Full Stack Assignment

DAY 4

1. Define a variable named `user age` and assign it the value of 25. Also, create an identifier `user name` and assign it the value "John Doe."

2.Create a variable called `pi` and assign it the value of the mathematical constant π (pi). Additionally, define a literal named `favourite number` and assign it the value 7.

3.Write a Python `while` loop that prints the squares of numbers from 1 to 5.

DAY 5

1.Write a Python program to print the following pattern:

```
1
22
333
4444
55555
```

2.Create a Java program to generate the following pattern:

```
A
AB
ABC
ABCD
ABCDE
```

3.Implement a C++ program to print the following pattern:

```
*****
 ****
  ***
   **
    *
```

DAY 6

1.Write a C program to find the sum of elements in a 1D array of integers.

2.Create a Python program to multiply two matrices (2D arrays) and print the result.

3.Implement a Java program to create a 2D jagged array to store and display the lengths of words in a sentence.

DAY 7

1.Write a C++ program to initialize and display a 3D array of integers.

2.Create a Java program to implement a 3D jagged array to store and display the heights of students in a classroom.

3.Write a Java program using the `Arrays` class to sort an array of strings in alphabetical order.

DAY 8

1.Explain the concept of memory segregation on RAM and how it influences program performance. Provide examples of situations where proper memory management is crucial.

2.Introduce the concept of Java String and discuss its significance in Java programming. Explain how strings are stored in memory and highlight the immutability of Java strings.

3.Explore the concept of immutable strings using the String class in Java. Discuss the advantages of using immutable strings and provide examples demonstrating their immutability.

DAY 9

1.Explain the concept of mutable strings in Java using the `StringBuffer` class. Compare the `String` and `StringBuffer` classes, highlighting scenarios where mutable strings are advantageous. Provide code examples to illustrate the mutability of `StringBuffer`.

2.Introduce the `StringBuilder` class in Java and discuss how it differs from `StringBuffer`. Provide examples showcasing the mutability of strings using `StringBuilder`. Highlight scenarios where `StringBuilder` is preferred over `StringBuffer`.

3.Design practice programs focusing on string manipulation. Include tasks such as reversing a string, finding the frequency of characters, and checking for palindromes. Encourage students to use both immutable and mutable approaches (String and StringBuilder/StringBuffer) where applicable.

DAY 10

1.Explain the concept of method overloading in Java. Provide examples to illustrate method overloading with different parameter types, numbers, and orders. Discuss the benefits and use cases of method overloading.

2.Discuss the use of static variables in Java. Explain their significance, initialization, and how they differ from instance variables. Provide examples demonstrating the behavior of static variables in both static and non-static contexts.

3.Explain the role of static blocks and static methods in Java. Provide examples to demonstrate the use of static blocks for initialization and static methods for utility operations. Discuss when to use static blocks and methods.

DAY 11

1.Explain the concept of encapsulation in object-oriented programming. Describe how encapsulation is achieved in Java and why it is important. Provide examples illustrating the use of access modifiers to control access to class members.

2.Discuss the significance of constructors in Java. Explain the types of constructors, including default, parameterized, and constructor overloading. Provide examples illustrating the use of constructors in initializing objects.

3.Combine the concepts of encapsulation and constructors to design a class that encapsulates the details of a bank account. Use private data members for account details and provide constructors for object initialization. Include methods for deposit, withdrawal, and balance inquiry, ensuring proper encapsulation.

DAY 12

1.Explain the concept of inheritance in Java. Discuss how it promotes code reusability and describe the relationship between superclasses and subclasses. Provide examples illustrating single inheritance and multiple inheritance.

2.Discuss the rules of inheritance in Java. Explain concepts such as method overriding, access modifiers, and the "super" keyword. Provide examples to illustrate how these rules are applied in practical scenarios.

3.Explain the types of methods in inheritance, including instance methods, static methods, and final methods. Discuss how these methods behave in the context of inheritance and when each type is appropriate. Provide examples to illustrate their usage.

DAY 13

1.Explain the delegation model in Java. Discuss how delegation is different from inheritance, and provide examples to illustrate the use of delegation in designing classes. Highlight scenarios where delegation is a preferable design choice.

2.Discuss the execution of constructors in inheritance. Explain how constructors are called in both the superclass and subclass during object creation. Provide examples to illustrate the sequence of constructor execution in single and multiple inheritance scenarios.

3.Explore the concept of constructor chaining in inheritance. Discuss how constructors can call other constructors within the same class or in the superclass. Provide examples illustrating constructor chaining and discuss its implications on object initialization.

