

Sr no.	Topic	Duration (Mins)	Session No (2 Hours)	Session No (4 Hours)
1	Overview of Java and its Architecture	20	1	1
2	Compiling Source Code and Packaging Applications	20	1	1
3	Java Console based and Web based Applications	30	1	1
4	Deployment to Tomcat and Consuming Java Applications	30	1	1
5	OOPs Concept	30	2	1
6	Practical	50	2	1
7	Practical	60	2	1

Introduction to Java Concepts

1. Overview of Java and its Architecture

Information

- Java is a popular programming language that was developed by Sun Microsystems (now owned by Oracle Corporation).
- It is widely used for building a wide range of applications, including desktop software, web applications, mobile apps, enterprise systems, and more.
- Java is known for its platform independence, robustness, security features, and extensive ecosystem of libraries and frameworks.

A. Java Programming Language:

Java is an object-oriented programming language with syntax and features derived from C and C++. Some key characteristics of Java include:

1. Platform Independence:

- Java programs are compiled into bytecode, which can run on any platform with a Java Virtual Machine (JVM).
- This "write once, run anywhere" approach allows Java applications to be portable across different operating systems and hardware architectures.

2. Object-Oriented Programming (OOP):

- Java is designed around the principles of OOP, including encapsulation, inheritance, and polymorphism.
- It provides robust support for creating reusable and modular code through classes, objects, and interfaces.

3. Automatic Memory Management:

- Java uses a garbage collector to automatically manage memory allocation and deallocation, relieving developers from manual memory management tasks.
- This simplifies memory management and helps prevent memory leaks and other memory-related issues.

4. Rich Standard Library:

- Java comes with a comprehensive standard library (Java API) that provides a wide range of pre-built classes and functions for common programming tasks, such as file I/O, networking, threading, database access, and more.
- This reduces the need to write low-level code from scratch and speeds up development.

5. Strong Community and Ecosystem:

- Java has a vast and active community of developers, which contributes to the extensive ecosystem of libraries, frameworks, and tools.
- These resources offer solutions for various domains and make development faster and more efficient.

6. Security and Safety:

- Java has built-in security features that protect against various vulnerabilities and risks.
- It enforces strict access controls, provides a sandboxed execution environment for untrusted code, and has robust mechanisms for handling exceptions and error handling.

7. Scalability and Performance:

- Java provides excellent scalability and performance capabilities.
- It offers multi-threading and concurrency support, allowing developers to create highly scalable applications that can handle multiple concurrent tasks efficiently.
- Additionally, Java has advanced performance optimization techniques, such as Just-In-Time (JIT)

compilation, which dynamically compiles bytecode into machine code for improved execution speed.

8. Enterprise-Ready:

- Java has strong support for building enterprise-level applications.
- The Java Enterprise Edition (Java EE) platform provides a set of specifications and APIs for developing scalable, secure, and distributed systems.
- Java EE includes features like servlets, JavaServer Pages (JSP), Enterprise JavaBeans (EJB), and Java Persistence API (JPA) for database access.

B. Java Architecture:

- Java architecture refers to the various components, patterns, and principles that guide the design and organization of Java-based applications.
- Java is a versatile programming language widely used for building a wide range of applications, from simple desktop programs to complex enterprise systems.
- Here are some key aspects of Java architecture:

1. Java Platform:

- The Java platform consists of two main components: the Java Development Kit (JDK) and the Java Runtime Environment (JRE).

- The JDK includes tools for developing, compiling, and debugging Java applications, while the JRE provides the runtime environment needed to execute Java applications.

2. Java Virtual Machine (JVM):

- The JVM is a critical component of Java architecture. It provides a platform-independent execution environment for Java applications.
- Java source code is compiled into bytecode, which the JVM interprets and runs.
- Different JVM implementations are available for various platforms, ensuring Java's "write once, run anywhere" capability.

3. Object-Oriented Programming (OOP):

- Java is known for its strong support for object-oriented programming.
- It follows OOP principles like encapsulation, inheritance, and polymorphism, allowing developers to create modular, maintainable, and extensible code.

4. Application Components:

- Java applications are typically organized into classes, packages, and modules.
- Classes define the data and behavior of objects, packages group related classes, and modules

(introduced in Java 9) provide a more structured way to organize code and encapsulate dependencies.

