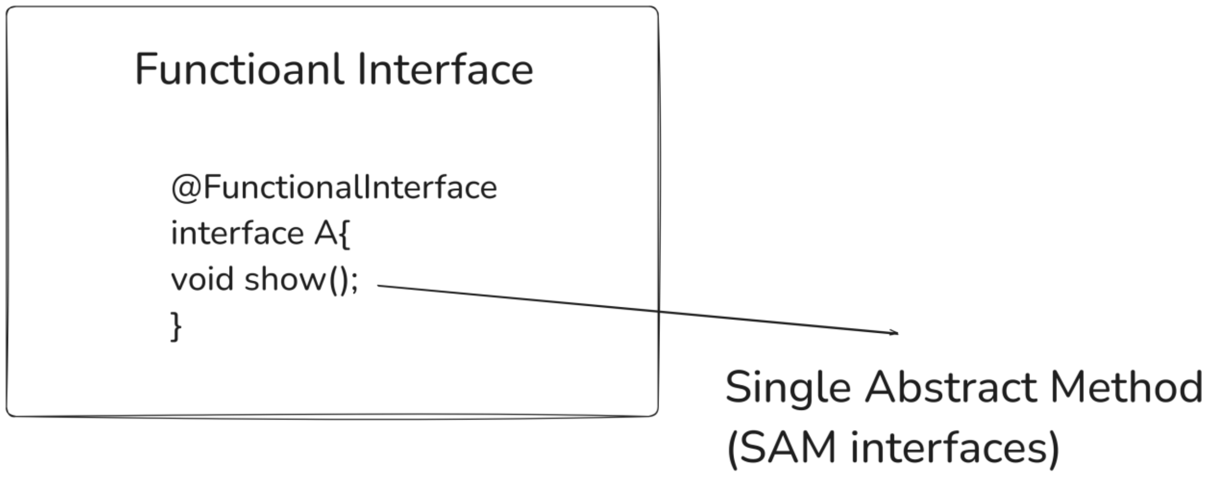
10.12-Functional Interface New

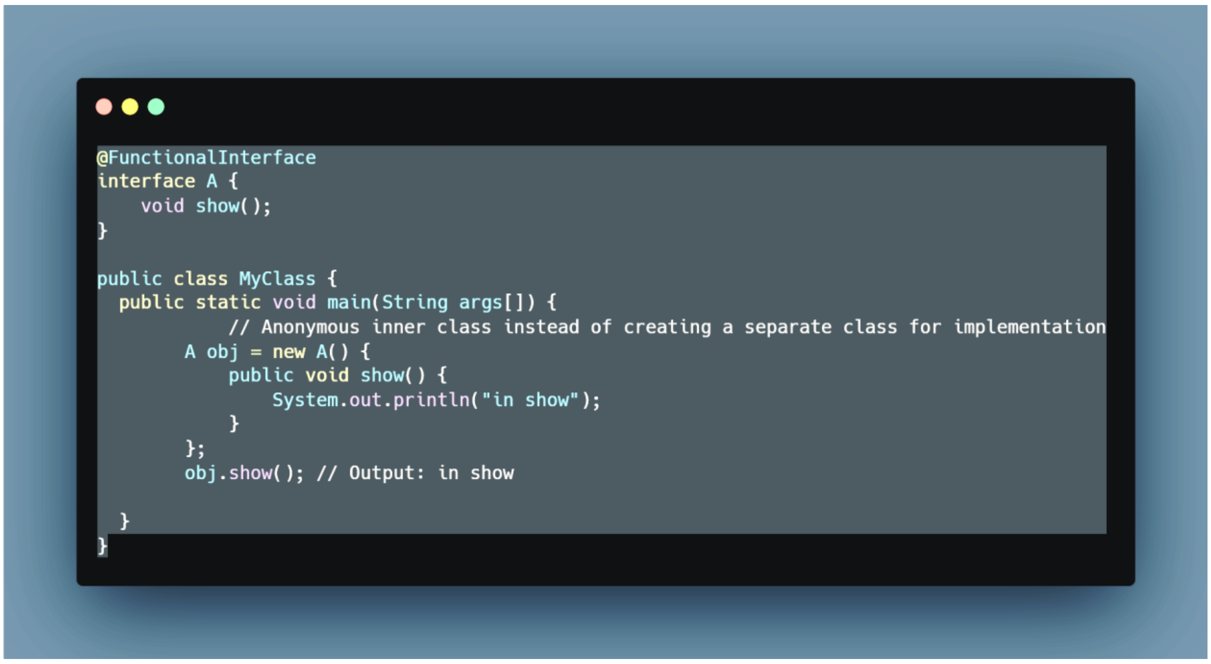
Introduction

A **functional interface** is an interface that contains only one abstract method. It is used when there is only one functionality that needs to be exhibited. In Java, functional interfaces are commonly referred to as **Single Abstract Method (SAM) Interfaces** because they have exactly one abstract method. These interfaces can have default and static methods, but only one abstract method. Functional interfaces are annotated with @FunctionalInterface to explicitly declare them as functional interfaces.



In Java SE 8 and later, functional interfaces work seamlessly with **lambda expressions** and **method references**, allowing cleaner, more readable code.

Example



```java
@FunctionalInterface
interface A {
    void show();
}

public class MyClass {
  public static void main(String args[]) {
          // Anonymous inner class instead of creating a separate class for implementation
        A obj = new A() {
            public void show() {
                System.out.println("in show");
            }
        };
        obj.show(); // Output: in show

  }
}
```

In this example, an anonymous inner class is used to provide an implementation for the show() method of interface A. Since A only has one method, it can be treated as a functional interface.

Built-In Java Functional Interfaces

From Java SE 1.8 onwards, several interfaces were converted into functional interfaces. These interfaces are annotated with @FunctionalInterface to enforce the functional interface design.

Here are some built-in functional interfaces:

- **Runnable**: Contains only the run() method.
- **Comparable**: Contains only the compareTo() method.
- **ActionListener**: Contains only the actionPerformed() method.
- **Callable**: Contains only the call() method.

These functional interfaces are widely used across Java applications and frameworks.

Functional Interfaces in Java 8

Java is often considered a verbose language because of its detailed and explicit code requirements. In Java 8, **lambda expressions** were introduced to reduce verbosity and make code cleaner and easier to understand. Lambda expressions are a key feature that works only with functional interfaces.

For example, instead of using anonymous inner classes, lambda expressions can simplify the code significantly.

Key Points

- A **functional interface** can only have one abstract method.
- Functional interfaces can have default and static methods, but only one abstract method.
- Lambda expressions and method references can be used with functional interfaces to simplify code.
- **@FunctionalInterface** annotation is used to explicitly declare an interface as functional, although it's optional.
- Functional interfaces are extensively used in **streams**, **concurrency**, and **event handling** in Java.