DAY 14

1.Explain the concept of polymorphism in Java. Discuss the types of polymorphism, including compile-time (method overloading) and runtime (method overriding) polymorphism. Provide examples to illustrate each type and discuss the benefits of polymorphism in object-oriented programming.

2.Discuss the concept of abstraction in Java. Explain how abstraction involves hiding complex implementation details and exposing only relevant features through abstract classes and interfaces. Provide examples to illustrate the use of abstract classes and interfaces in achieving abstraction.

3.Explore the concept of runtime polymorphism through interfaces in Java. Discuss how interfaces allow for multiple inheritance and how they are used to achieve abstraction. Provide examples illustrating the implementation of interfaces and their impact on achieving code flexibility.

DAY 15

1.Discuss the rules of interfaces in Java. Explain the principles and limitations that govern the use of interfaces. Provide examples to illustrate how interfaces are declared, implemented, and used in Java programs.

2.Compare and contrast abstract classes and interfaces in Java. Discuss the similarities, differences, and scenarios where each is most appropriate. Provide examples to illustrate the use of abstract classes and interfaces in different situations.

3.Explain the concept of default methods in interfaces and how they address the issue of backward compatibility. Discuss the rules and considerations when using default methods. Provide examples to illustrate the use of default methods in interfaces.

DAY 16

1.Differentiate between errors and exceptions in Java. Explain the types of errors and exceptions, and discuss their impact on program execution. Provide examples to illustrate scenarios where errors and exceptions may occur.

2.Discuss the mechanisms for exception handling in Java. Explain the importance of exception handling in maintaining robust and reliable programs. Describe the key components of the exception handling mechanism, such as the try-catch block, throw statement, and the finally block.

3.Explain the try-catch block in Java for handling exceptions. Discuss the syntax, purpose, and best practices associated with try-catch blocks. Provide examples illustrating the use of try-catch blocks to handle different types of exceptions.

DAY 17

1.Explain the concept of "ducking" or "ignoring" exceptions in Java. Discuss the consequences of ignoring exceptions and situations where it might be appropriate. Provide examples to illustrate the concept of ducking exceptions.

2.Discuss the concept of rethrowing exceptions in Java. Explain when and why rethrowing exceptions might be necessary. Provide examples illustrating the scenarios where exceptions are caught and then rethrown.

3.Discuss the concept of custom exceptions in Java. Explain why and when custom exceptions are useful and how they can be created. Provide examples of creating and using custom exceptions to enhance the exception handling mechanism.

DAY 18

1.Explain the concept of a thread in Java. Discuss the advantages and challenges of using multithreading in a program. Provide examples illustrating the creation and execution of threads in Java.

2.Discuss the role of the thread scheduler in managing threads in a multithreaded environment. Explain how the scheduler determines the execution order of threads and the factors influencing thread scheduling. Provide examples illustrating the thread scheduling mechanism in Java.

3.Explain the concept of single-threaded programming. Discuss the advantages and limitations of single-threaded programming compared to multithreading. Provide examples illustrating scenarios where single-threaded programming is appropriate.

DAY 19

1.Explain how to implement multithreading in Java using the `Thread` class. Discuss the steps involved in creating and running multiple threads using the `Thread` class. Provide examples illustrating the implementation of multithreading using the `Thread` class.

2.Discuss the concept of multithreading in Java using the `Runnable` interface. Explain the advantages of implementing multithreading through the `Runnable` interface compared to extending the `Thread` class. Provide examples illustrating the implementation of multithreading using the `Runnable` interface.

3.Compare and contrast the implementation of multithreading using the `Thread` class and the `Runnable` interface in Java. Discuss the scenarios where each approach is preferable and the benefits of using one over the other. Provide examples to illustrate the differences and similarities.

DAY 20

1.Explain the concept of a race condition in multithreaded programming. Discuss the scenarios where race conditions may occur, the challenges they pose, and strategies for preventing or handling race conditions in Java.

2.Discuss the concept of daemon threads in Java. Explain what daemon threads are, their characteristics, and scenarios where they are useful. Provide examples illustrating the use of daemon threads in Java.

3.Discuss the common problems associated with multithreaded programming in Java. Explain issues such as deadlocks, livelocks, and resource contention. Provide examples to illustrate each problem and discuss strategies for avoiding or resolving them.

DAY 21

1.Explain the different states of a thread in Java. Discuss the life cycle of a thread, including the various states a thread can be in. Provide examples illustrating the transition between different thread states.

