtime environments that implements these APIs are generally called application servers.
- An application server that passes a test suite called Technology Compatibility Kit (TCK) for Java EE is known as a Java EE Compliant application server.
- Java EE includes support for multiple profiles or subsets of APIs eg Java EE 7 specification defines two profiles i- Full profile and ii- Web profile
i- Full profile contains all APIs defined by Java EE 7 (including all E items in e web profile). When developing EJBs, messaging apps, and web services (in contrast to web apps), you should use the full profile.
ii- Web profile - is designed for web application development and supports a subset of the APIs defined by Java EE 7 related web-based technologies.

Building, Packaging and deploying Java SE and Java EE Applications.
For relatively simple standalone Java SE apps, the code can be built, packaged, and run on E command line by using the compiler and run time tools (java, javac, jar, jdb and so on) that are part of the JDK.
- several mature (IDEs) such as RedHat JBoss Developer Studio or Eclipse, are used to simplify the building and packaging process
The preffered way to ship standalone Java SE app in a platform neutral way is to package E application as a Java Archive (JAR) file.

* Manifest entries (a plain text file packaged alongside the Java classes inside E Java file.

Deployment process for Java EE apps
1- JAR files - individual modules of an application and Enterprise Java Beans (EJBs) can be deployed seperate as JAR files third-party libraries and frameworks are also packaged as JAR file.
2- Web Archive files (WAR) files - If your Java EE app has a web based front-end or is providing RESTful service endpoints, the code and assets related to E web front-end and E services can be packaged as a WAR file
3- Enterprise Archive (EAR) files - An EAR file has an extension of .ear. It is useful in scenarios where E application contains multiple WAR files or reuses some common JAR files across modules.
* Apache Maven to simplify building, packaging, testing, executing and deploying Java SE and Java EE apps.

Quiz 1

1- Which of E following two statements can be considered an enterprise app?
a) An online banking system for a bank with millions of customers.
b) An online payment gateway for a credit card company that processes millions of transactions per day.
2- Which of E following two statements about E Java EE specification and application servers are correct?
c) A fully Java EE compliant application can be deployed on different Java EE compliant servers without recompiling and re-implementing features.
d) a Java EE compliant application server provides facilities for asynchronous messaging.

(3) which statement describing a java EE compliant application server

(c) The application server provides automatic transaction management.

(4) A company named ABC Inc is migrating a large complex legacy mainframe-

(c) Role-based security.

(d) Batch operations for scheduled execution of a reporting application, that generates reports from RDMS on daily, monthly & quarterly basis.

(5) - which of the following two statements about deployment types in Java EE 7 are correct

(b) web apps are typically packaged as WAR files for deployment to an application server

(d) An EAR can contain WAR files, JAR files and deployment descriptors.

6 - which of the following two statements about Apache Maven are correct?

(a) Maven can be used to build, package and test both Java EE and Java SE apps.

(d) Maven can automatically deploy and undeploy apps from JBoss EAP, there is no need to restart the application server after every deployment and undeployment.

Multi-Tiered Application Architecture

Java EE apps are designed with a multi-tier architecture in mind.

- The app is slit in2 components, each serving a specific purpose.
- Each component is arranged logically in tier
- Some of the tiers run on seperate physical machines or servers while the actual data for the database can be stored on a seperate server
- The application's business logic can run on application servers hosted in one data center.

In a classic web-based Java EE application, there are four tiers

i - Client tier - This is usually a browser for rendering the user interface on the end-user machines, or an applet embedded in a web page (increasingly rare)

ii - Web tier - Components run inside an application server and generate HTML or other markup that can be rendered or consumed by components in the client tier. This tier can also serve non-interactive clients such as other enterprise systems (both internal and external) via protocols such as Simple Object Access protocol (SOAP) or Representational State Transfer (REST) web services.

iii - Business logic tier - The components in the business logic tier contain the core business logic for the application. These are usually mix of Enterprise Java Beans (EJBs), POJO, Entity Beans, Message driven Bean, Data Access Objects (DAO) which interface with persistent storage such as RDMS, LDAP and others.

iv - Enterprise Information system (EIS) Tier - many enterprise applications store and manipulate persistent data that is consumed by multiple systems and applications within an organization. eg RDMS, lightweight Directory Access protocol (LDAP) directory services, NOSQL database, in-memory databases, mainframes or other backend systems that store and manage an organization's data securely.

