**Content Page**

|  |  |  |
| --- | --- | --- |
| **SI No** | **Course Title** | **Page No** |
| 1 | Full Stack Development: Introduction |  |
| 2 | Workflows and Projects using Atlassian JIRA |  |
| 3 | Creating and Running an Agile Project in JIRA |  |
| 4 | Git Essentials: Become a Git and GitHub Ninja |  |
| 5 | Essentials of Cloud Computing |  |
| 6 | Spring Essentials |  |
| 7 | Hibernate Framework |  |
| 8 | Introduction Automation Testing |  |
| 9 | Deploying and Running Docker Containers |  |
| 10 | Docker Skills: Advanced Docker Orchestration |  |

**FULL STACK DEVELOPMENT: INTRODUCTION**

**Full-Stack Development**

* Full-stack development refers to building end-to-end software solutions including frontend and back-end development.
* Front-end refers to the front part of the application or UI which is seen by the end-user and used for end-user interaction.
* The back-end part is related to server-side development, database, and API development.

**What is a full-stack web developer?**

* Full stack web developers have the ability to design complete web applications and websites. They work on the frontend, backend, database and debugging of web applications or websites.
* They are proficient (Skilled or expert) in both frontend and backend languages and frameworks, as well as in server, network and hosting environments.

**History of Web Development:**

* Networks
* HTML
* Browsers
* Browsers++
* Server side
* Server side languages
* Multimedia
* The established players

**Modern Web Development:**

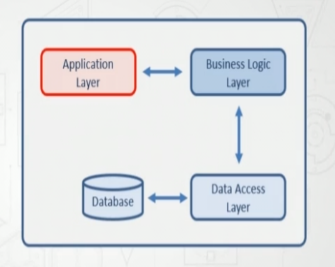
* Mobile
* Open source
* Standards
* Cloud
* Fiber optics
* Social media
* Finance/commerce

**Full Stack Developer Role:**

* Startups
* Cost savings
* Wholistic view
* Communication
* Flexibility
* Managers

**The First Layer:**

* The client side
* HTML page
* CSS
* JavaScript
* Multimedia

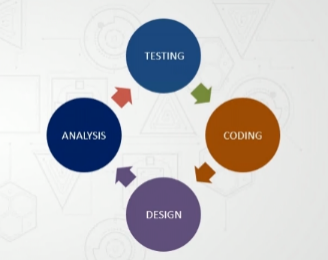
**The Second Layer:**

* Business logic
* Server
* Communication
* Development environment

**The Third Layer:**

* Form data
* Other data
* Persistence
* SQL
* Stored procedures
* Database functions
* Transactions

**Test, QA, and Deploy:**

* Build tools
* Testing the application
* Automated testing tools
* Quality assurance
* Deploy/workflow
* Hosting/DNS

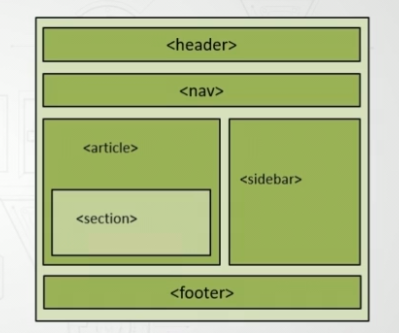
**Maintenance:**

* Changes and updates
* Git and GitHub
* Agile

**Services:**

* Payment systems
* Web services
* SEO

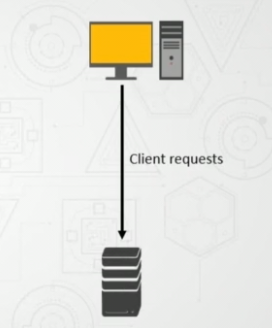
**The Basic Client Side:**

* HTML
* CSS
* CSS frameworks like Bootstrap
* JavaScript
* JS frameworks like jQuery
* Browser based tools
* Chrome developer tools
* Firebug

**Front End CSS Frameworks:**

* Semantic UI
* Foundation
* Bootstrap
* Materialize
* Pure
* Skeleton
* jQuery
* React
* Angular JS

**Web Server Technologies:**

* Definition of a web server
* Functions of a web server
* Logs, virtual directories, and port allocation
* Internet Information Server (IIS)
* IIS Express
* Apache
* Modules
  + SSL
  + IPv6
  + FTP
  + Perl, Lua, and PHP
  + Bandwidth throttling
  + Load balancing

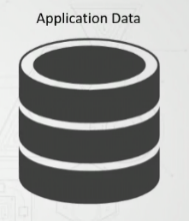
**The Third Layer:**

* The persistence layer
* Data does not change
* Data changes
* State
* Convenience

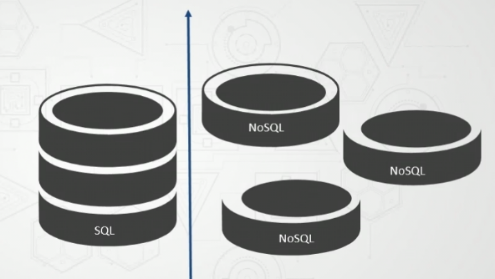
**Local Storage:**

* Session
* Cookies
* HTML 5 local storage

**Organized Data:**

* Types of organized data and database
* Relational database
* Operational database
* Data warehouses
* Distributed databases

