**Lecture 1**

**JAVA 8 FEATURES**

Java 8 provides following features for Java Programming:

* Functional interfaces,
* Lambda expressions,
* Stream API,
* Method references,
* Default methods,
* Base64 Encode Decode,
* Static methods in interface,
* Optional class,
* Collectors class,
* ForEach() method,
* Nashorn JavaScript Engine,
* Parallel Array Sorting,
* Type and Repeating Annotations,­
* IO Enhancements,
* Concurrency Enhancements,
* JDBC Enhancements etc.

Java 8 is still a Long-Term Support (LTS) version. Oracle provides updates for Oracle JDK 8 until at least December 2030 for commercial use. However, for non-commercial use, updates are no longer provided by Oracle, but other vendors such as AdoptOpenJDK, Amazon Corretto, and others continue to provide updates and support for Java 8.

If you are using Java 17, Java 8 features are still present, which is why they remain important.

Java 8 stands out as a highly anticipated release in the evolution of the Java programming language, introducing more significant features than any previous version. It marked a major milestone with the inclusion of numerous essential features. Released by Oracle on March 18, 2014.

Since Java 8, Java provides support for functional programming and has reduced the amount of code needed.

**Lecture 2**

1. Default Method

In Java 8, the concept of default methods (also known as defender methods or virtual extension methods) was introduced in interfaces.

A default method is a method in an interface that has a body. Default methods were introduced to support backward compatibility and allow interfaces to provide method implementations

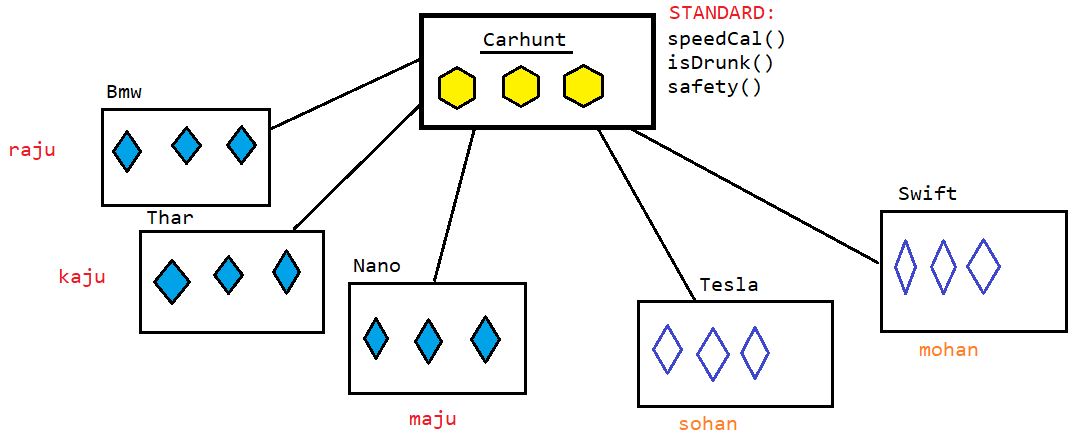
The default keyword in Java is not an access modifier. It is used specifically in the context of interfaces to define default methods.

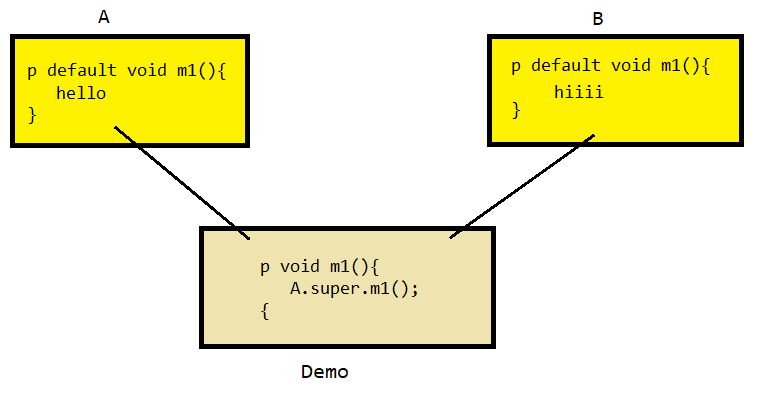
If you don’t explicitly specify an access modifier for a method in an interface, it is implicitly public.

Java 7 : abstract method only

Java 8: abstract, static, default methods

Default methods make it possible to extend existing interfaces with new functionality while still supporting legacy classes that were written before the interface was updated. This way, the older code remains functional and compatible with the new version of the interface.





**Functional interface**

1. What is Functional Interface?

* An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.
* Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces.

Some Built-in Java Functional Interfaces

* Since Java SE 1.8 onwards, there are many interfaces that are converted into functional interfaces. All these interfaces are annotated with @FunctionalInterface. These interfaces are as follows –
* Runnable –> This interface only contains the run() method.
* Comparable –> This interface only contains the compareTo() method.
* ActionListener –> This interface only contains the actionPerformed() method.
* Callable –> This interface only contains the call() method.

Java SE 8 included four main kinds of functional interfaces which can be applied in multiple situations as mentioned below:

* Consumer
* Predicate
* Function
* Supplier

2.Why Functional Interface?