Types of Multi-Tier Application Architecture

1 - web-centric architecture - This type of architecture is simple applications with a browser-based front end and a simple backend powered by Servelets, Java Server Pages (JSP)

Or Java Server faces (JSF) features such as transactions, asynchronous messaging, and database access are not used.

ii- Combined web and business logic Component-based architecture - In this architecture, a browser is the client tier interfaces with a web tier consisting of servelets, JSPs, or JSF pages which are responsible for rendering the User Interface, Controlling page flow and security. The core business logic is hosted in a separate business logic which has Java EE Components such as EJB, Entity Beans

iii - Web service application architecture — Modern application architectures often use web services, providing an API accessible via HTTP-based protocols like SOAP or REST. These services are consumed by non-interactive apps or interactive front-ends using frameworks like AngularJS, backbone.js and React.

RED HAT JBOSS DEV STUDIO

Is an Integrated Development Environment (IDE) provided by Red Hat to Simplify the development of Java EE applications. It is a set of integrated and well tested plug-ins on top of the Eclipse™ platform.

Built-in features

i - plug-ins to simplify development of apps using Red HAT JBoss middleware

ii - Unit testing plugins and wizards to do Test Driven Development (TDD)

iii - A visual debugger to help debug local and remote Java applications.

iv - Syntax highlighting and Code completion for most commonly used Java EE APIs, such JPA, JSF, JSP, EL and many more

v - Maven integration to simplify project builds, packaging, testing and deployment

vi - Unit Adapters and plug-ins to work with JBoss EAP, you can control the life cycle (start, stop, restart, deployment, undeployment) of EAP without leaving the IDE (Integrated Development Environment)

APACHE MAVEN

Is a project Management tool that uses a declarative approach (in an XML file called pom.xml at the root of the project folder) to specify how to build, Package execute (for Java EE apps) and deploy apps Logged with dependency information.

features / plug-ins of Apache Maven

1- predefined build life cycles for end products, called artifacts like WAR, EAR, and JAR.

2- Build-in best practices such as source file locations and running unit test for each build.

3- Dependency management with automatic downloading of missing dependencies.

4- Extensive plug-in collection including plug-ins specific to JBoss dev or deployment

5- project report generation including Javadocs, test coverage and many more.

- The group-id is like a java package
- " artifact-id is a project name
- " packaging defines how the project will be packaged.
- A Dependency describes the resources a project depends on.
- " plugins for the project

Maven Directory structures

Asset	Directory	Outcome
Java Source code	src/main/java	E directory contains Java classes included in WEB/INF/classes for WAR root or JAR
Configuration files	src/main/resources	" " " Configuration files in " " " " " "
Java Test code	src/test/java	E directory " the test source code
Test config files	src/test/resources	" " " " the test resources

Maven Dependency Scope

Scope	Outcome
Compile	Compile is E default scope....
test	Test is required to compile and run E unit test. It is not included in E artifact
runtime	The runtime dependency is not required for compilation, it is used for any execution and is included in E artifact.

Maven Common Commands

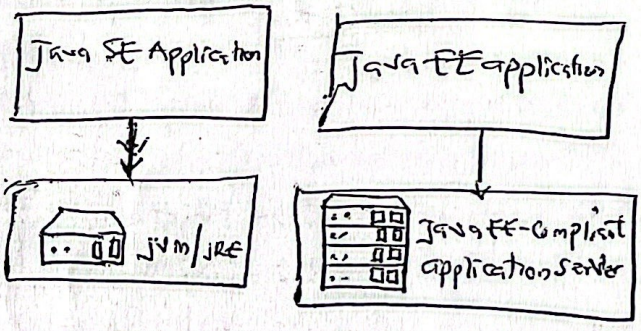
* Mvn package - Compiles, tests and build E artifact
* Mvn package -Dmaven.test.skip=true - builds the artifact and skip all tests.
* mvn wildfly:deploy - To deploy E artifact to E instance running at $JBOSS_HOME
* Mvn install - Installs E artifact in ur local maven repository for use in other projects as a dependency.

To build, package, and run a standalone application that uses only E Java SE Api with Maven, you will run E following Commands

```
$ Mvn clean package
$ java -jar target/myapp.jar
```

CHAPTER 2

Application Servers - is a software component that provides E necessary runtime environment and infrastructure to host and manage Java EE enterprise applications. It provides fertures such as Concurrency, distributed component architecture, Portability to multiple platforms, transaction management, web services, Object relational mappings for databases (ORM), asynchronous messaging, and security for enterprise apps.



JBOSS ENTERPRISE APPLICATION PLATFORM (EAP)

Red Hat JBoss Enterprise Application platform 7, JBoss EAP 7, or simply EAP is an application server to host and manage Java EE apps.

* EAP 7 is built on open standards, based on E wildfly open source software, and provides E following features.

1- It is certified for both javaEE 7 full and web profiles

2- A Centralized management of multiple server instances and physical hosts.