**Web Application Databases:**

* SQL vs NoSQL
* SQL
  + MySQL
  + Oracle
  + SQL Server
* NoSQL
  + Couch base
  + MongoDB
  + Mark Logic

**MVC Components:**

* The process
* Separation of concerns
* Faster development
* Maintainable
* Flexible presentation layer
* Testable

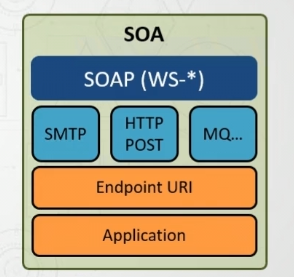
**Representational State Transfer:**

* HTTP, HTTP2, and URLs
* Data
* REST

**REST Verbs:**

* HTTP GET
* HTTP POST
* HTTP PUT
* HTTP DELETE
* HTTP PATCH

**Service Oriented Architecture:**

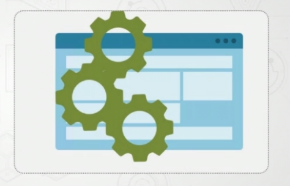
* Service Oriented Architecture (SOA) became the

norm around the mid-2000s

* Components provide services to other components
* SOA resembles micro services
* Each service has specific responsibility
* Services can be accessed remotely
* Communication between services
* A service is self-contained

**Micro Services:**

* Application with very small footprint
* Runs in its own process
* Provides singular functionality

**Testing:**

* HTML
* CSS
* JavaScript
* Cross browser
* Automated testing

**Reasons for Build Tools:**

* Unit Testing
* Web page refreshing
* Dependencies
* New file creation
* Changes

**GIT ESSENTIALS: BECOME A GIT AND GITHUB NINJA**

**Git:** Version control system is a tolls that helps to track changes in code.

It is Popular, free and open source, fast and scalable.

Why Do We Need a Version Control System?

Version control systems allow multiple developers, designers, and team members to work together on the same project. It helps them work smarter and faster! A version control system is critical to ensure everyone has access to the latest code and modifications are tracked. As development becomes increasing complex and teams grow, there’s a bigger need to manage multiple versions and components of entire products.

**GitHub:** Website that allows developers to store and manage their code using Git.

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018

**GitHub Account:**

* Create a new Repository.
* Make our first commit.

**Setting up Git:**

* Visual Studio Code
* Windows (Git Bash)
* Terminal

**Configuring Git:**

* git config --global user.name “my name”
* git config --global user.e-mail “my e-mail”
* git config --list

**Fundamentals of Git:**

Here is a basic overview of how Git works:

1. Create a “repository” (project) with a git hosting tool (like Bitbucket)
2. Copy (or clone) the repository to your local machine
3. Add a file to your local repo and “commit” (save) the changes
4. “Push” your changes to your main branch
5. Make a change to your file with a git hosting tool and commit
6. “Pull” the changes to your local machine
7. Create a “branch” (version), make a change, commit the change
8. Open a “pull request” (propose changes to the main branch)
9. “Merge” your branch to the main branch

**Git Client installation and setup:**

**Step 01:** Open your local web browser, chrome or Microsoft Edge. and search the below link. [*https://git-scm.com*](https://git-scm.com)

**Step 02:** Now, download for windows and select standalone windows setup of Git. (*64 bit*)

**Step 03:** Finally, Installed the Git.

**Step 04:** Then you will check the version. (*git --version*)

**Create a Repository:**

**Step 01:** Open browser, search for GitHub Login. and Sign in with your username and password.

**Step 02:** Your GitHub dashboard will be displayed on the screen.

**Step 03:** Create a new repository and In the upper-right corner, use the drop-down menu, and select Newrepository.

**Step 04:** Give a name for your repository. For example, "fullstackdevelopment".

**Step 05:** Add a description of your repository. For example, "Mini Project I" and Click Create repository.

**Git Operations:**

**Clone and Status:**

* **Cloning** a repository on our local machine. *git clone <-some link->*
* **Status** displays the state of the code. *git status*
* **Untracked** new files that git doesn’t yet track.
* **Modified** Changed
* **Staged** file is ready to be committed.
* **Unmodified** Unchanged

**Example:**

Git clone: *git clone* [*https://github.com/divyatalawar/fullstackdevelopment.git*](https://github.com/divyatalawar/fullstackdevelopment.git)

Git Status: *git status*

**Add and Commit:**

* **Add** adds new or changed files in your working directory to the Git staging area. *Git add <-file name->*
* **Commit** It is the record of change. *git commit –m “some message”*

**Example:** git add .

git status

git commit –m “Add new paragraph”

**Push command:**

Push upload local repo content to remote repo git push origin main.

**Init Command:**

Init –used to create a new git repo.

* Git init [Initialized Empty]
* Git remote add origin <-link->
* Git remote –v [to verify remote]
* Git branch [to check branch]
* Git branch –m main [to rename branch]
* Git push –u origin main

**Branch Commands:**

* Git branch [to check branch]
* Git branch –m main [to remote branch]
* Git checkout <-branch name-> [to navigate]
* Git checkout –b <-new branch name-> [to create new branch]
* Git branch –d <-branch name-> [to delete branch]

**Example:** git branch

git checkout –b feature

git push origin feature

**Merging Code:**

**Way 1**

Git diff <-branch name-> [to compare commits, branches, files and more]

Git merge <-branch name-> [to merge 2 branches]

**Way 2**

Create a PR [Pull Request]

**Example:** git diff main

git merge main

git pull origin main

**Pull Request:**

It lets you tell others about changes you have pushed to a branch in a repository on GitHub.