2.Discuss the concept of a deadlock in multithreaded programming. Explain what causes a deadlock, the conditions necessary for a deadlock to occur, and strategies for preventing or resolving deadlocks in Java.

3.Discuss the Producer and Consumer problem in the context of multithreading. Explain the challenges involved in designing a solution for the Producer and Consumer problem and provide examples illustrating how it can be implemented in Java.

DAY 22

1.Explain the concept of ArrayList in Java. Discuss its characteristics, advantages, and use cases. Provide examples demonstrating the creation, modification, and traversal of ArrayLists in Java.

2.Discuss the LinkedList data structure in Java. Explain its characteristics, advantages over ArrayList in certain scenarios, and use cases. Provide examples demonstrating the creation, modification, and traversal of LinkedLists in Java.

3.Compare and contrast ArrayList and LinkedList in Java. Discuss their differences in terms of performance, memory usage, and suitability for different use cases. Provide examples illustrating scenarios where one might be preferred over the other.

DAY 23

1.Explain the concept of `ArrayDeque` in Java. Discuss its characteristics, advantages, and use cases. Provide examples demonstrating the creation, modification, and traversal of `ArrayDeque` in Java.

2.Discuss the `PriorityQueue` data structure in Java. Explain its characteristics, advantages, and use cases. Provide examples demonstrating the creation, modification, and traversal of `PriorityQueue` in Java.

3.Discuss the `TreeSet`, `HashSet`, and `LinkedHashSet` data structures in Java. Explain their characteristics, advantages, and use cases. Provide examples demonstrating the creation, modification, and traversal of each set implementation in Java

DAY 24

1.Explain the process of sorting complex objects in Java. Discuss the challenges associated with sorting objects that do not have a natural ordering. Provide examples illustrating how to implement sorting for complex objects.

2.Discuss the `Comparable` interface in Java. Explain its role in enabling natural ordering for objects. Provide examples demonstrating how to implement the `Comparable` interface for complex objects to support sorting.

3.Discuss the `Comparator` interface in Java. Explain its purpose in providing custom sorting logic for objects. Provide examples illustrating how to implement the `Comparator` interface for complex objects to support custom sorting.

DAY 25

1.Explain the `Collections` class in Java. Discuss its role in providing utility methods for working with collections. Provide examples illustrating the use of methods from the `Collections` class for sorting, searching, and modifying collections.

2.Discuss the `HashMap` class in Java. Explain its characteristics, advantages, and use cases. Provide examples illustrating the creation, modification, and traversal of `HashMaps` in Java.

3.Discuss the `LinkedHashMap` and `TreeMap` classes in Java. Explain their characteristics, advantages, and use cases. Provide examples illustrating the creation, modification, and traversal of `LinkedHashMaps` and `TreeMaps` in Java.

DAY 26

1.Explain the concept of a Functional Interface in Java. Discuss its characteristics and the role of the `@FunctionalInterface` annotation. Provide examples illustrating the creation and use of functional interfaces, including scenarios involving lambda expressions.

2.Discuss the process of creating objects of interfaces and abstract classes in Java. Explain scenarios where creating objects of interfaces and abstract classes is beneficial. Provide examples illustrating the creation of objects for both interfaces and abstract classes.

3.Discuss the `Optional` class in Java. Explain its role in handling potentially null values and avoiding null pointer exceptions. Provide examples illustrating the use of `Optional` to work with nullable values in Java.

DAY 27

1.Design a collection-based project in Java. The project should involve the creation, modification, and retrieval of data using various collection classes, such as ArrayList, HashMap, or TreeSet. Implement functionalities that showcase the versatility of collections in handling different types of data.

2.Create a library management system using Java collections. Implement classes to represent books, users, and transactions. Utilize collection classes to efficiently manage the storage and retrieval of these entities. Implement features like borrowing and returning books, checking user history, and searching for books.

3.Develop a student enrollment system using Java collections. Create classes to represent students, courses, and enrollments. Use collection classes to manage student information, course details, and enrollment records efficiently. Implement functionalities like enrolling students in courses, checking course availability, and generating student transcripts.

DAY 28

1.Develop a collection-based project in Java that manages a task-tracking system. Create classes for tasks, users, and projects. Implement functionalities for creating tasks, assigning tasks to users, and tracking task completion

2.Design a collection-based project in Java for a contact management system. Implement classes for contacts, groups, and communication logs. Include functionalities such as adding new contacts, organizing contacts into groups, and logging communication activities.