3- A reliable, Standards, Compliant, light-weight and supported infrastructure for deploying application

4- A modular structure that allows Users to enable services only when they are required.

5- preconfigured options for features such as high availability clustering, messaging and distributed caching are also provided.

Containers - A container is a logical component within an application server that provides a runtime context for applications deployed on the application server.

- Acts as an interface btw E application components and the low-level infrastructure services provided by the application server.

Types of Containers within a java EE application server

i- Web Containers - Deploy and configure web components such as servlets, JSP, JSF and other web-related assets.

ii- EJB Containers - Deploy and configure EJB, JPA, and JMS-related components.

* Containers are responsible for security, transactions, JNDI lookups and remote connectivity and more.

* Containers can also manage runtime services such as EJB and web components life cycles, data source pooling, data persistence and JMS messaging.

Java EE 7 profiles

A profile is a set of component APIs that target a specific application type.

i- Full profile - Contains all java EE technologies, including all APIs in the web profile as well as others

ii- Web profile - Contains a full stack of Java EE APIs for developing dynamic applications.

Packaging and deploying Java EE Applications

Java EE applications can be packaged into various deployment types, including compressed archive files containing classes, application assets, and XML deployment descriptors, depending on the application type and components.

Three (3) Most common types of deployment

1- JAR files - Contain plain old java object (POJO) classes, JPA Entity Beans, utility java classes, EJBs and MDBs. When deployed into an application server, depending on the type of components inside the JAR files the application server looks for XML deployment descriptors, or code-level annotations, and deploys each component accordingly.

2- WAR files - is used for packaging web applications. It can contain one or more JAR files, as well as XML deployment descriptor under the WEB-INF or WEB-INF/classes/META-INF folders.

3- EAR files - An EAR file contains multiple JAR and WAR files, as well as XML deployment descriptor in the META-INF folder.

example

```
<plugin>
<groupId>org.wildfly.plugins</groupId>
<artifactId>wildfly-maven-plugin</artifactId>
<version>${version.wildfly.maven.plugin}</version>
</plugin>
```

To build, package, and deploy an app to EAP

```
$ mvn clean package wildfly:deploy
```

To undeploy an app from EAP

```
$ mvn clean wildfly:undeploy.
```

CHAPTER 3

An Enterprise Java Bean (EJB) is a java EE component typically used to encapsulate business logic in an Enterprise.

Advantages of EJBs

i- EJBs provide low-level system services, such as multi-threading and concurrency, with requiring the developer to write code explicitly for these services.

ii- Client code is simplified bcoz the client can focus on just the user-interface with missing business logic.

iii- EJB components can be secured for access on a group or role basis.

iv- EJB provides transactional capabilities to enterprise application.

v- EJB can be accessed by multiple different types of clients, ranging from stand-alone remote clients, other java EE components or web service clients using standard protocols like SOAP or REST.

Two(2) Types of EJB

1- Session: performs an operation when called from a client. Usually an application's core business logic is exposed as a high-level API (session Facade pattern) that can be distributed an can be accessed over a no of protocols. (RMI, JNDI, web services).

2- Message Driven Bean (MDB) - Used for asynchronous communication btw components in a java EE application and can be used to receive Java messaging service (JMS) compatible messages and take some action based on the content of the received messages.

Session Bean - provides an interface to clients and encapsulates business logic methods that can be invoked by multiple clients either locally or remotely over different protocols.

Three (3) types of session beans

1- stateless session Beans (SLSB) does not maintain conversational state with clients btw calls.

- A stateless session bean is useful in scenarios where the application has to serve a large number of clients concurrently accessing the bean's business method.
- A stateless bean is also the preferred option for exposing SOAP or REST service end-points to web services clients.

2- stateful session Beans (SFSB) maintain conversational state with client across multiple calls.

- stateful session beans are used in scenerious where conversational state has to be maintained with a client for the duration of the interaction
- stateful must be scoped as private

3- Singleton Session Beans are session beans that are instantiated once per application and exists for the life cycle of the application.

- Unlike (SLSB) which are pooled by application server, there is only a single instance of a singleton session beans in memory
- Similar to (SLSB), singleton session beans can also be used for implementing web service endpoints

Converting a POJO to an EJB

Converting a POJO to an EJB is simple process of annotating the POJO with one or more annotations defined in the Java EE standard and running the resultant EJB in the context of an application server.