**Pull Command:**

Used to fetch and download content from a remote repo and immediately update the local repo to match the content. *git pull origin main*

**Resolving Merge Conflicts:**

An event that takes place when git is unable to automatically resolve differences in code between two commits.

**Undoing Changes:**

**Case 01:** Staged changes.

git reset <-file name->

git reset

**Case 02:** Commited changes (for one commit)

git reset HEAD ~ 1

**Case 03:** Commited changes (for many commits)

git reset <-commit hash->

git reset –hard <-commit hash->

**Fork:**

A fork is a new repository that shares code and visibility settings with the original “upstream” repository. Fork is a rough copy.

**HIBERNATE FRAMEWORK:**

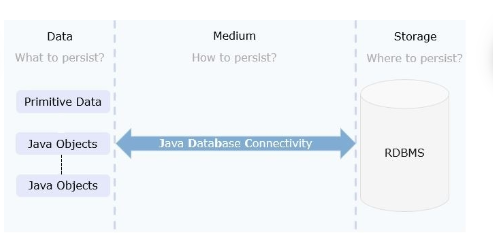
Hibernate Framework is an open source Java based framework. Hibernate framework shields developers from ‘Messy’ SQL and allows the developer to concentrate and work with the object model. It is a tool for object relational mapping. The technique for mapping a model in an object oriented domain to a relational type of database is ORM, Object Relational Mapping. Hibernate is free software.

**Observations on the movie ticket booking scenario:**

* The booking details for the movie was stored in some repository.
* Both Annie and Scott opened one application from their desktop and could check the number of available tickets.
* When Annie booked the tickets the number of available tickets got updated permanently in the repository.
* When Scott checked again he could see the latest details of availability.

**JDBC or ‘Java Database Connectivity’ is a Java Core API for performing database interaction:**

* Using JDBC API, a Java application can access a variety of databases such as MySQL, Oracle, etc.
* JDBC follows a relational database oriented approach to work with the data using SQL queries.



**JDBC API:**

The problem with Serialization is solved by JDBC, but it does not store the Java objects directly. The data from the objects need to be converted into a SQL quey and then executed, for persistence.

* SQL code has to be embedded within Java Programs which makes it non-portable.
* JDBC API allows the developer to fire the SQL queries from the Java code. This means the developer needs to know the specific SQL constructs for the Relational Database Management System (RDBMS) used.
* Also, it is the responsibility of the programmer to make sure that the data model and the object model are synchronized properly.

**Object Relational Impedance:**

JDBC, I/O, Serialization do not solve the problem of data persistence effectively. For a medium to be effective, it needs to take care of the fundamental difference in the way Object Oriented Programs(OOP) and RDBMS deals with the data.

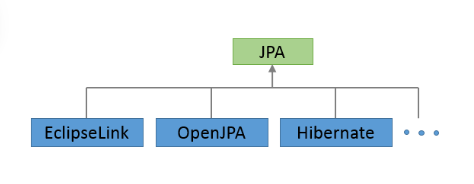
* In Programming languages like Java, the related information or the data will be persisted in the form of hierarchical and interrelated objects.
* In the relational database, the data is persisted as table format or relations.

**Object Relational Mapping (ORM):**

We use ORM mainly for the following reasons:

* ORM resolves the Object Relational Paradigm mismatch.
* The lower level interaction with the database is handled by the ORM. Framing and executing the database dependent queries is taken care of by the ORM framework.
* ORM helps the developer to get rid of “messy SQL”. The developer need not waste time in writing the plumbing code.
* ORM allows the developer to concentrate on the business logic and work with the object model.
* ORM is database independent. All database vendors give the necessary support for ORM. Hence the application code becomes portable without worrying about the underlying database.

In Object Relational Mapping, the Java entity classes are mapped to relational database tables. In this technique the entity classes are mapped to the database tables, the data members are mapped to the database table columns, objects of Java entity classes are mapped to the records of the database tables.



**Consider a Java Entity class “Account” with a mapping database table “Account” as below:**

In the provided example:

* Java Entity class Account is mapped to a table Account
* Data members id and balance of Account class are mapped to the table columns ID and Balance
* An instance of Account class with values 1001(id) and 50000(balance) is a record (row) in a table.

**History of Hibernate Framework:**

* Hibernate framework was started by Gavin King and his colleagues in the year 2001 with the aim of offering better persistence support than those offered by the JEE component Enterprise Java Bean 2. It tried to simplify the complexities and supplement certain missing features of EJB2.
* JBoss later took the lead developers of Hibernate for the further development of this framework.
* The subsequent versions provided various new features and enhancements for better performance.
* The current version is 5.0.2, released Sep 2015.