5. Java APIs and Libraries:

- Java provides a rich set of libraries and APIs that cover a wide range of functionalities, such as I/O operations, networking, GUI development, data manipulation, and more.
- These APIs enable developers to build applications efficiently by leveraging pre-built components.

6. Design Patterns:

- Java applications often incorporate design patterns, which are reusable solutions to common software design problems.
- Patterns like Singleton, Factory, Observer, and MVC help structure code, promote code reusability, and enhance maintainability.

7. Java EE (Java Platform, Enterprise Edition):

- Java EE, now known as Jakarta EE, extends the Java SE platform to provide APIs and tools for building enterprise-level applications.
- It includes components for web applications, messaging, persistence (JPA), and more.
- Java EE applications often run on application servers like Apache Tomcat, WildFly, or IBM WebSphere.

8. Spring Framework:

- The Spring Framework is a popular Java framework that simplifies the development of Java applications by providing comprehensive infrastructure support.
- It offers features like dependency injection, aspect-oriented programming, and more.
- Spring Boot, a part of the Spring ecosystem, streamlines the process of building production-ready applications.

9. Microservices Architecture:

- Java is commonly used for building microservices-based architectures, where applications are broken down into smaller, independent services that communicate via APIs.
- Spring Boot and Spring Cloud provide tools for developing and managing microservices.
- Java's architecture flexibility, portability, and rich ecosystem make it a preferred choice for building a wide range of applications, from desktop software to large-scale enterprise systems and web applications.
- Developers can choose the appropriate architecture and frameworks based on the specific requirements of their projects.

1. JDK Installation:

- Visit the Oracle JDK download page (<https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>).
- Accept the license agreement.
- Choose the appropriate JDK version for your operating system and download the installer.
- Run the installer and follow the installation instructions. Specify the installation directory for the JDK.
- After the installation is complete, set the JAVA_HOME environment variable to point to the JDK installation directory.
- This step is necessary for running Java applications and compiling Java source code from the command line.

2. JRE Installation:

- The JRE is typically included in the JDK installation. However, if you only need to run Java applications and not develop them, you can install the JRE separately.

- Visit the Oracle JRE download page (<https://www.oracle.com/java/technologies/javase-jre8-downloads.html>).
- Accept the license agreement.
- Choose the appropriate JRE version for your operating system and download the installer.
- Run the installer and follow the installation instructions.
- After the installation is complete, the JRE should be ready to use.

3. JVM (Java Virtual Machine):

- The JVM is included in both the JDK and JRE installations.
- You don't need to separately install the JVM.

4. Eclipse Installation:

- Visit the Eclipse downloads page (<https://www.eclipse.org/downloads/>).
- Choose the appropriate Eclipse package based on your Java development needs. For Java development, you can choose "Eclipse IDE for Java Developers" or "Eclipse IDE for Enterprise Java Developers".
- Download the installer for your operating system.
- Run the installer and follow the installation instructions. You may need to specify the JDK location during the installation process.

- After the installation is complete, you can launch Eclipse and start using it for Java development.
- Make sure to verify the system requirements for each software component and choose the appropriate versions based on your operating system and specific needs.
- Once everything is installed, you can start creating and running Java applications using Eclipse or by using the command line tools provided by the JDK.
- Remember to set up your project and configure the build path in Eclipse to use the installed JDK and JRE.

Variables :

- In programming, variables are used to store and manipulate data.
- They act as named placeholders or containers that hold values of different types, such as numbers, text, or objects.
- Variables enable the flexibility and dynamic nature of programming by allowing values to be assigned, modified, and retrieved during the execution of a program.

1. Declaration:

Variables must be declared before they can be used.

The declaration specifies the variable's name and its data type, which determines the kind of values it can store. For example:

```
int age; // Declaration of an integer variable named 'age'  
String name; // Declaration of a String variable named 'name'  
double salary; // Declaration of a double variable named 'salary'
```

2. Initialization:

- Variables can be initialized with an initial value at the time of declaration or at a later point in the program.
- Initialization assigns a value to the variable, allowing it to hold data. For example:

```
int age = 25; // Initialization of 'age' variable with the value 25  
String name = "John"; // Initialization of 'name' variable with  
the value "John"  
double salary; // Declaration of 'salary' variable  
salary = 5000.50; // Assignment of value 5000.50 to 'salary'  
variable
```