* POJO has four business methods
i- add ii- update
iii- find iv- delete to do item

- To convert pojo to a stateless session EJB

```
@Stateless
public class TodoBean {
...
}
```

- To convert this pojo to stateful ---

```
@Stateful
public Class TodoBean {
...
}
```

- To convert this pojo to singleton ---

```
@Singleton
public Class TodoBean {
...
}
```

* for a singleton to perform some initialization, add the @Startup annotation

* His also possible to annotate an initialization method with the @PostConstruct annotation, which tells the EJB Container to call that method immediately after instantiating the EJB

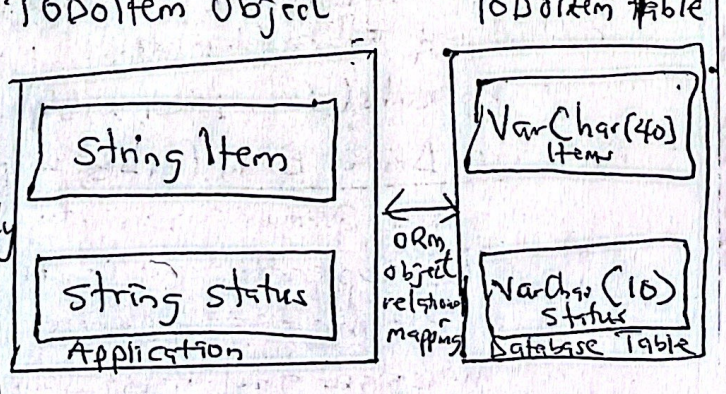
* Init() method to setup the initial state

eg

```
@Singleton
@Startup
public Class TodoBean {
@PostConstruct
public void init() {
// do some initialization
}
...
}
```

CHAPTER 4

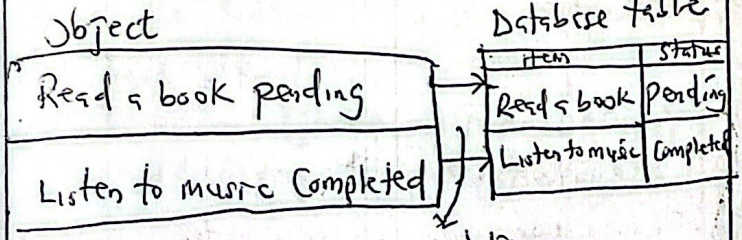
- Persistence — is when an application stores data to a permanent store like a flat file, XML file or a database for durability
- Relational Databases are one of the most common data stores an enterprise application uses to preserve data to reuse.
- Business data in Java EE enterprise application is defined as Java objects
- These objects are preserved in corresponding database tables.
- Java objects and database tables use different data type such as String in Java and Varchar in a database



Impedance Mismatch.

- As data moves between the application and the database as a result of write operations, it can cause differences btw the object model and the relational model. This is disprepancy is known as an Impedance Mismatch.

- Object Relational Mapping (ORM) is the technique to automate bridging the impedance mismatch.
- ORM software uses metadata to describe mapping between the classes defined in an application and the schema of a database table.



Object-relational mapping

* Java EE provides the Java persistence API (JSR 338) specification that is implemented by various ORM providers.
- EclipseLink and Hibernate are example of ORM software.
- A fully implemented ORM provides optimization techniques, caching, database portability, query language in addition to object persistence.

Entity class and Annotations

An entity is a lightweight domain object that is persistable.
- An entity class is mapped to a table in a relational database.
- Each instance of an entity class has a primary key field.
- In Java an entity is a pojo class that is annoted with @Entity annotation.

- Persistent fields - are all the fields in an entity class stored in the database by default.

Declaring Entity class

```
import javax.persistence.*;
import java.io.*;
@Entity
public class ToDoItem implements
Serializable {
@Id
...
```

Entity	Table
Entity class	Table name
Attributes of entity class	Columns in a database table
Entity instance	Record or row in a database table.

@Entity - orm.xml can be used instead of @Entity.

The @Entity annotation specifies that a class is an Entity.

The orm.xml contains all configuration details required to declare a java class as an entity.

@Table - The @Table annotation is used to specify mapping btw an entity class and a table.

```
Eg
@Entity
@Table(name="ThingsToDo")
public class ToDoItem {
...
}
```

The ToDoItem entity class is mapped to the ThingsToDo table

@Column

The @Column annotation is used to map a field or property to a column in the database.

@Column(name="itemname")
private String item;
...
i.e an item attribute is mapped to the Column itemname in the table.

@Temporal — The @Temporal annotation is used with a Date type attribute.
Temporal annotation manages mapping for a
java.util.Date or
java.util.Calender eg

@Entity
public class TODOItem implements Serializable {
@Temporal (TemporalType.DATE)
private Date CompletionDate;

@Transient — is used to specify a non-persistent field.