**Features of Hibernate:**

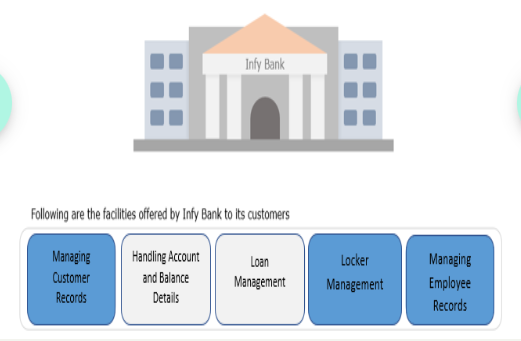
* Object Relational Mapping: Hibernate, being an ORM framework aims to resolve the Object Relational Impedance Mismatches proving itself as an effective data persistence medium.
* Scalability and Reliability: Hibernate works well in the client server based environment and drivers a scalable architecture. Hibernate provides good stability and quality, hence it is reliable.
* Extensible: Hibernate is highly configurable and extensible.
* High Performance: Hibernate has high performance due to various features like multiple fetch strategies, optimistic locking with automatic versioning and time stamping, caching, etc.
* Idiomatic Persistence: Hibernate enables the development of persistent classes that follows object oriented idioms like inheritance, composition, association, and Java collections framework.

**Benefits of Hibernate:**

* Lightweight: Hibernate implements ORM using simple POJO classes.
* Open Source: Hibernate is freely available and may be redistributed and modified
* Vendor Independent: Hibernate, or in general JPA, prevents writing code according to the database vendor. Hence t is vendor-independent.
* Non-Invasive: Hibernate does not force the developer to extend or implement any class or interface.

**Infy Bank Application:**

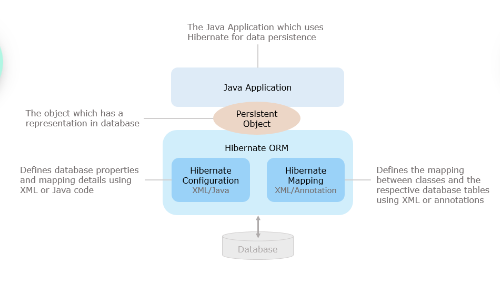
Let us understand more about Hibernate and its concepts using an example. This example would be used throughout this course for a deeper understanding of the various features offered by Hibernate.



The management of Infy Bank wants to build a Java application using Hibernate Framework.

In this course, we will develop a few examples for Managing Customer Records, Managing Employee Records, and Locker Management using use cases. Other use cases will be discussed in the Hibernate Framework Advanced course.

**Hibernate Application Architecture:**

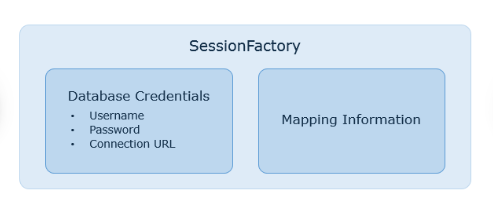
****

**Hibernate APIs:**

* org.hibernate.Session Factory
* org.hibernate.Session
* org.hibernate.Transaction
* org.hibernate.Transaction Factory

**Session Factory API :**

* Session Factory API is available in org.hibernate package, is used to create a number of database connection requests.
* Using the Session Factory object, Hibernate Session objects are created.
* Ideally, there should be one Session Factory instance defined for each database.



**Transaction API:**

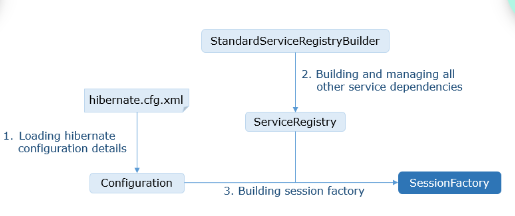
* A Transaction is a single-threaded, transient object to specify one single unit of work.
* Transaction API is available in org.hibernate package.
* A session might make several transaction instances whenever required.
* It creates an abstraction of the application from the JDBC, JTA, or CORBA transaction.
* Session API requires Transaction API for implementing CRUD (Create, Read, Update, Delete) operations.

**Service Registry API:**

* The main purpose of a service registry is to hold, manage, and provide access to services.
* This is the central service API.
* In Hibernate this API helps in managing and providing access to standard services like Dialect Factory and Connection Provider.
* This is available in the org. hibernate. Service package.

**Let us discuss the basic steps for an application in Java to connect to the database using Hibernate.**

* Create an instance of Configuration class and loads hibernate configurations and related mappings.
* Use org.hibernate.service.StandardServiceRegistryBuilder API and create an instance of ServiceRegistry.
* Use the instance of ServiceRegistry and create the instance of SessionFactory.



**Operations n Hibernate:**

* CREATE is used to create a record in the database table.
* READ is used to read the data from the database table.
* UPDATE is used for updating a database table record.
* DELETE is used for deleting a databases table record.

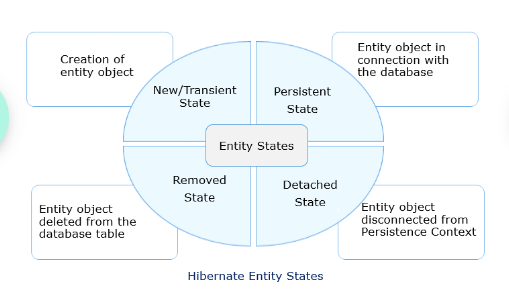
The Session API provides methods for performing the CRUD operation.