3. Data Types:

- In Java, data types define the type and size of values that can be stored in variables or used as parameters in methods.
- Java supports two main categories of data types: primitive data types and reference data types.

a. Primitive Data Types:

- Primitive data types represent basic, built-in value types and are not objects.
- They have a fixed size and are directly stored in memory.
- byte: Represents a signed 8-bit integer.
- short: Represents a signed 16-bit integer.
- int: Represents a signed 32-bit integer.
- long: Represents a signed 64-bit integer.
- float: Represents a single-precision 32-bit floating-point number.
- double: Represents a double-precision 64-bit floating-point number.
- char: Represents a single 16-bit Unicode character.
- boolean: Represents a boolean value (true or false).

b. Reference Data Types:

- Reference data types refer to objects or instances of classes.
- They do not store the actual data but store references to the memory location where the data is stored.

i. Classes:

- User-defined reference types that represent objects with specific attributes and behaviors.

ii. Interfaces:

- Define a contract that classes can implement, specifying a set of methods that must be provided by implementing classes.

iii. Arrays:

- Ordered collections of elements of the same data type.

Operators:

In programming, operators are symbols or keywords that perform operations on one or more operands (values or variables) and produce a result.

Operators allow you to perform various computations, comparisons, and manipulations in your code.

1. Arithmetic Operators:

- + Addition: Adds two operands together.
- -Subtraction: Subtracts the second operand from the first operand.
- *Multiplication: Multiplies two operands.
- / Division: Divides the first operand by the second operand.
- % Modulus: Returns the remainder after division.

2. Assignment Operators:

- = Assignment: Assigns the value on the right to the variable on the left.
- += Addition assignment: Adds the right operand to the left operand and assigns the result to the left operand.
- -= Subtraction assignment: Subtracts the right operand from the left operand and assigns the result to the left operand.
- *= Multiplication assignment: Multiplies the left operand by the right operand and assigns the result to the left operand.
- /= Division assignment: Divides the left operand by the right operand and assigns the result to the left operand.
- %= Modulus assignment: Computes the modulus of the left operand and the right operand and assigns the result to the left operand.

3. Comparison Operators:

- == Equality: Checks if two operands are equal.
- != Inequality: Checks if two operands are not equal.
- > Greater than: Checks if the left operand is greater than the right operand.
- < Less than: Checks if the left operand is less than the right operand.
- >= Greater than or equal to: Checks if the left operand is greater than or equal to the right operand.
- <= Less than or equal to: Checks if the left operand is less than or equal to the right operand.

4. Logical Operators:

- && Logical AND: Returns true if both operands are true.
- || Logical OR: Returns true if at least one of the operands is true.
- ! Logical NOT: Negates the boolean value of the operand.

5. Increment and Decrement Operators:

- ++ Increment: Increments the value of the operand by 1.
- --Decrement: Decrements the value of the operand by 1.

6. Bitwise Operators:

- & Bitwise AND: Performs a bitwise AND operation on the binary representations of the operands.
- | Bitwise OR: Performs a bitwise OR operation on the binary representations of the operands.
- ^ Bitwise XOR: Performs a bitwise exclusive OR (XOR) operation on the binary representations of the operands.
- ~ Bitwise complement: Flips the bits of the operand, changing 0s to 1s and vice versa.

- << Bitwise left shift: Shifts the bits of the left operand to the left by the number of positions specified by the right operand.
- >> Bitwise right shift: Shifts the bits of the left operand to the right by the number of positions specified by the right operand.
- These are just a few examples of operators in Java.
- Each operator serves a specific purpose and is used to perform different types of operations on operands.
- Understanding and correctly using operators is crucial for writing efficient and effective code.

Classes :

- In object-oriented programming (OOP), a class is a blueprint or template that defines the properties (attributes) and behaviors (methods) of objects.
- It serves as a blueprint for creating instances or objects of that class.
- A class provides a way to encapsulate related data and functionality into a single unit.

1. Objects and Instances:

- A class is used to create objects or instances.
- An object is a specific occurrence or realization of a class, representing a unique entity with its own set of attribute values.
- For example, a "Car" class can be used to create individual car objects with their own specific characteristics.

2. Attributes (Data Members):

- Attributes, also known as data members or fields, are the variables associated with a class.
- They represent the state or characteristics of objects created from the class.
- For example, a "Car" class might have attributes like "color," "brand," and "price."

