Session 1- Facilitation Guide

Introduction to Java

Index

For (2 hrs) ILT

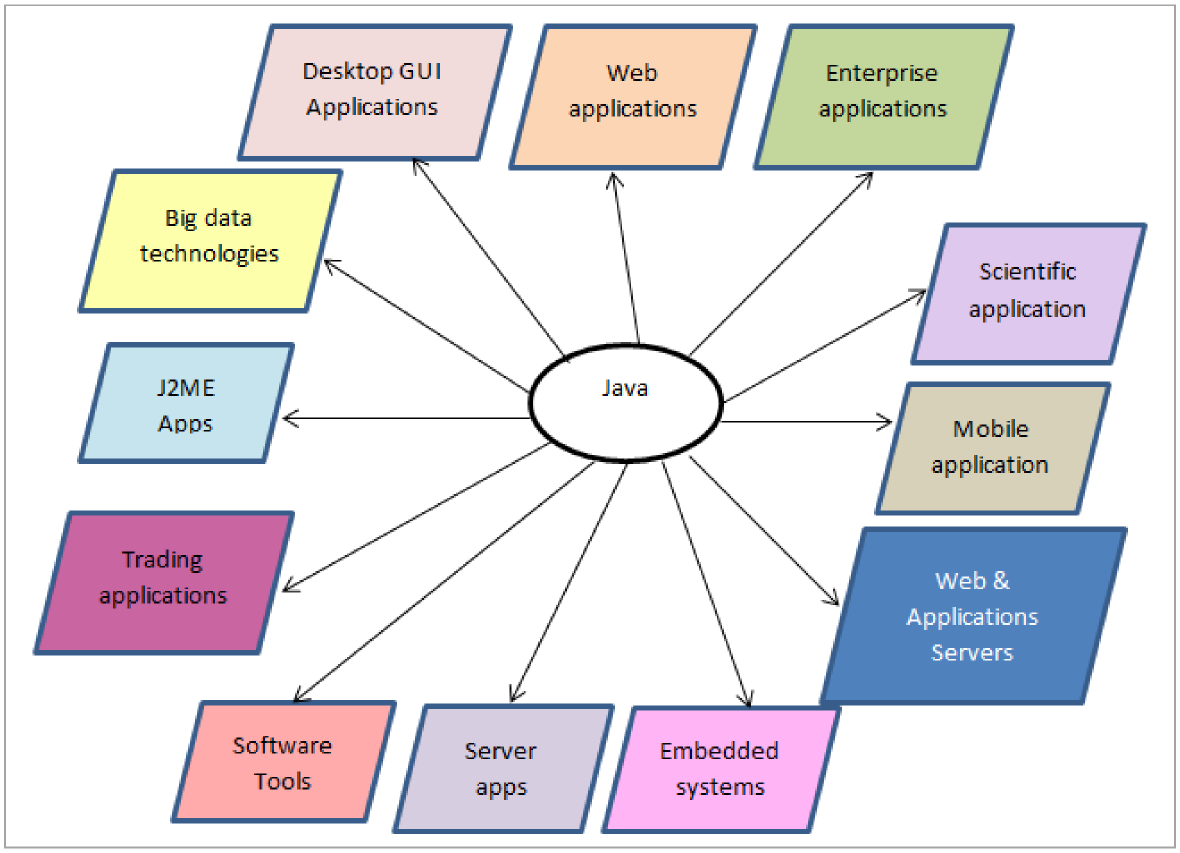
I. Trainer & Students' Introductory Session - (15 Mins)