* To CREATE/UPDATE the data, the methods available are:
  + Save()
  + Persist()
  + Update()
  + Save or update()
* To READ the data, the methods available are:
  + Get()
  + Load()
  + Refresh()
* To DELETE the data, the methods available is:
  + Delete()

**Hibernate Entity States:**

We have discussed CRUD operations till now. The session API methods provide various methods for performing CRUD operations on entities. There are some intricate differences between the methods. However, to decide the usage of appropriate session methods(eg: get() or load()) requires the understanding of Persistence Context and the Hibernate Entity States.

The entity state is defined with respect to its relation with the Session instance, or in other words, its proximity with the Persistence Context. As shown below an entity is in any one of the four different states: transient, persistent, detached, or removed.



**Following operations brings an entity instance to a Persistent state:**

* By persisting a new record into the database using persist () or save () method.
* By fetching an entity record from the database using get () or load () method.
* By updating the entity record in the database using save Or Update () , update() or merge() method(explained later).

**Detached State:**

An entity is in the detached state when it is disconnected from the persistence context but the corresponding record is still available in the database.

An entity instance can be in the detached state for the following reasons:

* If an entity is detached specifically from the persistence context using evict(entity) method.
* If the entire persistence context is cleared using clear() method.
* If the persistence context is closed using close() method.

**Hibernate Query Options:**

* Retrieval by Identifier
* Hibernate Native SQL Query
* Hibernate Query Language
* Criteria API

**For implementing the requirement of providing Locker Facility to Infy Bank's existing customers we will be using one to one association mapping feature of Hibernate.**

Infy Bank is introducing Locker Facility to its customers. A customer can open a locker and locker cannot be shared by more than one customer.



To implement this scenario we are using two tables CUSTOMER and LOCKER. The relationship between CUSTOMER and LOCKER has been defined as one to one mapping in the database by imposing a unique constraint on the foreign key column.

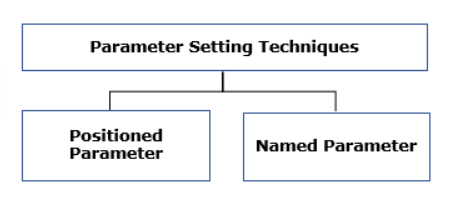
The CUSTOMER and LOCKER tables are shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CUSTOMER** | | | | | |
| **CUSTOMERID** | **CUSTOMERNAME** | **DATEOFBIRTH** | **ADDRESS** | **PHONENO** | **LOCKERID** |
| 7001 | William | 17-05-1995 | Kolkata | 9067767872 | LOC101 |

|  |  |  |
| --- | --- | --- |
| **LOCKER** | | |
| **LOCKERID** | **LOCKERTYPE** | **RENT** |
| LOC101 | Small | 1100 |

**HQL Support for Security:**

Hibernate provides a solution to SQL injection attacks by allowing the user to construct the queries dynamically at runtime using parameters binding technique. If required, the compiled query can be reused with different parameters, which results in increased performance. Hibernate supports two ways for parameter binding or parameter setting.



* Positioned Parameter: It uses a question mark (?) to define a named parameter in the query, and according to the position sequence, the value is set to the parameter.
* Named Parameter: It uses a colon, for example: id to define a named parameter in the query.

**HQL functions which help in manipulation of Strings are:**

| **FUNCTIONS** | **DEFINITION** |
| --- | --- |
| length(string Value) | Computes the number of characters |
| upper(string Value) | Converts the letters of string to uppercase |
| lower(string Value) | Converts the letters of string to lowercase |
| concat(stringVal1,stringVal2) | Concatenates two given strings |
| substring(stringVal1,position) | Breaks the string into two substrings from the given position to the last index |
| substring(stringVal1,pos1,pos2) | Breaks the string into two substrings from pos1 to pos2 |

**Below are the steps required for mapping one to one relationship:**

1. From the given entities find out the source and the target entity

The foreign key column (LOCKERID) of the CUSTOMER table references the primary key column of the LOCKER table, so the Customer entity can be the source and the Locker entity can be the target. To map the foreign key column, we need to associate source entity with the target entity i.e. the Customer class has a reference of Locker.

2. Give appropriate annotations in the Customer entity to map with Locker entity

The annotations used in Customer class (source) for one to one mapping with Locker (target) are:

* @OneToOne(cascade = CascadeType.ALL)

This annotation is applied to the reference attribute of Locker in the Customer entity which indicates that the relationship has one to one cardinality. The cascade attribute of the annotation is mandatory. This attribute transfers operation (such as insert, update, delete) done on the Customer object to its Locker object.

* @JoinColumn(name = "lockerId", unique = true)

The annotation has two attributes, name attribute specifies the name of the foreign key column in the Customer table and the unique attribute assures unique values in the foreign key column to achieve one to one mapping.

**HQL Aggregate Functions:**

Hibernate Query Language like SQL, allows all the standard aggregate functions supported by SQL. In HQL, aggregate functions can be used on the entity instances or persistent fields thereby taking care of ORM aspects. Aggregate functions help to compute results using the attribute values available in the database.