3. Methods (Member Functions):

- Methods are functions defined within a class that define the behavior or actions that objects created from the class can perform.
- Methods can access and manipulate the attributes of the class.
- For example, a "Car" class might have methods like "startEngine," "accelerate," and "brake."

4. Encapsulation:

- Encapsulation is a fundamental concept in OOP that combines data and methods into a single unit (class) and hides the internal implementation details.
- The attributes of a class are typically declared as private or protected, and methods provide controlled access to those attributes through public interfaces.

5. Inheritance:

- Inheritance is a mechanism in OOP that allows a class (subclass) to inherit the properties and methods of another class (superclass).
- The subclass can extend or modify the behavior of the superclass while reusing its existing attributes and methods.

1. Polymorphism:

- Polymorphism allows objects of different classes to be treated as objects of a common superclass.
- This allows for code reuse and flexibility. Polymorphism is achieved through method overriding and method overloading.

2. Constructors:

- Constructors are special methods used to initialize objects of a class.
 - They are typically called when an object is created using the 'new' keyword.
 - Constructors have the same name as the class and may accept parameters to set initial attribute values.
-
- Classes provide a way to structure and organize code, making it easier to manage and maintain complex systems.
 - They facilitate code reuse, encapsulation, and modularity, enabling the development of scalable and maintainable applications.

Conditional Statements :

- Conditional statements, also known as control structures, are programming constructs that allow the execution of different blocks of code based on specific conditions.
- These statements evaluate a condition or a set of conditions and then perform different actions depending on the outcome of the evaluation.

1. If Statement:

- The if statement is the simplest form of a conditional statement.
- It evaluates a condition and executes a block of code if the condition is true.
- Here's the basic syntax of an if statement in Java:

```
if (condition) {  
    // Code to be executed if the condition is true  
}
```

2. If-Else Statement:

- The if-else statement extends the if statement by providing an alternative block of code to be executed when the condition is false.
- Here's the syntax:

```
if (condition) {  
    // Code to be executed if the condition is true  
} else {  
    // Code to be executed if the condition is false  
}
```

3.1

- The if-else-if statement allows multiple conditions to be evaluated in sequence.
- It provides multiple alternative blocks of code, each associated with a specific condition.
- Here's the syntax:

```
if (condition1) {  
    // Code to be executed if condition1 is true  
} else if (condition2) {  
    // Code to be executed if condition1 is false and condition2  
    is true  
} else {  
    // Code to be executed if all conditions are false  
}
```

4. Switch Statement:

- The switch statement allows for multi-way branching based on the value of an expression.

- It provides different cases, each representing a possible value of the expression, and executes the code block associated with the matching case.
- Here's the syntax:

```
switch (expression) {  
  case value1:  
    // Code to be executed if expression matches value1  
    break;  
  case value2:  
    // Code to be executed if expression matches value2  
    break;  
  // more cases...  
  default:  
    // Code to be executed if expression doesn't match any case  
}
```

- Conditional statements are powerful constructs that enable programmers to control the flow of execution based on specific conditions.

- They allow for decision-making and branching within the code, making programs more flexible and adaptable.

Looping Statements :

- Looping statements, also known as iteration statements, are programming constructs that allow a block of code to be repeated multiple times.
- They provide a way to execute a set of instructions repeatedly until a specific condition is met.

1. For Loop:

- The for loop is used to iterate a specific number of times.
- It consists of an initialization, condition, and increment/decrement section.
- Here's the syntax of a for loop in Java:

```
for (initialization; condition; increment/decrement) {  
    // Code to be repeated  
}
```

2. While Loop:

- The while loop repeats a block of code as long as a given condition is true.
- The loop continues until the condition evaluates to false.

- Here's the syntax of a while loop:

```
while (condition) {  
    // Code to be repeated  
}
```

3. Do-While Loop:

- The do-while loop is similar to the while loop, but it guarantees that the code block is executed at least once before checking the condition.
- It continues repeating as long as the condition remains true.
- Here's the syntax of a do-while loop:

```
do {  
    // Code to be repeated  
} while (condition);
```

4. Enhanced For Loop (Foreach Loop):

- The enhanced for loop, also known as the foreach loop, is used to iterate over arrays or collections.
- It simplifies the process of accessing elements in a collection or array.
- Here's the syntax of an enhanced for loop:

```
for (element : collection) {  
    // Code to be executed for each element  
}
```


- Looping statements are powerful tools for automating repetitive tasks and processing collections of data.
- They allow programmers to write efficient and concise code by eliminating the need for manual repetition.
- Each type of loop serves different purposes, and the choice depends on the specific requirements of the task at hand.