3.Create a collection-based project in Java for a simple inventory management system. Implement classes for products, inventory, and transactions. Include functionalities such as adding new products, updating inventory quantities, and generating sales reports.

DAY 29

1.Provide a comprehensive introduction to databases. Define what a database is, and explain its significance in modern computing. Discuss the primary purposes and advantages of using databases in various applications.

2.Trace the history of databases, highlighting key milestones and developments. Discuss the evolution of database systems from early file-based systems to modern database management systems (DBMS). Explain how the need for efficient data management led to the development of database technologies.

3.Explain the concept of a Database Management System (DBMS) and its key features. Discuss how a DBMS facilitates data organization, retrieval, and manipulation. Provide examples of popular DBMS platforms and their applications in real-world scenarios.

DAY 30

1. Explain the process of creating a database using SQL. Provide a step-by-step guide on how to create a new database. Include considerations for specifying the database name, character set, and collation.

2.Explain how to view existing databases in SQL. Provide SQL queries to retrieve a list of all databases and information about a specific database.

3.Explain the process of dropping a database in SQL. Provide a step-by-step guide on how to drop an existing database, including considerations for ensuring data integrity.

Day-31:

1. Explain the differences between DDL and DML statements in SQL. Provide examples for each type of statement.

2. What are three common DDL statements in SQL? Provide a simple example for each.

3.Explain the basic usage of three DML statements in SQL: INSERT, UPDATE, DELETE.

Day-32:

1. What is the purpose of the SELECT statement in SQL?

2.Explain three types of operators in MySQL with simple examples

3.What does the ORDER BY clause do in a SELECT statement? Provide a straightforward example.

Day-33:

1.What is the purpose of the SELECT statement in SQL?

2. Explain three types of operators in MySQL with simple examples.

3.What does the ORDER BY clause do in a SELECT statement? Provide a straightforward example.

Day-34:

1. Explain the use of the GROUP BY clause in a SELECT statement. Provide an example demonstrating its usage.

2.Describe the purpose of the HAVING clause in a SELECT statement with the GROUP BY clause. Provide an example illustrating its use.

3.Explain the usage of the BETWEEN clause in a SELECT statement. Provide an example demonstrating how it works.

Day-35:

1.Explain the concept of Views in SQL. What are the benefits of using views, and how are they different from tables?

2.Describe the purpose of Indexes in a database. Provide an example of a scenario where creating an index would be beneficial.

3.Explain the concept of Stored Procedures in SQL. Provide an example of a simple stored procedure.

Day-36:

1.Explain the purpose of the `<html>`, `<head>`, `<title>`, and `<body>` tags in HTML. Provide a simple example of their usage.

2.Describe the usage of headings, paragraphs, and lists in HTML. Provide examples for each.

3.Explain the use of the `<strong>` and `<em>` tags in HTML. Provide an example to illustrate their usage.

Day-37:

1.Explain the purpose of the `<meta>` tag in HTML and how it is used to specify the Unicode character set. Provide an example.

2.Describe the purpose of anchor tags (`<a>`) and the `href` attribute. Provide examples of linking to other websites and linking to pages within a website.

3.Explain how to open a link in a new browser window or tab using HTML. Provide an example.

Day-38:

1.Explain the purpose of the `<img>` tag and the `src` attribute in HTML. Provide an example and describe the significance of the `alt` attribute.

2.Describe how to use the `width`, `height`, and `alt` attributes in the `<img>` tag. Provide an example to illustrate their usage.

3.Explain the purpose of the `<hr>` tag in HTML and how to use it to create horizontal rules. Introduce the `<style>` tag and describe its role in adding CSS to HTML.

Day-39:

1.Explain the purpose of the `class` **attribute in HTML. Provide an example demonstrating its usage.**

2.Describe how CSS class selectors work. Provide an example illustrating the usage of class selectors.

3.Explain the purpose of the `<span>` **tag in HTML. Provide an example demonstrating its usage.**

Day-40:

1.Explain the purpose of the `<div>` tag in HTML and how it is used to divide content. Provide an example illustrating the use of `<div>`.

2.Describe the purpose of assigning IDs to `<div>` elements. Provide an example of how to assign and use IDs in HTML and CSS.

3.Explain the usage of the `width` and `max-width` properties in CSS. Provide an example illustrating how to set these properties.

Day-41:

1.Explain the steps to open the DevTools in Google Chrome. What are the primary panels available in DevTools?

2.Describe how to edit HTML using the DevTools Elements panel. Provide an example of making a simple edit.