**Aggregate functions supported by Hibernate:**

* Avg()
* Max()
* Min()
* Sum()
* Count()
* Conut(\*)

**Best Practices in Hibernate:**

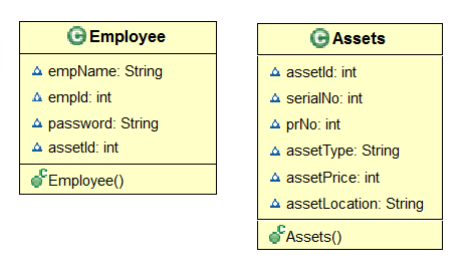
* Hibernate should not update database schemas.
* The queries used should not be vulnerable to injection attacks.
* Avoid database tables associated with more than one entity.
* Persistent classes should implement hashcode() and equals().
* The equals() and hashcode() of the persistence class must access its field through getter methods.

**Hibernate Basics Capstone:**

**Problem Description:**

**Note:** Since mapping is not provided among classes, assetId has been used in the Employee class to show the relationship between employee and associated asset details. We need to use the same assetId to provide the relation between employee and asset details.

**Refer to the below class diagram:**

****

* Create the below entities as part of the Asset Management System
* Assets class with an appropriate mapping to table Assets\_Table
* Employee class with an appropriate mapping to table Employee\_Table

Implement CRUD Operations:

* Create:

Create a new Employee object and add the details into the database as empId as the id field

Create a new Assets object and add the Asset details into the database with assetId as the id of the newly persisted record

* Read:

Retrieve the Assets details using get() or load() for the appropriate id given

Retrieve the Assets details with maximum price.

Count the total no. of employees from the Employee table.

* Update:

For the given empId, update the password with the new value.

* Delete:

Delete the Employee details of the given empId and the associated Asset details from the Asset table

**AUTOMATION TESTING:**

**Scenario 01**

You have a functional test case to verify the search functionality of an Employee Directory System by entering an employee number.

To execute this test case manually, you will have to execute the following steps:

* Open Employee Directory System web app
* Enter the employee number to be searched
* Check whether the details of the corresponding employee is displayed
* Update the test case result

To execute the above test case, you might take approximately one minute.

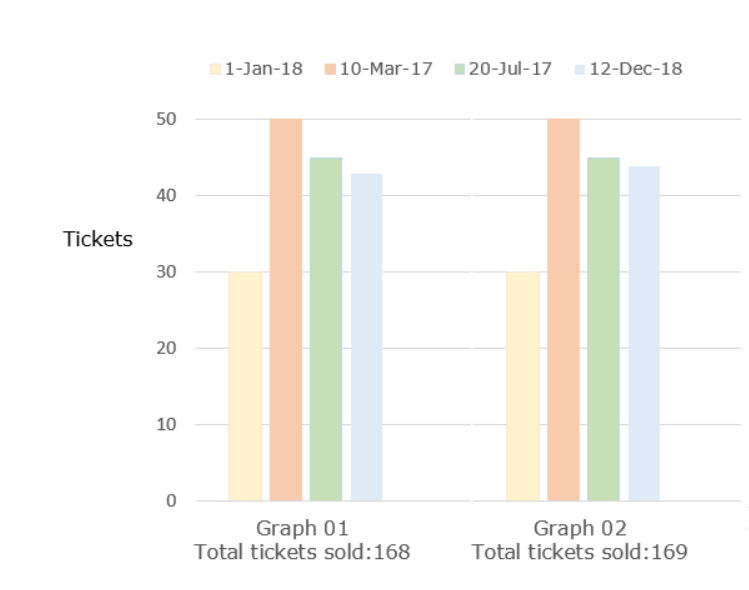
Now if you are asked to test the same functionality with 100 different employee numbers, you will have to repeat the same steps with different test data a 100 times and the process would take at least a 100 minutes.

But if you use an automation testing tool and write a script to execute all four test steps, all you have to do is start the script execution. Everything else, from launching the application to logging the test results, will be taken care of by the automation testing tool for all 100 employee numbers. The bonus is that the overall testing process would only take around ten minutes!

**Scenario 02**

A flight reservation application produces graphs which displays the daily ticket sales. The graph is be updated after each successful ticket booking. You have a functional test case to verify whether the graph is updating for every successful ticket booking.

Given below are two graphs, the first was generated before inserting a ticket and the other after inserting a ticket, for the date 12-Dec-2018.



**Scenario 03**

'Loyal Bank' has an online banking application which allows its customers to perform banking transactions in an easy manner. Its fund transfer module (refer below image) has a performance requirement which states:

'If 1000 users are transferring funds simultaneously, the transaction must be successful and a success message should be displayed on the page within 3 seconds, for all 1000 users.'

**Test automation:**

* Process consists of converting manual tests into automated ones with the help of appropriate testing tools
* Hence serves better if implemented on an existing manual process
* Scripts are designed, coded and unit tested before the actual test execution
* Is mainly used during regression testing phase, where the same test cases are repeatedly executed

Listed below are some of the key activities which are mandatory for achieving test automation.