3. Java Console based and Web based Applications

Information

Java supports the development of both console-based applications and web-based applications.

1. Console-Based Applications:

- Console-based applications, also known as command-line applications, are programs that interact with users through a command-line interface (CLI).
- They typically read input from the user and display output in the console window.
- Java provides a robust platform for developing console-based applications, making it suitable for tasks that require text-based input/output or command-line interaction.

- Console-based applications in Java can be developed using the Java Standard Edition (Java SE) platform and its core libraries.
- Developers can utilize the java.io package to handle input/output operations, and the java.util package for various utility classes.
- By leveraging these libraries, developers can create interactive console applications for tasks such as data processing, file manipulation, or text-based games.

2. Web-Based Applications:

- Web-based applications are applications that are accessed and interacted with through web browsers.
- They typically have a client-server architecture, where the client (web browser) sends requests to the server, which processes the requests and returns responses.
- Java offers robust support for developing web-based applications through the Java Enterprise Edition (Java EE) platform.
- Java EE provides various APIs, frameworks, and tools to build web applications.
- Java Servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF) are commonly used technologies for developing web applications in Java.
- These technologies allow developers to create dynamic web pages, handle HTTP requests and

responses, manage user sessions, and interact with databases and other resources.

- Furthermore, Java EE also provides additional features such as security, transaction management, and integration with various enterprise systems.
- Application servers like Apache Tomcat, JBoss, and Oracle WebLogic provide a runtime environment for deploying and running Java web applications.
- Developing web-based applications in Java often involves working with HTML, CSS, JavaScript, and other web technologies alongside Java code.
- Java frameworks like Spring MVC and JavaServer Faces (JSF) simplify web application development by providing high-level abstractions and ready-to-use components.
- Java's versatility allows developers to choose the most suitable approach based on the requirements of their application, whether it's a console-based application or a web-based application.

Developing Console based application:

- Developing console-based applications involves creating programs that interact with users through a command-line interface (CLI).

- These applications typically read input from the user, perform operations, and display output in the console window.

1. Design and Plan:

- Define the requirements and functionalities of your application.
- Identify the tasks it needs to perform and the data it will handle.
- Plan the structure and flow of your program.

2. Choose a Programming Language:

- Select a programming language suitable for console-based application development, such as Java, C++, Python, or C#.
- Each language has its own syntax and libraries for console I/O.

3. Write the Code:

- Use a text editor or an integrated development environment (IDE) to write the code for your application.
- Start with the main entry point of your program, which typically contains the main function or method.

4. Input Handling:

- Use appropriate methods or functions to read input from the user.
- For example, in Java, you can use the Scanner class to read user input from the console.

5. Perform Operations:

- Implement the necessary logic and operations based on the requirements of your application.
- This may involve calculations, data manipulation, decision-making, or interactions with external resources.

6. Output Display:

- Use methods or functions to display output in the console.
- For example, in Java, you can use the `System.out.println()` method to print text on the console.

7. Error Handling:

- Implement error handling mechanisms to handle invalid input or unexpected situations.
- Use appropriate techniques like exception handling to catch and handle errors gracefully.

8. Testing and Debugging:

- Test your application with different input scenarios to ensure it behaves as expected.
- Debug and fix any issues or errors that arise during testing.

9. Refine and Enhance:

- Refactor your code, optimize performance, and improve the user experience.

- Consider adding additional features or functionalities based on the requirements or feedback.

10. Build and Deploy:

- Once your application is ready, compile or package it into an executable form suitable for the target platform.
- This may involve creating an executable file or a script to run the application.

11. Distribution:

- Distribute your console application to the intended users or audience.
- You can share the application's executable file or make it available for download on a website or through other distribution channels.
- Console-based applications are useful for various tasks, including data processing, text-based games, command-line tools, or utilities that don't require a graphical user interface (GUI).
- By following these steps, you can develop functional and interactive console-based applications tailored to your specific requirements.