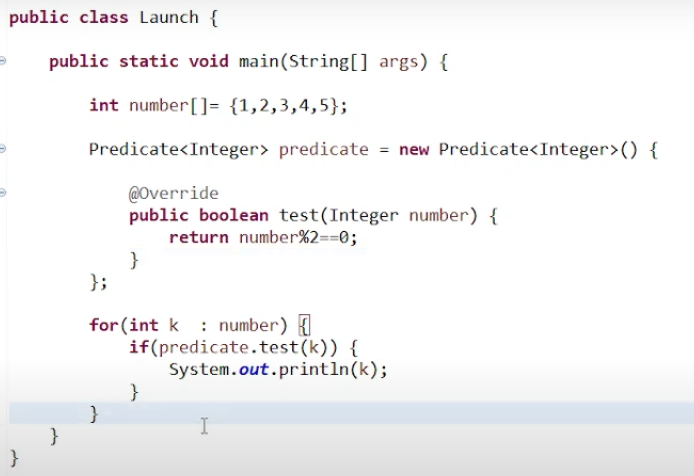
* Functional interfaces in Java are a fundamental concept that was introduced with Java 8 to support functional programming features.
* To fully support functional programming in Java, alongside functional interfaces, several other concepts and features are necessary like Lambda expression and stream API.

NOTE: Java 8 introduces several features that enable functional programming, but it's important to note that Java is not a purely functional programming language. Java 8 incorporates functional programming elements, allowing developers to write code in a functional style, but it still remains an object-oriented language at its core.

3.What is Functional Programming?

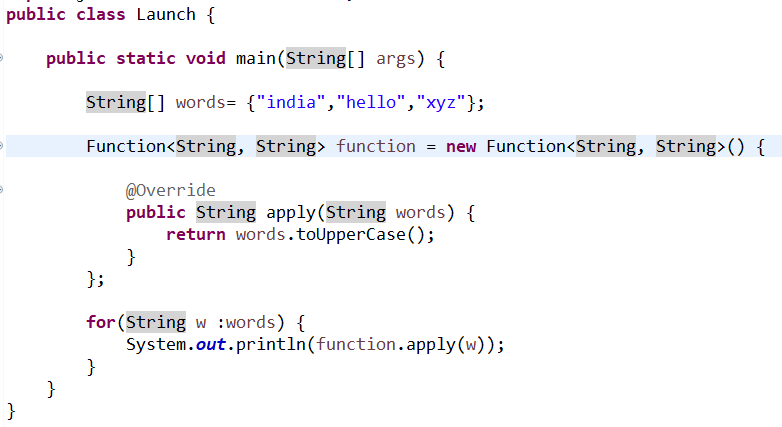
* It is a declarative style of programming rather than imperative.
* In functional programming, functions are to be considered as first-class citizens.
* The main goal of this programming style is to make the code more concise.
* Imperative programming is a style of programming where the code explicitly describes the steps that the computer must take to achieve a desired result. It focuses on how the result is to be achieved.
* Declarative programming is a style of programming where the code describes what the result should be, rather than the steps to achieve it. The focus is on the logic of the computation rather than the control flow
* Ability to treat functions as values
* Ability to pass a function as arguments
* Ability to return a function from another function
* Predicate

The Predicate<T> interface represents a boolean-valued function of one argument. It's used to test if a condition is true or false.



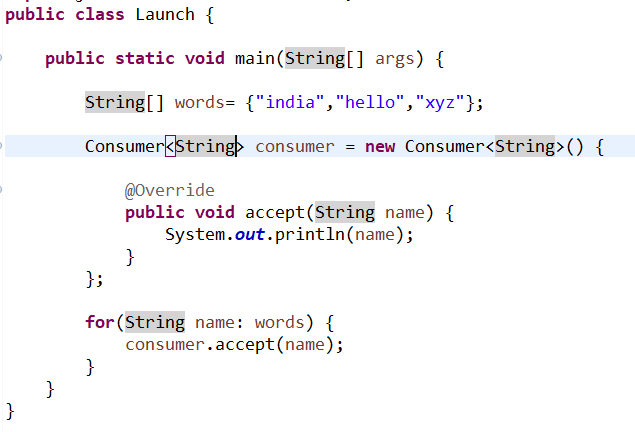
* Function<T, R>

The Function<T, R> interface represents a function that takes an argument of type T and returns a result of type R.



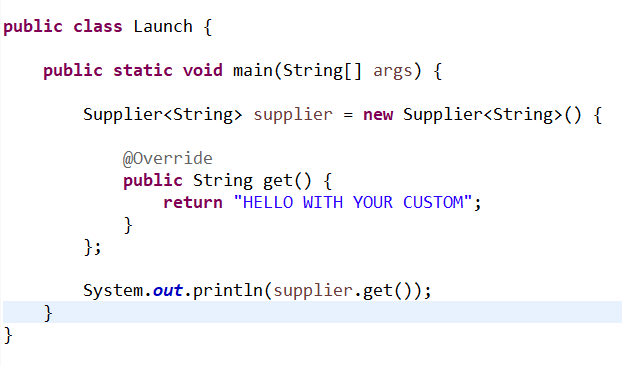
* Consumer<T>

The Consumer<T> interface represents an operation that takes a single argument and returns no result.



* Supplier<T>

The Supplier<T> interface represents a supplier of results. It provides a method that returns a result of type T with no input parameters.



<https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html>