Java -Curriculum

1. Java basics

1.1 Intoduction & History of Java

- Java is a powerful, object-oriented programming language designed for cross-platform compatibility, enabling applications to run on any device with a Java Virtual Machine (JVM).
- Renowned for its security and scalability, Java is widely used in web development, mobile apps (Android), enterprise software, and large-scale systems.
- Java was developed by Sun Microsystems in the early 1990s, led by James Gosling, with the aim of creating a platform-independent programming language.
- Released in 1995, Java quickly gained popularity for its "write once, run anywhere" capability, becoming a standard for web and enterprise applications.

1.2 Types of Java

- Java Standard Edition (Java SE)
- Java Enterprise Edition (Java EE)
- Java Micro Edition (Java ME)
- Java Embedded

1.3 Terminologies of Java

- JDK - Java Development Kit
- JRE - Java Runtime Environment
- JVM - Java Virtual Machine

1.4 Categories of Software

- System Software: OS, Drivers, Interpreters, Assemblers and Compilers
- Application Software: Word, Notepad

1.5 Java Technologies Based on Platform

- Desktop/Laptop/others PC - J2SE, Mobile/Android/IoT - J2ME. Web - J2EE

1.6 Tools

- Java Netbeans
- Eclipse
- Online gdb - www.onlinegdb.com

1.7 Basic Syntax

Class Declaration

- All Java programs are organized into classes, with each class containing methods and variables, and the main method serving as the program's entry point.

Method Structure

- Methods in Java define actions and follow the syntax returnType methodName(parameters), with the main method (public static void main(String[] args)) required in every Java application.

Statements and Semicolon

- Each statement ends with a semicolon (;), and Java is case-sensitive, distinguishing between uppercase and lowercase characters.

Curly Braces for Blocks

- Curly braces {} enclose code blocks, such as those in methods, classes, and control structures, helping to define the scope and structure of the code.

Java -Curriculum

1.8 DataTypes

Overview

- DataType is a fundamental block of any language

Purpose

- Store Values
- Perform operations on the values stored

Types of Datatypes

- Primitive Datatype - Boolean, Numeric, Character, Integral, Integer
- Non-Primitive Datatype - Aaryas, Classes, Strings, Inerfaces

1.9 Variables

Overview

- Variables in Java are containers for storing data values, defined with a specific data type to indicate the kind of value they hold (e.g., int, String).

Purpose

- Variables in Java store and manage data within a program, allowing for data to be accessed, modified, and reused throughout the code.
- Each variable is associated with a data type, defining the kind of values it can hold, ensuring type safety and efficient memory usage.

Types of Variables

- Local Variables
- Instance Variables
- Static Variables
- Final Variables

1.10 Conditions

If-Else Statements

- Java's if and else statements allow you to execute code based on conditions, enabling decision-making within a program for flexible control flow.

Nested If Statements

- Conditions can be nested within each other to check multiple criteria in complex scenarios, where each if checks a specific condition depending on previous outcomes.

Switch Statement

- The switch statement simplifies multi-way branching by matching a variable's value against multiple cases, often more readable than multiple if-else statements.

Ternary Operator

- The ternary operator (condition ? trueValue : falseValue) provides a concise way to perform conditional assignments, ideal for simple conditions in a single line.

Java -Curriculum

1.11 Functions

Overview

- In Java, functions (also called methods) are blocks of code that perform specific tasks and are defined within a class, helping to organize and reuse code effectively.

Return Type and Parameters

- Each function has a return type (e.g., int, void) and can accept parameters, which allow it to process different data inputs and return a result.

Calling Functions

- Functions are called by their name, allowing code to execute the specified task; they can be called from within the same class or from other classes if marked as public.

Method Overloading

- Java supports method overloading, allowing multiple methods in the same class to share the same name but with different parameter lists, enabling flexible use of functions.

1.12 Loops

Overview

- Java loops (for, while, do-while) enable code to repeat a set of instructions multiple times, reducing redundancy and improving efficiency in handling repetitive tasks.