3.Explain how to enable, disable, and edit CSS using the DevTools. Provide an example of fine-tuning CSS.

Day-42:

1.Provide a brief overview of JavaScript and its historical background.

2.Explain the relationship between JavaScript and ECMAScript. Why is ECMAScript significant in the context of JavaScript?

3.Discuss variable declaration, variable scope, and block scope in JavaScript.

Day-44:

1.Explain the concept of error handling in JavaScript. How can errors be thrown and caught in JavaScript?

2.Describe the difference between number literals and the Number object in JavaScript. Provide examples of each.

3.Explain the purpose of the Math object in JavaScript. Provide examples of methods available in the Math object.

Day-45:

1. Explain the concept of string literals in JavaScript. Provide an example of using string literals.

2.Describe the purpose of the String object in JavaScript. Provide examples of methods available in the String object.

3.Explain the process of creating and populating arrays in JavaScript. Provide an example of an array and how to add elements to it.

Day-46:

1.Explain the process of defining functions in JavaScript. Provide an example of a simple function.

2.Describe the concept of closures in JavaScript. Provide an example illustrating the use of closures.

3.Explain the differences between the Set object type and the Map object type in JavaScript. Provide examples of using both.

Day-47:

1.Explain the document structure in HTML and how it relates to the DOM. Provide an example of an HTML document structure.

2.Describe how to select document elements using query selectors in JavaScript. Provide an example.

3.Explain how to create, change, and delete nodes in the DOM using JavaScript. Provide an example that demonstrates these operations.

Day-48:

1.Explain the concept of event propagation in JavaScript. How do events propagate through the DOM, and what is event bubbling?

2.Describe the process of registering and invoking event handlers in JavaScript. Provide an example illustrating the registration of a click event handler.

3.Explain the concept of the event object in JavaScript. How can it be used in event handling?

Day-49:

1.Provide an introduction to AngularJS. What is AngularJS, and what are its key features?

2.Explain the concepts of MVC (Model-View-Controller) architecture and modular architecture in the context of AngularJS.

3.Describe the steps to set up the environment for AngularJS development. What tools and dependencies are commonly used?

Day-50:

1.Explain the concept of Number and String Expressions in Angular. How are expressions used in Angular templates, and provide an example?

2.Describe Object Binding and Expressions in Angular. How can you bind object properties to the view, and provide an example?

3.Explain the concept of Working with Arrays in Angular. How can you iterate over an array in the template, and provide an example?

Day-51:

1.Explain the concept of Conditional Directives in Angular. Provide an example of using the `*ngIf` directive and describe how it works.

2.Describe Styles Directives in Angular. How can you dynamically apply styles to elements using the `[style]` directive? Provide an example.

3.Explain the concept of Mouse and Keyboard Events Directives in Angular. Provide an example of using `(click)` and `(keyup)` events and describe how they work.

Day-52:

1.Explain the concept of Controllers in Angular. What role do controllers play in an Angular application, and how are they associated with the DOM?

2.Describe the programming of controllers and the use of the $\mathit{\$scope}$ object in Angular. How does the $\mathit{\$scope}$ object facilitate communication between controllers and views?

3.Explain how behavior is added to a $\mathit{\$scope}$ object in Angular. Provide an example demonstrating the addition of a method to handle a user action.

Day-53:

1.Explain the concept of Built-In Filters in Angular. What is the purpose of filters, and how are they applied to data in the view?

2.Describe the Uppercase and Lowercase Filters in Angular. How can these filters be applied to text data, and provide an example?

3.Explain the Currency and Number Formatting Filters in Angular. How can these filters be used to format numeric data, and provide an example?

Day-54:

1.Provide an introduction to JDBC (Java Database Connectivity). What is the role of JDBC in Java, and how does it facilitate database connectivity?

2.Explain the types of JDBC Drivers. What are the four types of JDBC drivers, and how do they differ in terms of architecture and implementation?

3.Describe the PreparedStatement interface in JDBC. What is its purpose, and how does it differ from a Statement?

Day-55:

1.Explain the concept of CRUD operations in the context of database interactions. What are the key operations involved in CRUD, and how do they correspond to SQL statements?

2.Demonstrate the use of the Statement interface for the CRUD operation "Read" in JDBC. How can you execute a SELECT query using the Statement interface, and provide an example?

3.Illustrate the use of the Statement interface for the CRUD operation "Create" in JDBC. How can you execute an INSERT query using the Statement interface, and provide an example?