@Transient
private int CountPending;

The CountPending field is not saved to the database table.

@Id — is used to specify the primary key

@Id
private int id;
...
}

Composite primary key class is defined @EmbeddedId or IdClass is used to specify the composite primary key

ID Generation

Every Entity Instance is mapped to a row in a database table.
- Each row in a table is Unique and is identified by a unique ID known as a persistent entity identity.
- The persistent entity is generated from the primary key field
- A primary key field is required in every entity class

A simple primary key should be one of the following
i- Java primitive types: byte, short, int long or char
ii- The java.lang.String type
iii- Java wrapper classes for primitive types: Byte, Short, Integer, Long, or Character.
iv- Temporal types: java.util.Date, or java.sql.Date.
- @GeneratedValue is applied to the primary key field or property to specify the primary key generation strategy.

GenerationType.AUTO — The AUTO strategy is the default ID generation strategy and means that the JPA provider uses any strategy of its choice to generate the primary key. eg

@Id
@GeneratedValue(GenerationType.AUTO)
private int id;
...
}

GenerationType.SEQUENCE — The SEQUENCE strategy means that the JPA provider uses the database sequence to generate the primary key.

```
@Id
@GeneratedValue(GenerationType.
SEQUENCE, generator="ITEMS_SEQ")
private int id;
...
}
```

GenerationType.IDENTITY — The IDENTITY strategy means that the JPA provider uses database identity column to generate the privaty key. eg

```
...
@Id
@GenerateValue(GenerationType.IDENTITY)
private int id;
}
```

GenerationType.TABLE — The TABLE strategy means that the JPA provider uses database ID generation table.

EntityManager Api is defined to perform persistence operations.

- Obtains a reference to an entity and perform the actual CRUD (Create, Read, Update and Delete) operations on the database.
- EntityManager instance can be obtained from an EntityManagerFactory object
- Entity manager works within a set of managed entity instances called Persistence Context.
- persistence.xml file stored in an application archive.
- persistence.xml is a configuration file that contains info abt the entity classes, data source, transaction type and other configuration info.

Creating EntityManager in EJB

An EntityManagerFactory object is created for a persistence unit and the object is used to obtain an instance of EntityManager.

Producer Technique is another way of obtaining an EntityManager instance in Java EE.

- An object can be injected using Context Dependency Injection (CDI)

CDI — is a set of component management services that allow type-safe dependency injection.

eg

```
Public Class EMProducer {
  @Produces
  @PersistenceContext(UnitName="
ItemPU")
  private EntityManager em;
}
```

- An EJB class can inject the EntityManager using @Inject annotation

```
@Stateless
Public Class ItemService {
@Inject
private EntityManager em;
...
```

- A persistence unit describes configuration settings related to a data source, transactions, concrete classes, and object-relational mapping (ORM)
- persistence unit is configured in a Persistence.xml file in the application's META-INF directory.

Impact of Transactions on persistence

When working with persistence, transactions ensure that changes to a database do not partially complete as a result of an operation failure

- JPA provides transactional behavior for operations on JPA resources using two approaches for transactions

1- Resource Local Transactions

2- JTA Transaction

1- Resource local Transactions are transactions with a scope spanning a single resource, such as a data source.

2- JTA Transactions span all resources in a container

Creating an Entity class

An Entity has several important distinctions that require management by the EntityManager. To Convert a POJO class to an Entity, ~~the~~ prepend an @Entity annotation to the class header.

e.g

```
@Entity
Public abstract class Customer {
}
```

Entity fields and properties

Non-transient data in an entity class is persisted to a database table.

- The way the state is accessed by the provider is known as the access mode

There are two types of access mode

1- Field-based access mode.

2- Property based access mode.

1- Field-based Access - is provided by annotating fields. A persistent field in an entity class must be declared with private, protected, or package level access.

- Field based access provides additional flexibility bcoz fields or helper methods that should not be part of the persistent state can be excluded using the @Transient annotation or by omitting the getter and setter methods

2. Property-based Access - To provide property-based access getter and setter methods must be defined in a Java Entity class

- provides better encapsulation, as the access is only through methods.

- The getter and setter methods must be either Public or protected and must follow the Java beans naming conventions.

Entity States

An entity can exist in one of four states during its lifetime, These four states are:

1- New State - An entity instance created using java's new operator is in a new or transient state

- an entity instance does not have a persistent identity and is not yet associated with the persistence context

2- Managed state - An entity instance with a persistent identity and that is associated with a persistence context is in a managed or persistent state.