Types

- **For Loop:** The for loop is typically used when the number of iterations is known in advance, with a clear initialization, condition, and increment/decrement step.
- **While Loop:** The while loop continues executing as long as its condition remains true, making it useful when the number of iterations is not predetermined.
- **Do-While Loop:** Similar to while, the do-while loop guarantees at least one execution of the loop body, as the condition is evaluated after the loop executes.

1.13 DataStructures

Data Structures Overview

- Java provides various built-in data structures, including arrays, lists, sets, and maps, to efficiently store, organize, and manipulate data.

Arrays

- Fixed-size structures that hold elements of the same type, allowing quick access to elements via index, making them suitable for simple data collections.

Collections Framework

- Java's Collections Framework offers dynamic data structures like ArrayList, LinkedList, HashSet, and HashMap, which provide more flexibility and functionality for managing groups of objects.

Stack and Queue

- Specialized data structures, such as Stack (Last In First Out) and Queue (First In First Out), facilitate specific data handling requirements, useful in various algorithms and applications.

Java -Curriculum

2. Java Advanced

2.1 Object Oriented Programming (OOPs)

- Data Driven rather than operation driven
- Functions of OOPs
- Characteristics of OOPs
- Inheritance
- Polymorphism
- Containership
- Reusability

2.2 Exception Handling

Overview

- Exception are the errors occur during runtime

Types

- Memory Out of Bound, Inaccessible File
- Division by Zero, Stack Overflow
- Arithmatic Overflow, Unable to Connect Server
- Exception on Arrays

2.3 File Handling

Overview

- Java provides a robust set of classes and methods for file handling, allowing developers to read from and write to files efficiently using the java.io and java.nio.file packages.

Input and Output Streams

- File handling in Java utilizes input and output streams, such as FileInputStream and FileOutputStream, for reading and writing binary data, while BufferedReader and PrintWriter are used for character data.

File Class

- The File class represents a file or directory path and provides methods for file manipulation, such as creating, deleting, and checking file properties (e.g., existence and size).

2.4 Packages

Overview

- In Java, packages are used to group related classes and interfaces, providing a namespace to avoid name conflicts and organize code logically.

Built-in Packages

- Java includes several built-in packages, such as java.lang (automatically imported), java.util (utility classes like collections), and java.io (input and output operations), which provide essential functionalities.

Creating Custom Packages

- Developers can create their own packages using the package keyword, allowing for better organization of project files and enhancing code reusability.

Importing Packages

- The import statement is used to access classes from other packages, enabling the use of external functionalities without the need to fully qualify class names.

Java -Curriculum

2.5 Java Vitural Machine (JVM)

Overview

- The Java Virtual Machine (JVM) is an abstract computing machine that enables Java applications to run on any device or operating system by providing a platform-independent execution environment.

Bytecode Execution

- The JVM interprets or compiles Java bytecode, generated from Java source code, into machine code specific to the host operating system, allowing for portability across different platforms.

Memory Management

- The JVM handles memory allocation and garbage collection, automatically reclaiming memory that is no longer in use, which helps manage system resources efficiently.

JVM Components

- Key components of the JVM include the class loader (for loading classes), execution engine (for executing bytecode), and garbage collector (for memory management), working together to ensure smooth program execution.

2.6 Threads

Overview

- In Java, a thread is a lightweight process that allows concurrent execution of tasks within a program, enabling efficient use of CPU resources and improving application performance.

Creating Threads

- Threads can be created by extending the Thread class or implementing the Runnable interface, providing flexibility in defining thread behavior and sharing resources.

Thread States

- A thread in Java can exist in several states, including New, Runnable, Blocked, Waiting, and Terminated, allowing for complex lifecycle management and synchronization.

Synchronization

- To prevent data inconsistencies in multi-threaded environments, Java provides synchronization mechanisms, such as synchronized methods and blocks, ensuring that only one thread can access a critical section of code at a time.

2.7 Garbage

Overview

- In Java, garbage collection is an automatic memory management process that identifies and discards objects that are no longer referenced, freeing up memory resources for future use.