1. Automation feasibility analysis
2. Identifying the right automation tool(s)
3. Test data with expected results
4. Setting up the test environment

**Automation Testing Life Cycle:**



**There are mainly fours steps involved in the process of requirement gathering and automation feasibility analysis:**

* Understand the project and technical architecture of the AUT
* Baseline the requirements
* Automation feasibility analysis
* Zeroing down on the appropriate test automation tool(s)

**There are mainly fours steps involved in the process of strategy and design:**

* Perform ROI analysis based on estimated values
* Prepare the strategy document
* Identify the most suitable test automation framework
* Setting up test environment

**There are mainly fours steps involved in the process of scripting and preparing for execution:**

* Create test scripts and organize them into a library
* Associate the test scripts with test data
* Integrate the test with framework and perform a dry run
* Prepare the execution plan

**There are mainly fours steps involved in the process of execution and result analysis:**

* Execute the scripts and verify the results
* Log the defects for failed test cases
* Rerun the scripts after fixing the defects
* Share the results with necessary stakeholders

**To summarize, the key advantages of automation testing are:**

* It is very helpful if a set of tests have to be executed repeatedly for a large number of users
* Automation facilitates to execute regression tests in mainstream scenarios on time. For example, execution can be set to happen overnight without any manual intervention and you will get the results the next morning
* It helps a lot in cross browser testing as the same test script with very little modification can be run on multiple browsers
* Unlike manual testing where the tests are run sequentially, automated tests can be run at the same time on different machines

**Some of the common disadvantages of automation testing are:**

* The initial cost of creating test scripts and configuring automation framework is more than the cost of executing the tests manually
* Automating visual references is not possible. For example, a font or a colour cannot be input via code or automation tools and thus it becomes more or less a manual testing process
* Licensed automation tools are expensive even though they provide good support in test automation. Since companies are investing a lot of money on the tool licenses, they have to use the tool where ever possible to get the maximum utilization.
* Proper training on automation tools and knowledge of various scripting languages are a must for working with automation testing tools.

**The main challenges faced by companies trying to adopt test automation are:**

* Non-availability of a test tool which is a one-point solution for testing of applications developed in different technologies
* Huge upfront investment in automation tools and training and continous maintenance costs
* Tool vendors downplay the limitations of test automation
* Inability of tool vendors to provide appropriate solutions/support for third party controls in the applications
* Non availability of personnel who are experienced in test automation concepts
* Non-availability of personnel who are experienced in testing as well as programming skills

**Misconceptions about test automation:**

* Test automation is simply “record and playback”
* No, its not. You are supposed to do lot of enhancement in your test script to create the perfect script
* Test automation is the “silver bullet” to solve all testing problems
* No, its not. There are many limitations for test automation and it cant reveal you all the bugs.
* ROI will be achieved immediately
* Never. For getting ROI you have to utilize the tool very efficiently and lot of cycles of iterations are needed to achieve ROI.
* Manual testing would be replaced by automation testing
* Never. Manual testing is the base of testing and it have to be done on all the applications to make sure the AUT is stable. Automation testing can be only implemented in stable products.

**Case study:**

Below given is a case study of one client who achieved significant reduction in the quality control cost and testing efforts with the implementation of test automation.

The Client is a leading provider of Insurance in United States. Their Services include Group Life, Dental, Disability and Individual Disability Income Insurances. The project involves various testing of the following applications.

Life Track

* Web based Applications used by Group Life/CI/LTC Policy Administration, Claim generation and processing and Recordkeeping.
* PowerBuilder based application for SOH and Recordkeeping

Dental Track

* Mainframe Application for Dental claims filing, processing and verification

Disability Track

* Web based Applications used by Disability users for claim adjudication
* Individual Disability Income Policy Administration
* PowerBuilder based application for Claim generation and processing
* Web Services based Testing for Web based Absence Management application.

After one year of successful client engagement, the need to cut down on QC Costs and improve testing quality and productivity was sensed as an opportunity to build customer’s trust.

Test automation in the below areas were conceptualized to decrease the effort and increase the efficiency:

* Automation of regression test suite using the functional testing tool QTP 9.5
* Managing the testing activities using HP QC standardization
* Use of tools like macros
* Expand the testing process like web service testing into automation

The below graphs display the quality control cost and productivity improvement that were achieved after implementing the strategies conceptualized.

Also, after test automation, the company recorded:

* 100% defect removal efficiency
* 34% revenue growth in the project
* 26% increase in application coverage
* 14% decrease in effort

**DOCKER SKILLS: ADVANCED DOCKER ORCHESTRATION**

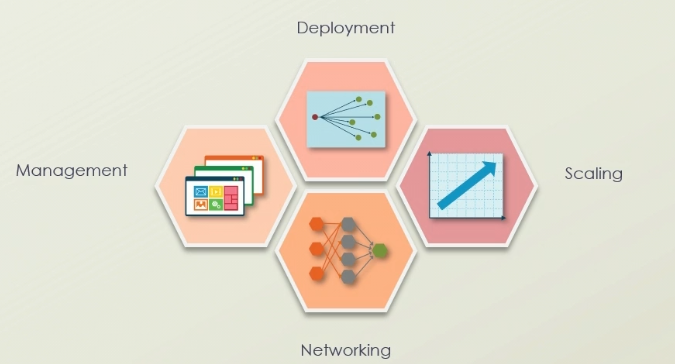
Docker is a platform that is designed to make it simple to create, update, deploy, and run containerized applications.

This competency area includes building images, understanding of basic DevOps commands for Docker, Tags, and Labels, and running multiple services in a container, among others.