3 - Removed State — A managed entity instance can be removed from the database table when a table is committed, or when a remove method of an entity manager is called

4 - Detached state — An entity has a persistent entity identity but is not associated with the persistence context. This can happen when the entity is serialized or at the end of a transaction

EntityManager Interface and Key Methods

The javax.persistence.EntityManager interface is used to interact with the persistence context

- The entity instances and their life cycles are managed within a persistence context.

The key methods for EntityManager are:

1. persist() — persist an entity and makes it managed, the persist() method inserts a row in a database table. PersistenceException — if persist operation fails.

```
e.g
...
try {
  entityManager.persist(Customer);
```

2 - find() — searches an entity of a specific class by its primary key and returns a managed entity instance.

```
e.g
try {
    Customer = entityManager.find
(Customer.class, CustId);
```

3 - Contains() — takes an instance as an argument and checks whether the instance is in the persistence context.

```
e.g
...
  return entityManager.Contains(Cust-
mer);
 }
}
```

Merge() — updates the data in a table for an existing detached entity.

- The merge() method inserts a new row in a database table for an entity that is in a new or transient state.

```
eg
...
  entityManager.merge(customer);
```

5. remove() - deletes a managed entity. eg

```
...
entityManager.remove(Customer);
```

6. Clear() — Clears the persistence context. After this operation, all managed entities are in the detached state.

```
...
  try {
      entityManager.clear();
```

7. refresh() — refreshes the state of an entity instance from a database table. eg

```
...
    try{
        EntityManager.refresh(Cu-
stomer);
```

Important Tags of persistence.xml File

The persistence.xml file is a standard configuration file that contains the persistence units.

1) persistence-unit name is the name of the persistence unit.

2) transaction-type can be JTA or RESOURCE_LOCAL. Transaction type defines what type of transactions an application intends to perform.

3) jta-data-source is the name of the data source. Each persistence unit must have a database connection.

4) Additional standard or vendor-specific properties

- The hibernate.hbm2ddl.auto property with an update value updates the schema automatically.
- The hibernate.Dialect property specifies which database is used.
- The hibernate.show_sql property with a value as true enables logging of SQL statements to the console.

Bean validation — The bean validation API specification is provided to avoid code duplication and to simplify data validation.

- Bean validation is a model for validating data in java objects by using built-in and custom annotations that can apply predefined constraints. Bean validation is common to all layers of JavaEE and java web applications.

- Java — bean validation 1.1 API is JSR 349.
- JBoss Eap is fully compliant with JSR 349.

Bean Validation Constraints and Annotations

Validation constraints are the rules that are applied to validate data.

- These constraints are applied in the form of annotations for attributes, methods, properties, or constructors
- java provides built-in constraints, and supports custom constraints defined by the user. The javax.validation.constraints

@NotNull — The value in the field or property is not null eg

```
@NotNull
private String itemName;
```

@Null — The value in the field or property is null.

@Size — The size btw the min and max including the boundary values. eg

```
@Size(min = 3, max = 40)
private String name;
```

@Min

@Max

@Digits - The precision and scale of the field. The field must be a no. within the specified range.

@DecimalMin — verifies if the value in the field or property is a decimal value greater than or equal to the value defined in the DecimalMin.

@DecimalMax

@Future — verifies if the value in the field or property is date in the future

@Past — verifies if the field or property is a date in the past. eg

```
@Past
private Date startDate;
```

@Pattern – verifies if the value in the field or property matches the regexp.

@AssertFalse – verifies if the value in the field or property is false eg

@AssertFalse
private boolean isAvailable;

@AssertTrue

Automatic Vs Manual Invocation

Automatic Invocation – JavaEE7 app server provides hibernate validator package, which includes bean validation annotations as well as an automatic invocation of the validation constraints.

– hibernate automatically validates that the data matches the annotation constraints placed on the fields.

Manual Invocation – occasionally developers need to programmatically trigger bean validation. To programmatically validate an instance of an entity, use the javax.validation.Validator API

– The validator interface provides methods to validate an entire entity or a single property of an entity.

– JPQL supports the SELECT, UPDATE and DELETE statements

Creating Queries

JavaPersistence Query Language (JPQL) is a platform-independent query language defined as a part of the JPA spec to perform queries on entities in an object oriented manner.

– JPQL is similar to SQL in syntax

– JPQL queries are expressed in terms of java entities rather than database tables and columns.

– The EntityManager API supports methods for creating both static and dynamic queries

– CreateNamedQuery method is used to create static queries whereas the CreateQuery method is used to create dynamic queries.

Dynamic queries are created at runtime by an application using the following process.

1- Create a string containing a JPQL query.

2. pass the string to the entity manager's CreateQuery method and store the returned Query object.