Generational Garbage Collection

- Java's garbage collection is based on a generational approach, which divides objects into different generations (Young, Old, and Permanent) to optimize memory management and improve performance.

Automatic Process

- The Java Virtual Machine (JVM) performs garbage collection automatically, allowing developers to focus on application logic without worrying about manual memory management or memory leaks.

Garbage Collection Algorithms

- Java employs various algorithms for garbage collection, such as Mark-and-Sweep, Copying, and G1 (Garbage-First) collector, each with different trade-offs regarding pause times and throughput.

Java -Curriculum

2.8 Generics

Overview

- Generics allow developers to create classes and methods that can operate on any data type while providing compile-time type safety.

Type Parameters

- By using type parameters (e.g., <T>, <E>), generics enable the creation of flexible and reusable code that avoids the need for casting.

Generic Collections

- Java collections like ArrayList<T> utilize generics to ensure that only specific types of objects can be added, enhancing type safety.

Wildcards

- Generics support wildcards (e.g., ?, ? extends T) to allow for more flexible method parameters and facilitate operations on multiple types.

2.9 Multithreading in Java

Overview

- Java supports multithreading, allowing multiple threads to run concurrently, enhancing application performance and resource utilization.

Thread Creation

- Threads can be created by extending the Thread class or implementing the Runnable interface, enabling flexible execution of concurrent tasks.

Synchronization

- Java provides mechanisms like synchronized methods and blocks to ensure thread safety when accessing shared resources, preventing data inconsistency.

Thread Lifecycle

- Threads in Java can exist in various states, including New, Runnable, Blocked, Waiting, and Terminated, allowing for effective management of thread execution.

2.10 Java Interfaces AWT

Java AWT (Abstract Window Toolkit)

- API to Delveope Graphical User Interface - Windows Based Apps
- Platform Dependent
- Specific for Specific Operating System
- Heavy Weight or Compute Intensive
- No Plug and Play support
- Less Component
- Not MVC (Model View Controller)

Java Swing

- Platform Independent
- Components are Light Weight
- Plug and Play support
- More Component
- Swing MVC (Model View Controller)

Java -Curriculum

2.11 Networking and Sockets

Components

- Clients (LAN), Servers, Modems
- Hubs, Swtiches, Access Points

Server

- Print Servers, Web Servers
- Application Servers, File Servers

Gateway Machines

- Hubs
- Routers
- Routers of other LANs

Network Models

- TCP/IP
- OSI

Protocols

- HTTP, SMTP
- POP3, FTP, SSH

Socket Models

- Connection-Oriented (TCP) Sockets
- Connectionless (UDP) Sockets
- Non-blocking (NIO) Sockets
- Secure Sockets (SSL/TLS)

3. Java Frameworks

3.1 Build Tools

Overview

- Piece of Code/Program used for automating the process of executing couple of applicatons

Responisibilites of Build tools

- Compiles Source Code into Byte Code
- Dependency Management - Download & Maintain 3rd Party Plugins
- Automated Test - Executes & Reports Bugs
- Deployment Package - WAR/JAR - Server

Tools

- Gradle
- Apache Maven
- Ant Design

Java -Curriculum

3.2 Apache Maven

Overview

- It is a Java build tool widely available open-source, which was developed in 2004 as an extension to Apache Ant.
- It is a Project Object Model which is based on Extensible Markup Language (XML)

Aspects Performed in Maven

- Build, Documentation Management, Reporting
- Dependency, SCMs, Distribution

Steps Involved in performing Aspects in Mavem

- Source, Resources
- Tests, Byte Code, Java Archive (JAR)

Features

- Easy Setup
- Easy Dependency Management
- Large Libraries and Community Support
- Model Based Builds
- Highly Compactible
- Easy Reporting

3.3 Gradle

Overview

- Gradle is a flexible, open-source build automation tool which was developed in 2000 to overcome the drawback of Ant.
- It is a Project Object Model which is based on Groovy

Features

- It is available via Domain Specific Language (DSL), based on groovy language
- It provides a declarative language
- It supports Java, Groovy, Open Service Gateway Initiative (OSGi)
- API Support
- Structuring
- Multi Project Support
- Migration Ease
- It uses groovy to build API

3.4 Logging

Overview

- Logging in Java allows applications to record runtime information, errors, and events, helping with debugging, monitoring, and maintaining code.