Simple Dynamic Web Application (Without Maven)
using Eclipse :

To create a simple dynamic web application using Eclipse without using Maven, you can follow these steps:

1. Install Eclipse:

- Download and install Eclipse IDE for Java EE Developers from the Eclipse website (<https://www.eclipse.org/downloads/>).
- Choose the version suitable for your operating system.

2. Create a new Dynamic Web Project:

- Open Eclipse and select "File" > "New" > "Dynamic Web Project" from the menu.
- Enter a project name and choose a target runtime (e.g., Apache Tomcat).
- Click "Next" to proceed.

3. Configure project settings:

- In the next screen, you can modify project settings like the source folder, Java version, and deployment descriptor (web.xml).
- You can leave the default settings for now and click "Next".

4. Choose project facets:

- Select the "Dynamic Web Module" checkbox and make sure the version is appropriate for your project.
- You can also select other desired facets (e.g., Java, JavaScript) based on your requirements.

- Click "Further configuration available" to proceed.

5. Configure deployment settings:

- In this screen, you can set the context root for your application.
- By default, it uses the project name as the context root.
- Make any desired changes and click "Finish" to create the project.

6. Create a JSP file:

- Right-click on the "WebContent" folder in your project, select "New" > "JSP File".
- Enter a file name with the .jsp extension (e.g., index.jsp) and click "Finish".
- This will create a JSP file where you can write the HTML content.

7. Write HTML content:

- Open the created JSP file and write your HTML content or dynamic code using JSP expressions, scriptlets, or JSTL (JavaServer Pages Standard Tag Library) tags.

8. Configure server runtime:

- If you haven't configured the server runtime, go to the "Servers" view in Eclipse (Window > Show View > Servers).
- Right-click and select "New" > "Server".

- Choose the appropriate server (e.g., Apache Tomcat) and click "Next".
- Configure the server location and click "Finish".

9. Run the application:

- Right-click on your project, select "Run As" > "Run on Server".
- Choose the configured server runtime and click "Finish".
- Eclipse will start the server and deploy your web application.
- The application will open in the default web browser.
- You can continue developing and adding more functionality to your web application by creating additional servlets, JSPs, CSS, and JavaScript files.
- Remember to clean and rebuild the project when necessary.
- Note: Without using Maven, you will need to manually manage dependencies by adding JAR files to the project's "WebContent/WEB-INF/lib" folder and configuring the build path in Eclipse.
- This is a basic guide to get started with a dynamic web application using Eclipse.
- For more advanced features and complex projects, it is recommended to use build tools like Maven or Gradle to automate dependency management and project configurations.

4. Deployment to Tomcat and Consuming Java Applications

Information

To deploy your Java web application to Apache Tomcat and consume it, you can follow these steps:

1. Build and Package your Application:

- Ensure that your application is properly built and packaged into a WAR (Web Application Archive) file.
- The WAR file contains all the necessary files and resources required for deployment.

2. Stop Tomcat Server:

- If Tomcat server is already running, stop it before deploying your application.

- This can be done by running the shutdown.sh (Unix/Linux) or shutdown.bat (Windows) script in the Tomcat installation's "bin" directory.

3. Copy the WAR file to Tomcat:

- Copy the WAR file of your application to the Tomcat installation's "webapps" directory.
- You can either manually copy the file or use the command-line interface.

4. Start Tomcat Server:

- Start the Tomcat server by running the startup.sh (Unix/Linux) or startup.bat (Windows) script in the
- Tomcat installation's "bin" directory.

5. Access the Application:

- Once the server starts, your application will be deployed.
- You can access it using a web browser by entering the URL in the following format: <http://localhost:8080/applicationName>, where applicationName is the name of your application or the name of the WAR file without the .war extension.

6. Consuming the Application:

- To consume your Java application, you can interact with it through the web browser.
- The application will respond to the HTTP requests based on the defined servlets, JSP pages, or REST endpoints.

7. Testing and Debugging:

- Test your application thoroughly to ensure that it functions as expected.
- Use logging and debugging techniques to identify and fix any issues that arise during testing.

