Brief Introduction to java.util.function

The package **java.util.function** provides a set of **standard functional interfaces** that are used throughout Java's functional programming features—especially **Streams, Lambdas, and Optional**.

These interfaces represent **common function patterns** so you don't have to create your own.

The core four are:

---

1 Predicate<T>

- **Purpose**: Test a condition
- **Returns**: boolean
- **Example use**: Filtering in streams
- **Method**: boolean test(T t)

Used in: filter(), removeIf(), validation logic

2.Function<T,R>

- **Purpose**: Transform a value
- **Returns**: Any type (R)
- **Method**: R apply(T t)

Used in: map(), Optional.map(), data conversion

3.Supplier<T>

- **Purpose**: Provide a value with **no input**
- **Returns**: A new value each time

- **Method**: T get()

Used in: lazy initialization, orElseGet(), factories

---

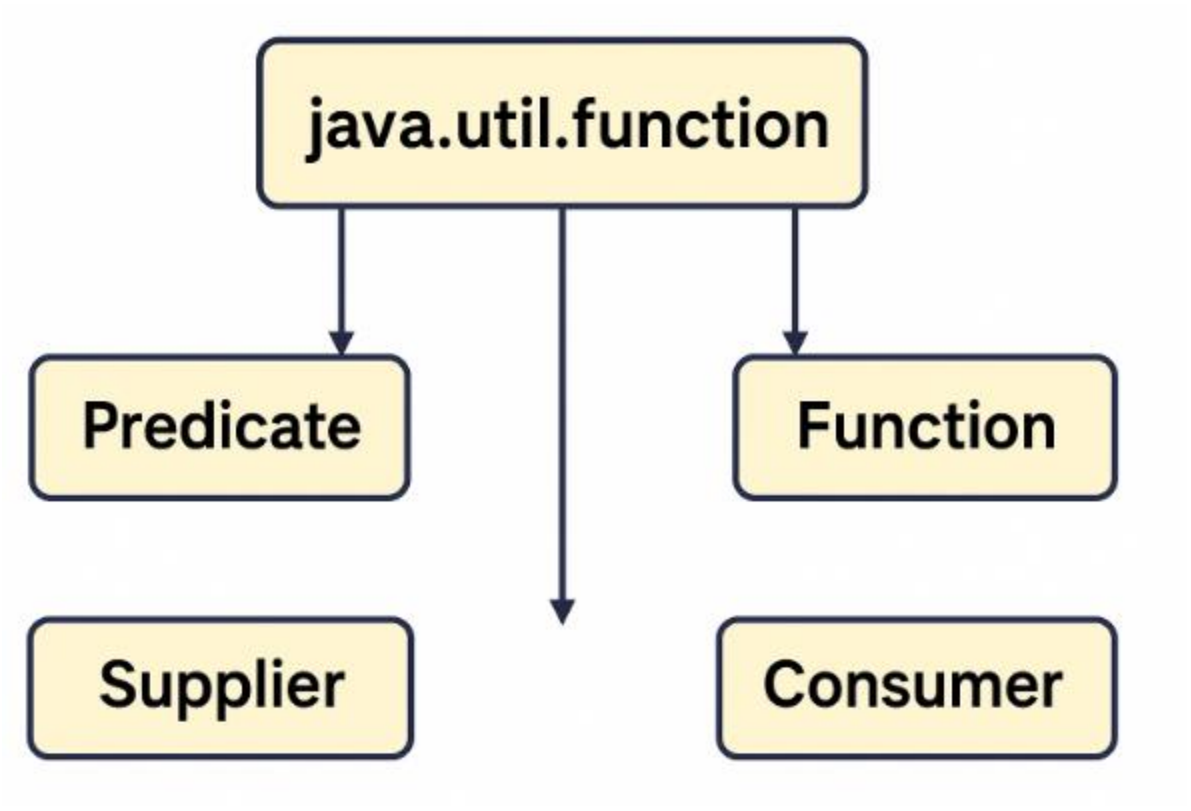
4. Consumer<T>

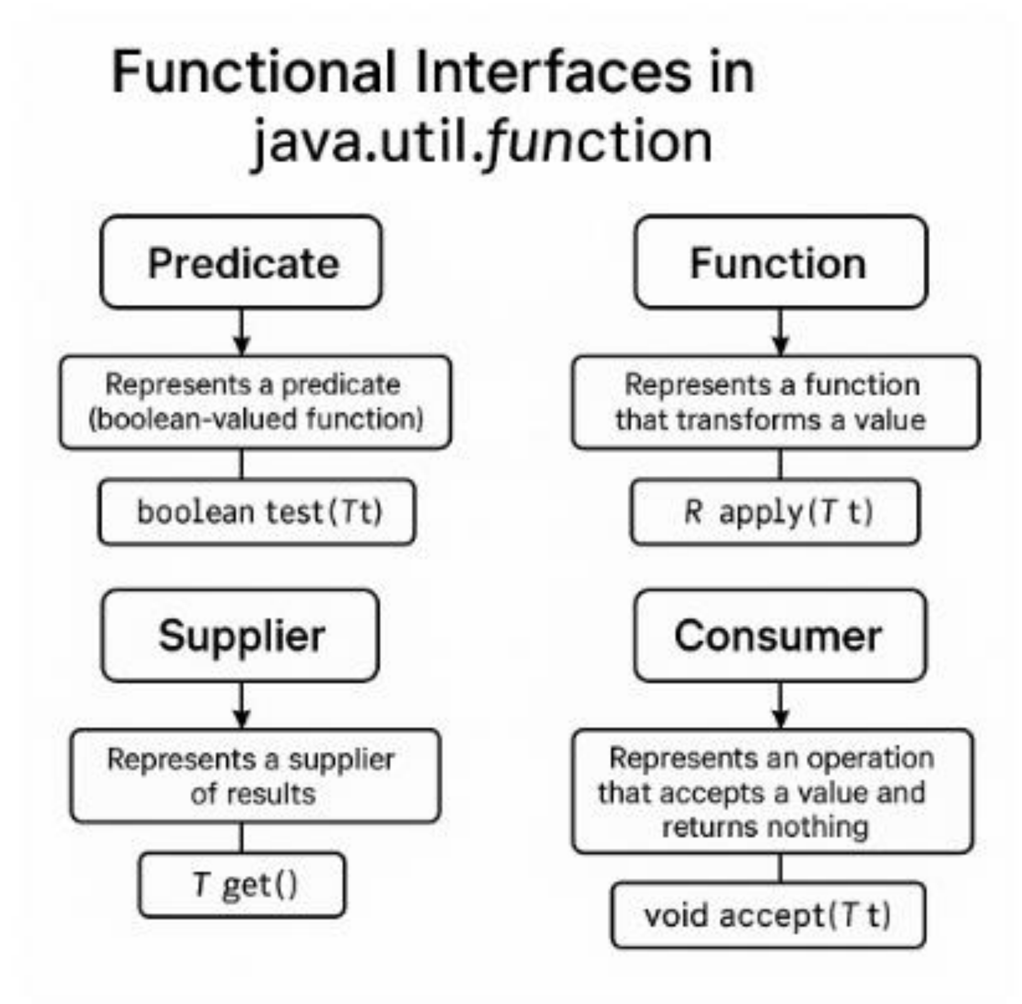
- **Purpose**: Accept a value and perform an action
- **Returns**: **nothing** (void)
- **Method**: void accept(T t)

Used in: forEach(), logging, printing, saving

Why these matter

These four are the **building blocks** of functional programming in Java.
All advanced interfaces (UnaryOperator, BinaryOperator, BiFunction, etc.) extend these patterns.


java.util.function
Predicate
Function
Supplier
Consumer


Functional Interfaces in java.util.*fun*ction
Predicate
Represents a predicate (boolean-valued function)
boolean test(*T*t)
Function
Represents a function that transforms a value
R apply(*T* t)
Supplier
Represents a supplier of results
T get()
Consumer
Represents an operation that accepts a value and returns nothing
void accept(*T* t)