Day-56:

1.Explain the significance of the PreparedStatement interface in JDBC. How does it enhance the execution of CRUD operations compared to the Statement interface?

2.Illustrate the use of the PreparedStatement interface for the CRUD operation "Update" in JDBC. How can you execute an UPDATE query using the PreparedStatement interface, and provide an example?

3.Demonstrate the use of the PreparedStatement interface for the CRUD operation "Delete" in JDBC. How can you execute a DELETE query using the PreparedStatement interface, and provide an example?

Day-57:

1.Define the concept of transaction management in the context of databases. What is a database transaction, and why is it important in ensuring data consistency?

2.Define the concept of transaction management in the context of databases. What is a database transaction, and why is it important in ensuring data consistency?

3.Describe the methods provided by JDBC for transaction management. How can you use the `commit` and `rollback` methods to control transactions in JDBC?

Day-58:

1.Define Web Application Architecture and its key components. What are the primary layers in a typical web application, and how do they collaborate to handle client requests?

2.Explain the concept of Servlet Chaining in a Java web application. How does servlet chaining allow multiple servlets to process a single client request, and what are its advantages?

3.Illustrate a scenario where Servlet Chaining can be beneficial. Provide a practical example of using multiple servlets in a chain to handle a specific client request.

Day-59:

1.Explain how Servlet Chaining can be implemented using the Model-View-Controller (MVC) architecture in a Java web application. How do the components of the MVC pattern collaborate in the context of servlet chaining?

2.Discuss the role of Sessions in a Java web application. What is the purpose of sessions, and how do they contribute to maintaining state information between client and server?

3.Provide a practical example of Servlet Chaining using the MVC architecture and incorporating session management. How can multiple servlets collaborate to process a request, and how can sessions be used to maintain user-specific data?

Day-60:

1.Explain the integration of Servlets and JDBC in a Java web application. How can Servlets be used to interact with a database using JDBC, and what are the key steps involved?

2.Discuss the role of Cookies in a web application. How are Cookies used to store and retrieve information on the client side, and what are their advantages and limitations?

3.Provide a practical example of using Servlets with JDBC and Cookies in a web application. How can a Servlet interact with a database to retrieve information, and how can Cookies be used to store user-specific data on the client side?

Day-61:

1.Explain the architecture of Hibernate in a Java application. What are the key components and their roles in the Hibernate framework?

2.Discuss the CRUD operations in Hibernate. What are the four main operations, and how do they correspond to Create, Read, Update, and Delete in a database?

3.Illustrate a scenario where Hibernate can be beneficial for CRUD operations. Provide a practical example, including entity mappings and database interactions.

Day-62:

1.Explain the concept of mapping in Hibernate. What is the purpose of mapping, and how does it establish a relationship between Java objects and database tables?

2.Discuss the different types of associations in Hibernate mapping. What are the main types of relationships between entities, and how are they defined in the mapping annotations or XML configuration?

3.Explain the concept of Hibernate Inheritance Mapping. How does Hibernate handle inheritance in entity classes, and what are the strategies for mapping inheritance relationships?

Day-63:

1.Explain the concept of Hibernate mapping. What is the purpose of mapping in Hibernate, and how does it establish a connection between Java objects and database tables? Provide an example with annotations.

2.Discuss the importance of associations in Hibernate mapping. What are the different types of associations, and how are they represented in entity classes using annotations? Provide an example.

3.Explain Hibernate Inheritance Mapping. What are the different strategies for mapping inheritance relationships, and how are they implemented using annotations? Provide an example.

Day-64:

1.Design the database schema for an online bookstore using Hibernate. Define entities such as `Book`, `Author`, and `Category`. Implement relationships between these entities, and provide annotations for mapping. Include a one-to-many relationship between `Author` and `Book`, and a many-to-one relationship between `Book` and `Category`.

2.Create a Servlet-based web application for the online bookstore project. Implement servlets for displaying a list of books, details of a specific book, and a form for adding a new book. Utilize Hibernate to interact with the database. Implement proper exception handling and error messages.

3.Implement user authentication and authorization in the online bookstore project using Servlets and Hibernate. Create a `User` entity with roles (e.g., `USER` and `ADMIN`). Design servlets for user registration, login, and logout. Restrict access to certain servlets based on user roles.

Day-65:

1.Explain the architecture of the Spring framework. Describe the key components and their roles in the Spring architecture. How does Spring support modularity and extensibility?