II. LMS & Diya App Orientation - (30 Mins)

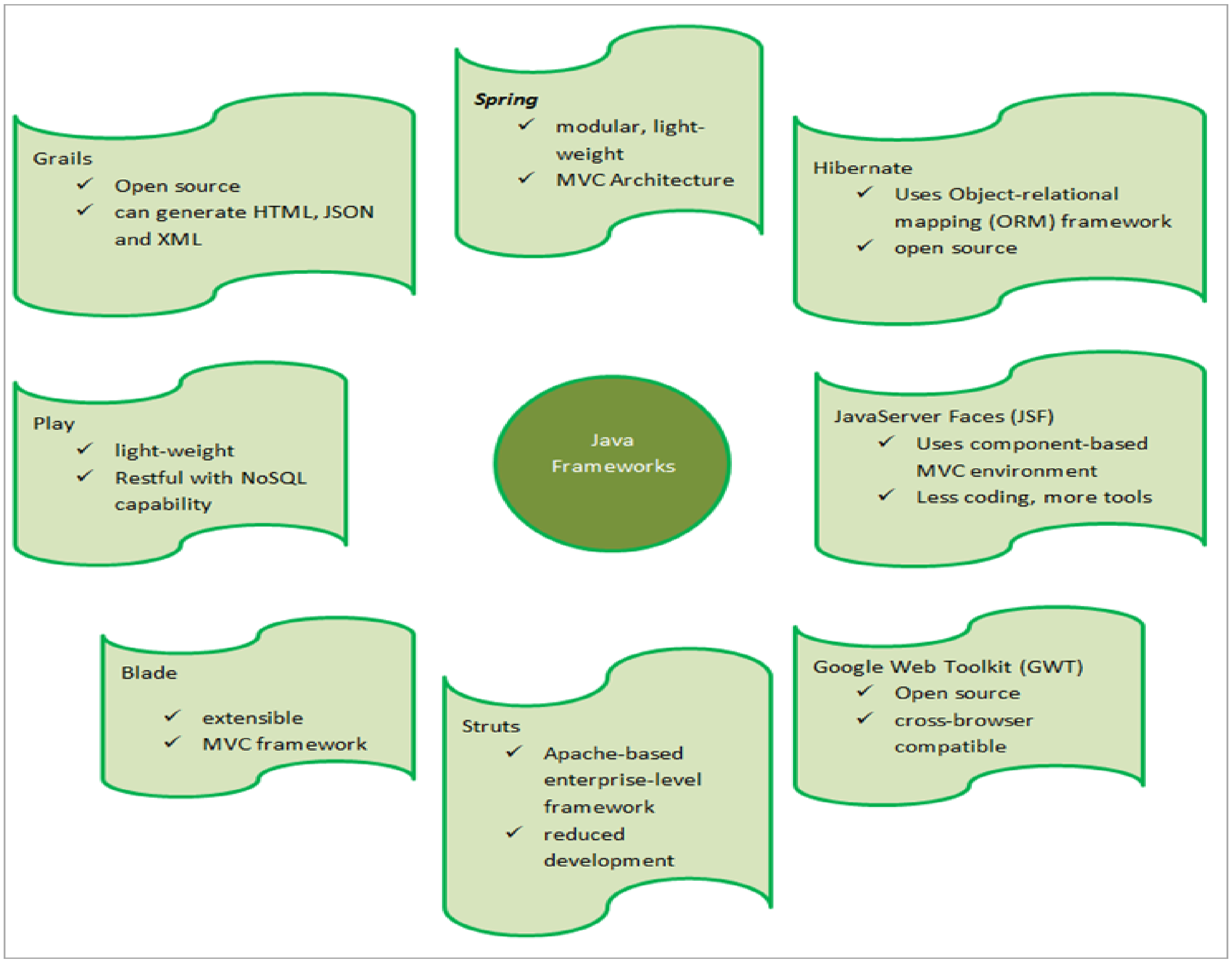
III. The genesis of Java

Why Java?

https://www.softwaretestinghelp.com/real-world-applications-of-java/



Most popular Java frameworks


Spring
✓ modular, light-weight
✓ MVC Architecture
Grails
✓ Open source
✓ can generate HTML, JSON and XML
Hibernate
✓ Uses Object-relational mapping (ORM) framework
✓ open source
Play
✓ light-weight
✓ Restful with NoSQL capability
Java Frameworks
JavaServer Faces (JSF)
✓ Uses component-based MVC environment
✓ Less coding, more tools
Blade
✓ extensible
✓ MVC framework
Struts
✓ Apache-based enterprise-level framework
✓ reduced development
Google Web Toolkit (GWT)
✓ Open source
✓ cross-browser compatible

Who invented Java?

The Java programming language was developed by **James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan** at **Sun Microsystems** in the early 1990s. The project was initially called **"Oak"** and was intended to be used in set-top boxes and other consumer electronics devices.

However, the team soon realized that Oak was not well suited for this purpose and shifted their focus to creating a general-purpose programming language that could run on any platform. The language was renamed Java and was released to the public in 1995.

One of the key features of Java is its **"write once, run anywhere" (WORA)** capability, which means that Java code can be compiled into platform-independent bytecode that can run on any machine with a **Java Virtual Machine (JVM)**. This has made Java a popular choice for developing cross-platform applications.

Since its release, Java has become one of the most widely used programming languages in the world, with applications ranging from desktop software to web applications, mobile apps, and even enterprise systems.

IV. Understanding Java Architecture

Java is an Object oriented programming language and prior to it the most popular object oriented programming language was C++. There were several procedural programming languages like C, Cobol, Pascal and Basic, which were also popular at that time.

The drawback of procedural programming is

- Excessive use of Global Variable without any access control
- Code is not easily reusable
- Lack of real life programming model

So C++ was getting attention from the developer community as a right choice for Object oriented language but it was not easily portable to all common operating systems. At that time Java came as a platform independent programming language with the tagline “Write once, run anywhere”.

Some of the key differences between procedural programming and object-oriented programming:

Data and Functions: In procedural programming, the data and functions are separate, whereas in object-oriented programming, the data and functions are encapsulated within objects.