Frameworks

- Common Java logging frameworks include java.util.logging (built-in), Log4j, and SLF4J, each providing tools for log formatting, filtering, and output.

Log Levels

- Logging uses levels (e.g., INFO, DEBUG, WARN, ERROR) to categorize messages, allowing developers to filter logs based on the importance of events.

Output Options

- Logs can be directed to various outputs such as console, files, or remote servers, making it easier to track application behavior in different environments.

Java -Curriculum

3.5 Log4j 2

Overview

- Log4j 2 is an enhanced logging framework in Java that provides powerful, flexible logging capabilities, succeeding the original Log4j with improved performance and reliability.

Asynchronous Logging

- Log4j 2 supports asynchronous logging, reducing performance impact by handling logging operations in a separate thread, which is ideal for high-throughput applications.

Configuration Options

- It offers multiple configuration formats (XML, JSON, YAML, and properties), allowing easy customization of logging behavior and outputs.

Plugin-Based Architecture

- Log4j 2's plugin system enables customization with additional components for filtering, formatting, and appending logs to various destinations, such as files, databases, or remote servers.

3.6 Frameworks (Advanced)

Overview

- Java frameworks are reusable, structured code libraries that simplify development by providing standardized tools and patterns.

Popular Java Frameworks

- Common frameworks include Spring (for enterprise applications), Hibernate (for database ORM), and Apache Struts (for web applications), each tailored to specific needs.

Time and Effort Savings

- Frameworks speed up development by offering pre-built modules, reducing boilerplate code, and improving consistency across projects.

Enhanced Application Structure

- Frameworks enforce best practices and provide a clear structure, making applications easier to maintain, test, and scale

3.7 Object Relational Mapping (ORM)

Overview

- Object-Relational Mapping (ORM) in Java is a technique that simplifies data handling by mapping database tables to Java objects, allowing developers to work with databases using object-oriented concepts.

Popular ORM Tools

- Hibernate and JPA (Java Persistence API) are widely used ORM frameworks in Java, automating SQL generation and database interactions.

Improved Productivity

- ORM reduces the need for complex SQL queries, letting developers focus on business logic by automatically handling data persistence.

Database Independence

- ORM allows applications to be more flexible and database-agnostic, enabling easier transitions between database systems without major code changes.

Java -Curriculum

Hibernate

- Open-source ORM framework for Java.
- Provides advanced caching and performance optimizations.
- Supports complex mappings for relational data.
- Allows flexible configuration via annotations and XML.

Java Persistence API (JPA)

- Standardized Java API for ORM.
- Supports annotations for entity mapping.
- Provides entity lifecycle management.
- Enables easy switching between ORM providers.

Spring Data JPA

- Simplifies data access layers in Spring applications.
- Built on top of JPA for streamlined ORM.
- Offers repository interfaces for CRUD operations.
- Provides dynamic query generation.

3.8 JDBC

Java database connectivity

- JDBC is an API that allows Java applications to interact with databases, executing SQL queries and retrieving data.

Database independence

- JDBC provides a common interface for various databases, enabling Java applications to connect to different databases seamlessly.

Query execution

- It allows developers to execute SQL statements for tasks like data insertion, updates, and retrieval through Statement and PreparedStatement objects.

Result handling

- JDBC returns results in ResultSet objects, allowing efficient processing of query results within Java applications.

JDBC Template

- Simplifies database operations in Spring applications.
- Provides automatic exception translation.
- Manages database connections and resource cleanup.
- Supports batch processing and query execution.

JDBC 3

- Introduces the DataSource interface for better connection management.
- Adds support for batch updates and batch processing.
- Enhances PreparedStatement and CallableStatement features.
- Provides improved transaction management with connection pooling.