3. Use the query's getResultList() method to execute the query and return the selected rows from the database.

* TypedQuery<?> class allows static typing of queries to avoid any issues with casting results eg

TypedQuery<Employee>

* JPQL also supports arithmetic functions in queries.

* The getSingleResult method returns and object array.
* The WHERE Clause is used to define Conditions on ē data that ē Query retrieves.
* <, =, >, <=, >=, <> are used to Compare ē arithmetic values.
* IN and NOT IN are used for all types
* LIKE and NOT LIKE are used for string values.
* BETWEEN and NOT BETWEEN are used for arithmetic, data, time and string values.
* MEMBER OF, NOT MEMBER OF, IS Empty, and IS NOT Empty are used for Collection types.
* _ (underscore) and % are used to build ē string expressions

Named parameters in Queries

* A named parameter is a query parameter serving as a placeholder for real values.
* A named parameter is bound to ē arguments by using the setParameter() method of javax.persistence.Query API e.g

```
query.setParameter("sal", salary);
```

* The positional parameters are query parameters in ē form of an index or ē ordinal position of a parameter in ē query.

The @NamedQuery annotation has four elements; name, query, hints and lockMode.

* A name is a required element of ē NamedQuery annotation. It defines ē name that is used by the EntityManager methods to refer to a query.
* The query is a required element of ē NamedQuery annotation and represents ē JPQL query string
* The hints represents query hints and properties. These hints can be vendor-specific, it is optional annotation.
* The LockMode represents the lock mode type to use in query execution. It is also optional element

CHAPTER FIVE (5)

Understanding Relationships btw Entities

- A foreign key is a column where the value of the data in ē column is a reference to ē ID or primary key of a row in another table.
* The foreign key is used to retrieve ē customer information as well as ē item information.
* developers use entity beans for one for each table.
* TO create a r/ship btw two entities class-level variables are used to represent an instance of one entity as an attribute of another entity

Annotation	Description
@OneToOne	defines an entity rlship as a single value, where one row in table X can be related to one or many rows in table Y
@ManyToOne	defines an entity rlship as multi-valued.
@ManyToMany	defines an entity rlship as multi valued
@JoinColumn	defines the column that JPA uses as the foreign key

Using a One-to-one Entity Relationship

Use the @OneToOne annotation when two entities relate to each other such that an instance of one entity only relates to a single instance of the other entity

* In a seperate table, then two entities such as User and UserSSN are related using a @OneToOne annotation
* Using a @OneToOne rlship, each User has a single SSN and each SSN has a single User.

Using One-to-Many Entity Rlship

Use the @OneToMany and ManyToOne annotations to map a JPA rlship anytime two entities relate to each other such that one instance of one entity relates to potentially multiple instances of the other entity.

* Using the @OneToMany and @ManyToOne annotations, implying that many Users belong to one group and one group can be assigned to many Users.
  - User entity
  - UserGroup entity.

Tuning the performance of Loading Relationship Data.

Nesting rlships or complex database schemas with lots of rlships can be especially harmful to performance.

* It is very easy to use JPA to retrieve more data than intended especially if every rlship is mapped bidirectionally on both entities

Using Lazy-Loading to improve JPA performance

To alleviate this problem and improve entity loading performance, JPA provides functionality called lazy loading or lazy fetching

* with lazy loading entities can map relationships, but only load those rlships when needed.
* The behaviour of whether or not JPA loads related entities is called the fetch type.

Two types of fetch
i- lazy ii- eager
i- Eager fetching is a default setting but can be set explicitly.

* To enable lazy loading set the FetchType value on the JPA rlship annotation.

Using the Join fetch clause in JPQL Queries to eagerly fetch Related entities

* The JOIN FETCH clause in JPQL queries overrides lazy-loading behaviour in entities and eagerly fetches the related entities.

* JOIN FETCH seperates management of the fetch behaviour for each query.
* To use a JOIN FETCH clause, include the name of the JPA-mapped collection that JPA should fetch eagerly.

Understanding Many-to-Many relationships

A many-to-many relationship occurs when each row in one table has multiple related rows in another table, and vice versa.

* To represent a many-to-many relationship in a relational database an intermediary table known as a cross-table or join-table must be used.
* A join-table is an intermediary table where each row in the table represents combinations of two IDs, one from each of the tables.

Using JPA Annotations to Map Many-to-Many Relationships

Use a @JoinTable annotation and two @JoinColumn annotations to map a many-to-many R/ship.
* The @JoinTable annotation is the bridge that allows JPA to use the join-table to populate the r/ships btw two objects, and two @JoinColumn annotations.
— are used to define how to join the join-table back to the entity table, as well as to the related entity table.
* The name of the join table to be used to join back to the entity class.
* accepts the instance of @JoinColumn
* JoinColumns — the column of the join-table to use to join back to the entity class
* InverseJoinColumns — This attribute accepts an instance of @JoinColumn, defined with a name attribute that maps to the column name on the join-table.

