What is a Functional Interface?

A **functional interface** is an **interface with exactly one abstract method**. This single abstract method defines the **contract** that can be implemented by:

- A **lambda expression**
- A **method reference**
- An **anonymous class**

Because it has just **one abstract method**, Java can automatically infer how your lambda should be converted to an instance of that interface.

Key Points

- **One abstract method:**
  A functional interface **must have only one abstract method**.
- **Can have default and static methods:**
  It **may include** any number of default or static methods without affecting its functional nature.
- **Annotated with @FunctionalInterface:**
  This annotation is **optional**, but recommended because:
  - It tells the compiler you intend this interface to be functional.
  - It causes an error if you accidentally add more abstract methods.
- **Used extensively in Streams and lambda expressions.**

Examples in the Java Standard Library

Java provides several built-in functional interfaces in **java.util.function** package. Common ones include:

Interface	Abstract Method	Purpose
Runnable	**run()**	Represents a task with no arguments and no result.
Supplier<T>	**get()**	Supplies a result of type T.
Consumer<T>	**accept(T t)**	Consumes a value of type T, returns nothing.
Function<T, R>	**apply(T t)**	Takes a value of type T and returns R.
Predicate<T>	**test(T t)**	Returns true or false for a given value.
BiFunction<T, U, R>	**apply(T t, U u)**	Takes two arguments and produces a result.

Why Are Functional Interfaces Important?

- They are the **foundation of lambda expressions** in Java.
- They enable **functional programming** constructs, such as mapping, filtering, reducing.
- They make code **more concise and expressive**.

Summary

Functional Interface
Exactly one abstract method
May have default/static methods
Enables lambdas and method references
Annotated with @FunctionalInterface (recommended)