Modularity: Procedural programming uses functions to break down a program into smaller, more manageable pieces. In OOP, objects encapsulate data and functions, providing a more modular approach to programming.

Reusability: OOP promotes reusability through inheritance and polymorphism, whereas in procedural programming, functions are often written for specific purposes and may not be reusable.

Abstraction: OOP provides abstraction, which allows programmers to focus on the essential features of an object and ignore the irrelevant details. In procedural programming, abstraction is limited.

Complexity: OOP is better suited for complex systems, as it provides a more modular, scalable approach to programming. Procedural programming can become difficult to manage as the program grows larger.

Overall, while procedural programming focuses on functions and their relationships, object-oriented programming focuses on objects and their relationships. OOP provides a more modular and reusable approach to programming, making it better suited for complex systems.

Java architecture refers to the overall structure and organization of the Java platform, including its programming language, runtime environment, and class libraries. The Java platform is designed to be portable and platform-independent, allowing Java applications to run on any platform that supports the Java Virtual Machine (JVM).

At a high level, the Java architecture consists of four main components:

Java Development Kit (JDK): The JDK is a software development kit that includes everything needed to develop and run Java applications, including the Java compiler, Java runtime environment, and a set of libraries and tools.

Java Virtual Machine (JVM): The JVM is an abstract machine that provides the runtime environment for Java programs. It is responsible for interpreting the compiled Java bytecode and executing it on the underlying hardware.

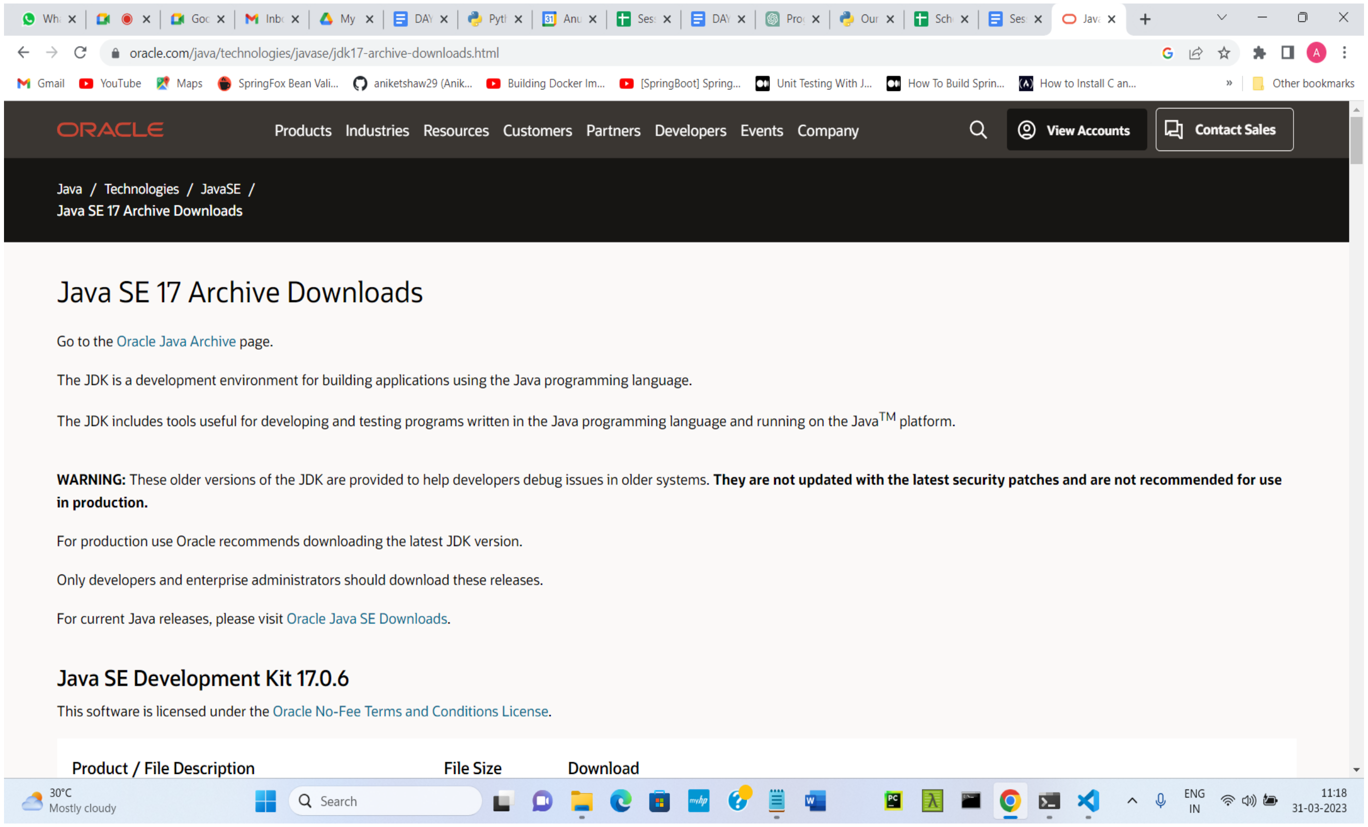
Java Class Libraries: The Java class libraries are a set of pre-written code modules that provide a wide range of functionality to Java programs, including input/output operations, networking, user interface development, and more.