CHAPTER SIX (6)

Web services — expose standardized communication for interoperability btw application components over HTTP.
* Using a standard format for data transfer, such as JSON or XML, allows apps that consume web services to require only the ability to make an HTTP request to the service and to process the service's response.

Types of web services

1- JAX-RS Restful web services
2. JAX-WS web services.
* The above mentioned provides the same benefit of using web services such as loose coupling and standardized protocols

JAX-RS – is the Java API for creating lightweight RESTful web services.
* RESTEasy is ē implementation of JAX-RS.
* RESTful web services do not require use of a WSDL or anything similar to what is required when consuming JAX-WS services.

JAX-WS – is ē java API for XML-based web services using ē Simple Object Access protocol (SOAP)
* JAX-WS services also require clients and consumers to make more formal requests compared to JAX-RS which can make requests to individual endpoints simply over HTTP.
* 311 specification entitled Java API for RESTful web services 2.0 and provides additional features for efficient development of REST services.
* JBossWS is ē JSR-224 Java API for XML-based web services 2.2 spec compliant implementation for JAX-WS is RedHat JBoss EAP7

* Client-server architecture – is by implementing a web service layer, developers can abstract ē front-end layer and create an application comprised of many loosely coupled components.
* RESTful web apps services are stateless.

* JEE version of ē TO DO List app utilizes a RESTful web service to abstract ē angular front-end from ē business logic and data layer.
* JavaEE 7 supports JAX-RS 2.0, which makes developing RESTful web services and adhering to their standards very simple!

Creating RESTful web services

* JAX-RS RESTful web service consists of one or more classes utilizing ē JAX-RS annotation to create a web service.
1st step – Create a class that extends the class:
javax.ws.rs.core.Application
– The new subclass is used to also define the base URL for ē web service with @ApplicationPath annotation.
* The @ApplicationPath annotation sets ē base URL for ē web services. e.g
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
@ApplicationPath("/api")
Public class Service extend Application {
// can be left empty
}

Annotation	Description
@Path	defines ē base URL for either ē entire root class or for an individual method
@Consumes	defines ē type of ē request content that is accepted by ē service class or method
@Produces	defines ē type of ē response content that is returned by ē service class or method
@GET	is applied to a method to create an endpoint for ē HTTP post request type. and is used to retrieve data

@POST	is commonly used to save or create data.
@DELETE	Commonly used to delete data
@PUT	Used to update existing data.
@PathParam	is used to retrieve a parameter passed in the URL.
@QueryParam	is used to retrieve a parameter passed in the URL as query parameter.

Customizing Requests and Responses

* @Produces defines the MIME media type for the response returned by the service.
* @Consumes defines the MIME media type for the request required by the service.
* The @Produces or @Consumes annotations can be modified to enforce XML as the response and request type

Injecting Parameters from the URL

To use a path parameter on a JAX-RS method, annotate it with the @PathParam annotation. This annotation is typically used when the client is requesting a specific resource, such as requesting a user's data.

- @QueryParam is used in searches and filtering users by their email preferences.

Consuming REST Services

You can use a simple Curl command to reach a REST endpoint or use a browser plugin to test a REST service.

There are three distinct components of JAX-RS client

i- Client - The client creates instances of web Targets.

ii- WebTarget - The URL destination of a REST endpoint

iii- Response - The object containing the response from the service

* The first step in creating a REST client is to initialize a client object using ClientBuilder

⇒ Client client = ClientBuilder.newClient();

⇒ Client.close(); - to close the client.

Configuring the Target

To do this, developer can append additional elements to extend the webTarget's path eg

```
WebTarget webTarget = client.target("http://localhost:8080/todo/api");
WebTarget items = webTarget.path("items");
```

Creating the Request

The request() method on the WebTarget class enables developers to define the type of HTTP request to make the REST endpoint.

⇒ Response response = webTarget.request().get();

Methods that are available for HTTP request

i - get() ii - post() iii - put() iv - delete.

Parsing the Response

After making a request with the WebTarget, the target returns a Response object. This Response object needs to be mapped to the expected entity or object type.

Authentication with REST Clients

REST APIs are secured with some kind of security, such as Basic or Digest Authentication

- Basic authentication is often the simplest and most form of restricting access to REST APIs.
- This requires developers to provide a credentials encrypted with a Base 64 encoder when making requests to the REST API.

HTTP Status Code