**Key Competencies:**

1. Building Images - Building Docker images using the Image API, including building from git, tar balls, text files, or URLs.
2. Basic DevOps - Understanding of basic DevOps commands for Docker including versioning, top, history, and ability to view and understand system events that are provided by the Docker API.
3. Tags and Labels - Creating, updating, listing, and sorting with tagged images.
4. Running Multiple Services in a container - Testing a user's knowledge of running multiple services in a container.
5. Orchestration - Running dockerized applications at scale using Docker Swarm or Kubernetes.

**Container Orchestration:**

****

**Containerized Orchestration Tool Benefits:**

* Portability
* Deployments
* Quality
* Security

**Docker Orchestration:**

* Swarm mode
* Nodes
* Tasks and Services
* Load balancing

**Kubernetes Orchestration:**

* Automated deployment
* Connect replication
* Scale-in and Scale-out
* Load balancing
* Rescheduling
* Network port management

**Docker Orchestration Tools:**

* Docker swarm
* Kubernetes
* Open Shift

**Docker Swarm Components:**

* Nodes
* Tasks
* Services
* Load balancing

**Open Shift Components:**

* Red Hat Enterprise Linux
* Enterprise Kubernetes
* Red Hat Marketplace
* Virtualization

**Cloud-based Orchestration Services:**

* Google Container
* Google Cloud Run
* AWS Elastic Kubernetes
* Amazon EC2 Container Service (ECS)
* Azure Container Instances
* Digital Ocean Kubernetes Service
* Red Hat Open Shift Online

**Orchestration and Cluster Management Tools:**

* Containerization
* Orchestrators
* Clustering
* Container interchangeability

**Multiple Docker Containers:**

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

**Docker containers that run on Docker Engine:**

**Standard:** Docker created the industry standard for containers, so they could be portable anywhere

**Lightweight:** Containers share the machine’s OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs

**Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

****

**Use Cases of Multiple Docker Containers:**

* Migrate existing applications
* Continuous integration and deployment (CI/CD) support
* Automate deployment of repetitive jobs and tasks
* Creating container native application
* Refactoring existing applications
* Providing micro service architecture support

**Docker Network Drivers:**

* Bridge
* Host
* Overlay
* Macvlan

**Multiple Docker Hosts:**

* Overlay Networks
* Swarm services

**Overlay Network Security:**

* AES algorithm
* OPT encrypted
* IPSEC tunnels

**Overlay Network Common Scenarios:**

* Default overlay network
* Standalone container overlay network
* User defined overlay network
* Container and swarm service overlay network

**Docker Swarm:**

Docker is a type of tool which is used to automate the process of application deployment as the lightweight container so that the particular application can work seamlessly in the different types of environments.

In simple words, the Docker a lightweight package of the software that has all the dependencies, libraries and frameworks that are required to run an application.

* Physical virtual machines
* Docker cluster
* Swarm manager
* Nodes

**Swarm Mode Features:**

* Task
* Service
* Global Service
* Replicated Service

**Swarm Mode Nodes:**

* Manager
* Worker

**Manager Nodes:**

* Cluster state
* Swarm recovery
* High availability requirements

**Swarm Node Availability Status:**

* Active
* Pause
* Drain

**Swarm Node Manager Status:**

* Leader
* Reachable
* Unavailable

**Docker Network Configuration:**

* Bridge
* Host
* Overlay
* IPvlan
* Macvlan

**Docker Bridge Network:**

* Link layer device
* Software bridge
* Docker daemon host
* User-defined bridge network

**Docker Host Network:**

* Performance optimization
* Linux hosts

**Cloud Orchestration:**

* Container Management
* Resource Management
* Service Management

**Cloud Orchestration Delivery Models:**

* Single cloud
* Multi cloud

**Containers as a Service:**

* Standard definition
* Infrastructure agnostic
* Portability
* Hybrid service structure
* Infrastructure as a Service
* Platform as a Service

**AWS Cloud Formation:**

* Automation
* Consistency
* Replication
* Simplicity

**AWS Cloud Formation Components:**

* Template
* Stack
* Change set

**Docker Compose:**

Docker Compose will execute a YAML-based multi-container application. The YAML file consists of all configurations needed to deploy containers Docker Compose, which is integrated with Docker Swarm, and provides directions for building and deploying containers. With Docker Compose, each container is constructed to run on a single host.

* Container orchestration
* Docker compose CU
* Docker compose YML

**Docker Compose Features:**

* Container volume data
* Multiple isolated environment
* Environment variables
* Container updates

**Docker Compose Use Cases:**

* Development environments
* Automated testing environments
* Single host deployments

**Docker Compose Commands:**

* Docker-compose up-build
* Docker-compose down –volumes
* Docker-compose PS
* Docker-compose stop

**YML File Sections:**

* Version
* Services
* Networks
* Volumes

**YML File Services Directives:**

* Image
* Build
* DB
* Restart
* Volumes

**Advanced Docker Orchestration:**

* Purpose of orchestration with Docker and the tools involved
* Features of the Kubernetes container cluster management tool
* Rationale and use cases for multiple Docker containers and working with multiple hosts
* Purpose of the Docker swarm and how to setup, add nodes, deploy, scale, and delete
* Options for Docker configuration and how cloud providers manage orchestration
* How to use Docker compose; deploy an application stack and services to a swarm

**DEPLOYING AND RUNNING DOCKER CONTAINERS:**