Java Runtime Environment (JRE): The JRE is the subset of the JDK that is required to run Java applications. It includes the JVM and the class libraries needed for running Java programs. Please note that the JRE is enough to run any Java program if we are not interested in compilation of Java code. JRE doesn't contain any development tools such as Java compiler, debugger.

The Java architecture is designed to be highly modular and extensible, allowing developers to write custom libraries and tools that can be used alongside the standard Java libraries. This modular design also enables developers to build complex applications using a combination of pre-existing libraries and custom code, making Java a popular choice for enterprise-level software development.

V. Downloading Installation and setting up path for JDK 17(LTS)

Download from https://www.oracle.com/in/java/technologies/downloads/#jdk17-windows



Java SE 17 Archive Downloads

Go to the Oracle Java Archive page.

The JDK is a development environment for building applications using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java^{TM} platform.

WARNING: These older versions of the JDK are provided to help developers debug issues in older systems. **They are not updated with the latest security patches and are not recommended for use in production.**

For production use Oracle recommends downloading the latest JDK version.

Only developers and enterprise administrators should download these releases.

For current Java releases, please visit Oracle Java SE Downloads.

Java SE Development Kit 17.0.6

This software is licensed under the Oracle No-Fee Terms and Conditions License.

Product / File Description	File Size	Download

VI. Explanation of Java related commands:-

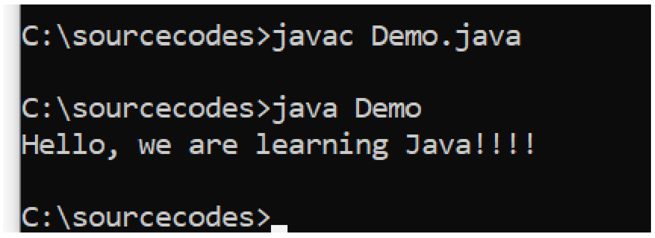
javac- for compilation
java - for interpretation
javap - for disassembly
javadoc - for documentation
jar - for creating Java archives
jdb - Java debugging

Type a simple Java class, which will look like the following:-

```
public class Demo {
  public static void main(String[] args) {
      System.out.println("Hello, we are learning Java!!!!");
   }
}
```

The output will be:

```
Hello, we are learning Java!!!!
```



```
C:\sourcecodes>javac Demo.java

C:\sourcecodes>java Demo
Hello, we are learning Java!!!!

C:\sourcecodes>
```

1. Save the file as **Demo.java** by creating a folder in any drive of your choice.
2. Open a command prompt by running the command “**cmd**”
3. Move to that folder from on command prompt
4. Run the command **javac Demo.java**
5. It will create the **Demo.class** which is the **BYTECODE**
6. Then run the command **java Demo** ————> It will run the program and output will appear on the console.

Also explain the different DOS commands to students like md, cd, dir, copy con etc to make the students comfortable with running commands from the command prompt.

VII. Garbage Collection - introduction:

Overview of Memory Management and Garbage Collection
Garbage collection is an automatic memory management mechanism in programming languages, including Java. It is designed to automatically reclaim memory that is no longer in use by the program, freeing up resources and preventing memory leaks.

However, garbage collection does come with some overhead. The garbage collector needs to perform its tasks periodically, which can introduce some pauses or overhead in the execution of the program. Careful consideration of memory usage and performance tuning may be necessary in certain situations to optimize the garbage collection process.

VIII. Overview of Java's Garbage Collector

Overall, garbage collection is an essential feature of modern programming languages, providing automatic memory management and freeing developers from the burden of manual memory deallocation, leading to more robust and secure applications.

One of the major advantages of Java is that in Java, garbage collection is an automatic memory management feature provided by the Java Virtual Machine (JVM). It ensures that objects that are no longer referenced by the program are automatically identified and reclaimed, freeing up memory resources. Unreachable objects are considered "garbage" and are eligible for collection. Although there is a Java statement (System.gc()) that can call a garbage collection function, it will just send a request to JVM for running garbage collection. Actual decision on when to run the garbage collection is not within the control of the programmer. However, programmers should write their code in such a way that more items will be eligible for garbage collection.

After the 2 hrs ILT, the student has to do a 1 hour live lab. Please refer to the Session 1 Lab doc in the LMS.