Java -Curriculum

3.9 Spring Core

- Foundation of Spring Framework
- Spring Core provides the essential features for building Java applications, including dependency injection (DI) and inversion of control (IoC).

Bean management

- It manages beans, or objects, in a container, allowing automatic creation and wiring of components for loose coupling.

Configuration flexibility

- Supports both XML and annotation-based configuration for defining application components and their dependencies.

ApplicationContext

- Central to Spring Core, ApplicationContext provides a way to access the bean container and manage the lifecycle of beans in the application.

Spring Boot

- Framework for building standalone, production-grade Spring applications.
- Simplifies configuration with embedded servers like Tomcat or Jetty.
- Supports auto-configuration and a wide range of pre-built templates.
- Enhances development speed with minimal setup and dependencies.

Spring MVC

- A framework for building web applications in Java using the Model-View-Controller design pattern.
- Supports request handling through controllers and views rendered by JSP, Thymeleaf, etc.
- Provides flexible routing with annotations like @RequestMapping and @GetMapping.
- Integrates with Spring's IOC (Inversion Of Control) containter for Dependency Injection and validation mechanisms for better scalability and maintainability.

Spring Data

- Simplifies database access and integrates with various data stores (relational, NoSQL).
- Provides repositories to perform CRUD operations without writing boilerplate code.
- Supports JPA, MongoDB, Redis, Cassandra, and other data stores.
- Enables easy pagination, sorting, and dynamic query generation with minimal configuration.

Spring Security

- Provides authentication and authorization for Java applications.
- Supports various authentication mechanisms (e.g., LDAP, OAuth).
- Enables method-level security and access control.
- Integrates seamlessly with Spring applications for secure web services.

Java -Curriculum

CAPSTONE PROJECTS

1 **Calculator Built in Java**

- This project involves the development of a calculator application that replicates the functionalities of the Windows Calculator, using Java and Swing for the user interface.
- The goal is to create a feature-rich and user-friendly application that supports basic arithmetic operations, scientific calculations, and memory functions.
- The project emphasizes implementing a responsive and intuitive interface using Swing components, while ensuring accurate and efficient computational performance.
- Through this project, users will gain insights into the design and development of GUI-based applications in Java.
- It also serves as a practical exercise in handling events and managing user inputs effectively.

2 **Build a Dynamic website using Java Servlets and JDBC**

- The project aims to develop a dynamic website using Java Servlets and JDBC.
- The website will showcase the user's skills, projects, and professional background, while also providing an interactive and engaging user experience.
- This project focuses on integrating backend technologies like Servlets for handling requests and JDBC for connecting to a database, ensuring smooth data retrieval and storage.
- The development process highlights the use of MVC architecture, form handling, and session management.
- The final product will serve as a functional demonstration of web application development using Java technologies, showcasing the user's technical proficiency.

3 **Student Course Management System**

- The Student Course Management System is a web-based application developed using Java, Thymeleaf, and MySQL.
- It facilitates the efficient management of student records, course details, and enrollments.
- The system employs the MVC (Model-View-Controller) architecture, with Thymeleaf providing dynamic rendering of web pages, enabling seamless integration of backend data with the user interface.
- MySQL serves as the relational database, ensuring robust data storage and retrieval for students, courses, and enrollments.
- This project offers a practical introduction to Java web development, combining dynamic web content, database management, and user interaction in a cohesive system.

LIVE PROJECT

1 **Develop a CRM for Employee Managment using Java Spring**

- This project involves the development of a Customer Relationship Management (CRM) system tailored for managing employee information and interactions, utilizing the Java Spring framework.
- The CRM system will streamline various employee-related tasks, such as attendance tracking, performance monitoring, task assignments, and communication.
- By leveraging Spring's robust features, the project ensures efficient backend management, secure data handling, and seamless integration with other modules.
- The system is designed to improve workflow efficiency and provide real-time insights into employee activities.
- Additionally, it will include a user-friendly interface for easy navigation and interaction.