8. Monitoring and Maintenance:

- Once the application is deployed and being consumed, monitor its performance and troubleshoot any errors or performance issues that may arise.
- Regularly update and maintain your application as needed.
- Remember to properly configure your application for the target Tomcat server version, including any required context path or server-specific configurations.
- Additionally, ensure that any external dependencies or database connections are properly configured for your application to work correctly.
- By following these steps, you can deploy your Java web application to Apache Tomcat and consume it through a web browser or by interacting with the defined endpoints.

What is Tomcat?

- Apache Tomcat, also known as Jakarta Tomcat, is an open-source web server and servlet container developed by the Apache Software Foundation.
- It provides a Java-based environment for running Java web applications that conform to the Java Servlet, JavaServer Pages (JSP), and Java Expression Language (EL) specifications.

- Here are some key features and characteristics of Apache Tomcat:

1. Servlet Container:

- Tomcat serves as a container for executing Java Servlets, which are Java classes that handle HTTP requests and generate dynamic web content.

2. JavaServer Pages (JSP) Support:

- Tomcat supports JavaServer Pages, a technology for generating dynamic web content by embedding Java code within HTML pages.

3. Java Standard Edition (Java SE) Compatibility:

- Tomcat is designed to work with the Java SE platform, making it compatible with various Java frameworks and libraries.

4. Web Server Functionality:

- Tomcat can function as a standalone web server, serving static web content, as well as handling dynamic web applications.

5. HTTP and HTTPS Support:

- Tomcat supports the Hypertext Transfer Protocol (HTTP) and Secure HTTP (HTTPS) for secure communication between clients and the server.

6. Multiple Protocols:

- Tomcat supports various protocols, including HTTP, HTTPS, WebSocket, and AJP (Apache JServ Protocol).

7. Configuration and Deployment:

- Tomcat provides a flexible configuration system, allowing customization of server settings, such as connection pooling, thread management, and session management.
- Web applications are deployed in the form of WAR (Web Application Archive) files.

8. Scalability and Clustering:

- Tomcat supports clustering, allowing multiple instances of Tomcat to work together to handle high traffic and provide scalability and fault tolerance.

9. Management and Monitoring:

- Tomcat includes management tools and web-based administration interfaces to monitor and manage deployed applications, server configuration, and performance.

10. Open-Source and Community-Driven:

- Tomcat is an open-source project with an active community of developers, contributing to its continuous improvement and widespread adoption.

- Tomcat is widely used as a lightweight and efficient web server and servlet container for Java web applications.
- It is popular among developers due to its simplicity, compatibility, and robustness.
- Tomcat is often used in conjunction with Java frameworks like Spring and Hibernate to build and deploy enterprise-level web applications.

What is webserver?

A web server is a software application or program that serves web content over the internet or an intranet.

It receives and responds to HTTP (Hypertext Transfer Protocol) requests from clients, typically web browsers, by delivering web pages, files, or other resources requested by the client.

Application Server :

An application server, also known as an app server, is a software framework or platform that provides an environment for running and managing enterprise-level applications.

Compiling and Execution of application :

Compiling and executing an application involves several steps that vary depending on the programming language and development environment you are using.

1. Code Writing:

- Write the source code for your application using a text editor or an integrated development environment (IDE).
- The code should follow the syntax and rules of the programming language you are using.

2. Compiling:

- Compile the source code into machine-readable instructions or bytecode.
- This step is necessary for languages that require compilation, such as C, C++, Java, or C#.
- During compilation, the source code is transformed into executable code specific to the target platform.

3. Execution:

- Once the code is compiled or linked, you can execute the resulting executable file or bytecode.
 - The method of execution varies depending on the programming language and the specific runtime environment.
 - For Java, the compiled bytecode (.class files) is executed using the Java Virtual Machine (JVM) or a Java runtime environment (JRE). You use the java command followed by the name of the class containing the main method.
- Application Verification:
- Application verification, also known as software verification or software testing, is the process of evaluating a software application to ensure that it

meets specified requirements and performs as expected.

4. OOPs Concept:

Information

- Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around objects, which are instances of classes that encapsulate data and behaviour.
- OOP emphasizes modularity, reusability, and code organization by representing real-world entities as objects and enabling interactions between these objects.

1. Classes and Objects:

- Classes are blueprints or templates that define the properties (attributes) and behaviour (methods) of objects.
- Objects are instances of classes that represent specific entities or instances in the real world.

2. Encapsulation:

- Encapsulation is the principle of bundling data and methods together within a class, hiding the internal implementation details and exposing only necessary interfaces.