2.Compare Spring with Enterprise JavaBeans (EJB). Discuss the similarities and differences between the two technologies. In what scenarios would you prefer using Spring over EJB, and vice versa?

3.Explain the concept of Inversion of Control (IoC) in the context of the Spring framework. How does IoC promote loose coupling and enhance the testability of applications? Provide a practical example.

Day-66:

1.Explain the concept of Dependency Injection (DI) using setter methods in the context of the Spring framework. How does this approach enhance modularity and maintainability in a Java application? Provide an example with annotations.

2.Discuss the benefits of using Constructor Injection for Dependency Injection in the Spring framework. How does Constructor Injection contribute to better testing and why is it considered a recommended practice in Spring applications? Provide an example with annotations.

3.Compare and contrast Setter Injection and Constructor Injection in the context of Spring Dependency Injection. What are the scenarios where one approach might be more suitable than the other? Provide examples illustrating the use of both Setter Injection and Constructor Injection.

Day-67:

1.Explain the architecture of the Spring MVC framework. Describe the key components and their roles in the Spring MVC architecture. How does the Model, View, and Controller interact to handle HTTP requests and responses in a Spring MVC application?

2.Create a simple Spring MVC application for managing a list of products. Implement a controller to handle requests for displaying the list of products. Use Thymeleaf as the view template engine. Provide a step-by-step explanation of the code and configuration involved.

3.Implement form handling in the Spring MVC application created in Assignment 2. Create a controller method for displaying a form to add a new product and another method for processing the form submission. Use Thymeleaf for rendering the form and display appropriate messages after form submission.

Day-68:

1.Explain the significance of annotations in Spring. Provide a brief overview of commonly used annotations in a Spring application, such as `@Component`, `@Autowired`, `@RequestMapping`, and `@Service`. How do these annotations contribute to the development of a well-organized and modular Spring application?

2.Integrate a Spring Eureka Server into a microservices architecture. Explain the role of Eureka Server in service registration and discovery. Provide an example configuration for a simple Eureka Server setup, and describe how microservices can register themselves with the server.

3.Discuss the use of the `@FeignClient` annotation in a Spring Cloud microservices architecture. What is its purpose, and how does it simplify communication between microservices? Provide an example scenario and demonstrate how `@FeignClient` is used.

Day-69:

1. Explain the role of the Spring Boot JDBC template in a Spring Boot application. Discuss how it simplifies database interactions and provides a higher-level abstraction over traditional JDBC. Provide an example scenario demonstrating the use of the JDBC template for querying a database in a Spring Boot application.

2.Discuss the benefits of using named parameters in Spring Boot JDBC template. How does the usage of named parameters improve the readability and maintainability of SQL queries in a Spring Boot application? Provide an example demonstrating the use of named parameters in a JDBC template query.

3.Explain how Spring Boot manages transactions with the JDBC template. Discuss the benefits of using programmatic transaction management and provide an example demonstrating the use of programmatic transactions with the JDBC template in a Spring Boot application.

Day-70:

1.Explain the concept of entity relationships in Spring Data JPA. Discuss the types of relationships supported (e.g., @OneToOne, @OneToMany) and provide an example scenario illustrating the use of such relationships in a Spring Boot application.

2.Discuss the concept of auditing in Spring Data JPA. How can you enable entity auditing to track changes such as creation and modification timestamps? Provide an example demonstrating the use of auditing annotations in a Spring Boot application.

3.Explain the use of Spring Data JPA projections. What are projections, and how do they help in retrieving specific subsets of data from entities? Provide an example scenario demonstrating the use of projections in a Spring Boot application.

Day-71:

1.Explain the concept of RESTful web services in the context of Spring. Discuss the key principles of REST, such as statelessness and uniform interface, and how Spring facilitates the development of RESTful APIs. Provide an example scenario illustrating the creation of a simple RESTful service using Spring.

2.Discuss the use of Spring Boot and the `@RestController` annotation in building RESTful web services. How does Spring Boot simplify the development of RESTful APIs, and how does the `@RestController` differ from traditional MVC controllers in Spring? Provide an example demonstrating the creation of a simple RESTful service using Spring Boot and `@RestController`.

3.Explain the concept of HATEOAS (Hypermedia as the Engine of Application State) in the context of RESTful web services. How does Spring support HATEOAS, and what benefits does it bring to RESTful APIs? Provide an example scenario demonstrating the use of HATEOAS in a Spring RESTful service.

Day-72:

1.Discuss the role of content negotiation in Spring RESTful services. What is content negotiation, and how does Spring support it? Provide an example scenario demonstrating the use of content negotiation in a Spring Boot application.

2.Explain the use of exception handling in Spring RESTful services. How does Spring handle exceptions in the context of RESTful APIs, and what mechanisms are available for customizing error responses? Provide an example scenario demonstrating the implementation of exception handling in a Spring Boot RESTful service.

Day-73:

1. Design a Spring Boot project for a simple online bookstore. Implement entity classes for books, authors, and customers. Utilize JPA for data persistence and retrieval. Include functionalities such as adding new books, searching for books, and managing customer orders.

2.Create a Spring Boot project for a task management system. Use JPA for data persistence and the JDBC template for database interactions. Implement entities for tasks, users, and projects. Include functionalities such as creating tasks, assigning tasks to users, and tracking project progress.

3.Develop a Spring Boot project for a blogging platform. Utilize JPA for data persistence and JDBC template for database interactions. Implement entities for posts, users, and comments. Include functionalities such as creating blog posts, commenting on posts, and managing user profiles.

Day-74:

1.Design a Spring Boot project for an e-commerce platform. Utilize JPA for data persistence and the JDBC template for database interactions. Implement entities for products, orders, and customers. Include functionalities such as adding products to a shopping cart, processing orders, and managing customer accounts.

2.Create a Spring Boot project for a student enrollment system. Utilize JPA for data persistence and JDBC template for database interactions. Implement entities for students, courses, and enrollments. Include functionalities such as enrolling students in courses, checking course availability, and generating student transcripts.

Day-75:

1. Design a Spring Boot project for a contact management system. Utilize JPA for data persistence and JDBC template for database interactions. Implement entities for contacts, groups, and communication logs. Include functionalities such as adding new contacts, organizing contacts into groups, and logging communication activities.

2.Create a Spring Boot project for a movie rental system using JPA. Design entities for movies, customers, and rentals. Implement functionalities such as renting movies, returning movies, and generating reports on customer rental history.

Day-76:

1.Explain the concept of NoSQL databases. Provide an overview of what NoSQL is and highlight its key characteristics. Discuss scenarios where NoSQL databases are suitable compared to traditional relational databases.

2.Examine the differences between NoSQL and Relational Database Management Systems (RDBMS). Discuss key distinctions in terms of data models, schema, scalability, and use cases.

3.Discuss the benefits of using NoSQL databases. Explain scenarios where NoSQL databases outperform traditional relational databases and highlight the advantages they bring to modern application development.

Day-77:

1.Explain the design goals of MongoDB. Discuss how MongoDB aims to address the limitations of traditional relational databases and achieve its objectives.

2.Introduce the MongoDB Shell and its significance in MongoDB. Discuss how developers can use the MongoDB Shell to interact with the database, execute queries, and perform administrative tasks.

3.Introduce JSON (JavaScript Object Notation) and its significance in MongoDB. Discuss the structure of JSON and how it is used for representing data in MongoDB.

Day-78:

1.Provide a guide on installing the necessary tools for working with MongoDB, including the MongoDB server and the MongoDB Shell. Explain the steps involved in the installation process.

2.Provide an overview of a blog project built using Node.js, Express, and MongoDB. Discuss the role of Swig as a template engine and the significance of Node Packaged Modules (npm) in managing project dependencies.

3.Explain the CRUD operations (Creating, Reading, and Updating Data) in MongoDB using the Mongo Shell. Provide examples of how developers can perform these operations interactively using the Mongo Shell.

Day-79:

1.Explain how indexes contribute to improving performance in MongoDB. Discuss the types of indexes available in MongoDB and how they can be used to optimize query execution.

2.Discuss the importance of monitoring and understanding performance in MongoDB. Outline the key metrics and tools available for monitoring MongoDB performance, and explain how they contribute to maintaining a healthy database system.

3.Examine the challenges and strategies associated with achieving high performance in sharded environments in MongoDB. Discuss how sharding impacts database scalability and how developers can optimize performance in sharded MongoDB clusters.

Day-80:

1.Discuss the goals of the MongoDB Aggregation Framework. Explain how the Aggregation Framework enables developers to perform complex data transformations and manipulations in MongoDB.

2.Explain the concept of the pipeline in the MongoDB Aggregation Framework. Discuss the stages that can be included in a pipeline and their significance in performing data transformations.

3.Compare the MongoDB Aggregation Framework with SQL facilities. Discuss how the Aggregation Framework provides similar functionality to SQL GROUP BY, JOIN, and other operations, highlighting its advantages in the context of MongoDB.