- It provides data protection and ensures that objects are accessed and manipulated through well-defined methods.

3. Inheritance:

- Inheritance allows classes to inherit properties and behaviour's from other classes, establishing a hierarchical relationship.
- The derived class (subclass) inherits the characteristics of the base class (superclass), enabling code reuse and specialization. It promotes code organization and supports the "is-a" relationship.

4. Polymorphism:

- Polymorphism enables objects of different classes to be treated as instances of a common superclass.
- It allows methods to be overridden in derived classes, providing different implementations while maintaining a common interface.
- Polymorphism supports flexibility, extensibility, and dynamic behavior.

5. Abstraction:

- Abstraction involves representing essential features of an object while hiding unnecessary details. It allows the creation of abstract classes

and interfaces that define a common structure or behavior for related classes.

- Abstraction promotes modularity, simplifies complex systems, and facilitates code maintenance.

Practical Includes :

■

1. Create a console based Java Application:


Here's an example of a simple console-based Java application that prompts the user for their name and greets them:

```
import java.util.Scanner;

public class ConsoleApplication {

    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user for their name
```

- 
- In this example, we use the 'Scanner' class from the 'java.util' package to read user input from the console.
 - The 'nextLine()' method is used to read a line of text entered by the user.

To create and run this console application:

- Create a new Java file named 'ConsoleApplication.java'.
- Copy the above code into the 'ConsoleApplication.java' file'.
- Save the file.
- Open a command prompt or terminal and navigate to the directory where the 'ConsoleApplication.java' file is located.
- Compile the Java file by running the following command:

```
javac ConsoleApplication.java
```

This will generate a compiled bytecode file named 'ConsoleApplication.class'.

Run the application by executing the following command:

```
java ConsoleApplication
```

The console application will start, prompt you for your name, and greet you with a personalized message.

You can modify this example to add more functionality or customize it according to your specific requirements.

2. Create a Dynamic web Application and Deploy it to Tomcat Server

To create a dynamic web application and deploy it to Tomcat Server, follow these steps:

1. Set up the Development Environment:

- Install Java Development Kit (JDK) on your computer if it is not already installed.
- Download and install Apache Tomcat Server on your computer.
- Create a Dynamic Web Project:

2. Open Eclipse IDE (Integrated Development Environment).

- Go to "File" -> "New" -> "Dynamic Web Project" to create a new web project.
- Provide a name for your project and select a target runtime as Apache Tomcat.
- Click "Finish" to create the project.

3. Design the Web Pages and Code:

- In the Eclipse Project Explorer, navigate to the "WebContent" directory.
- Create HTML, CSS, and JavaScript files to design the web pages.
- Create Java Servlets or JSP (JavaServer Pages) files to handle server-side logic and dynamic content generation.
- Write the necessary code and design the web pages according to your application's requirements.

4. Configure Web Deployment Descriptor (web.xml):

- In the "WebContent/WEB-INF" directory, locate the "web.xml" file.
- Configure the servlet mappings, URL patterns, and any other required settings in the web.xml file.

5. Build the Project:

- Right-click on the project in Eclipse and select "Run As" -> "Run on Server".
- Select the Apache Tomcat server from the list and click "Finish".

- Eclipse will build the project and deploy it to the Tomcat server.

6. Test and Debug:

- Open a web browser and enter the URL "http://localhost:8080/your-project-name" to access your web application.
- Test your web application's functionality and perform any necessary debugging or troubleshooting.

7. Package the Web Application:

- In Eclipse, right-click on the project and select "Export" -> "WAR File".
- Choose a location to save the WAR file and click "Finish".
- The WAR file contains your web application and can be deployed to other instances of Tomcat.

8. Deploy the Web Application to Tomcat Server:

- Copy the WAR file to the "webapps" directory in the Tomcat installation directory.
- Start or restart the Tomcat server.
- Tomcat will automatically deploy the web application by extracting the contents of the WAR file.
- Access your web application by entering the URL "http://localhost:8080/your-project-name" in a web browser.

- Remember to make any necessary configurations specific to your web application, such as database connections or additional libraries.
- This is a basic outline of the process to create and deploy a dynamic web application to Tomcat Server.
- You can customize and enhance your application based on your specific requirements and add more features as needed.

