Java Training

CODEMIND
Your Success Partner

By CODEMIND Technology

Contact us 7758094241

JDK 1.8

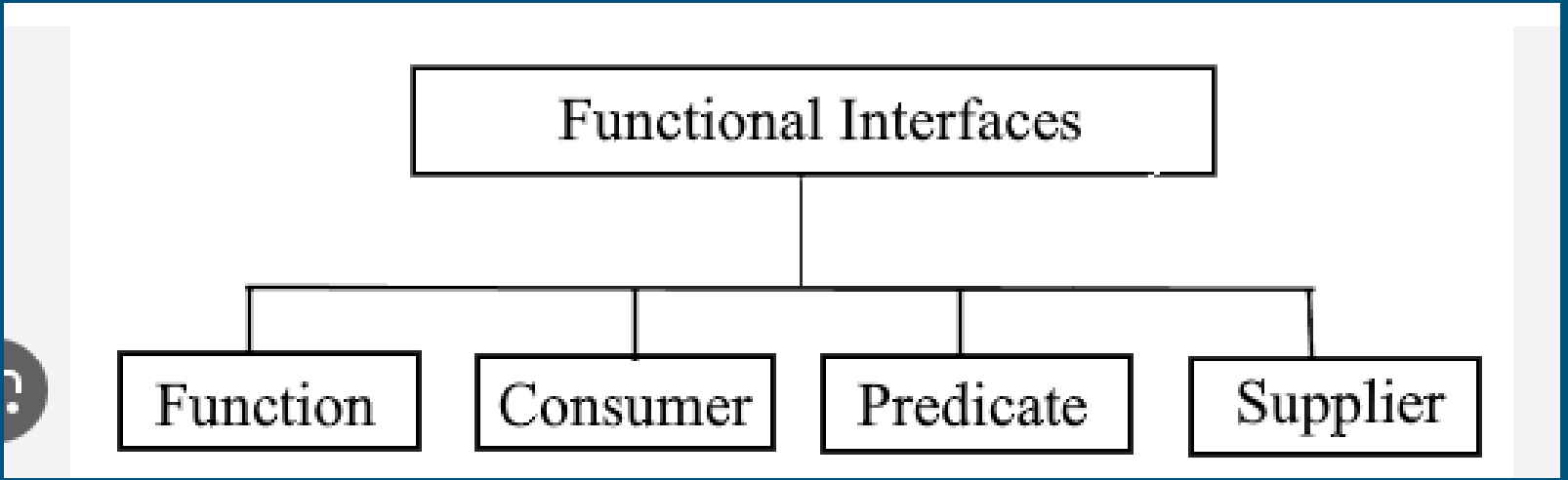
- ★ Introduction
- ★ Lambdas and Functional Interfaces
- ★ Functional Interface.
- ★ Interface’s Default and Static Methods
- ★ Method References

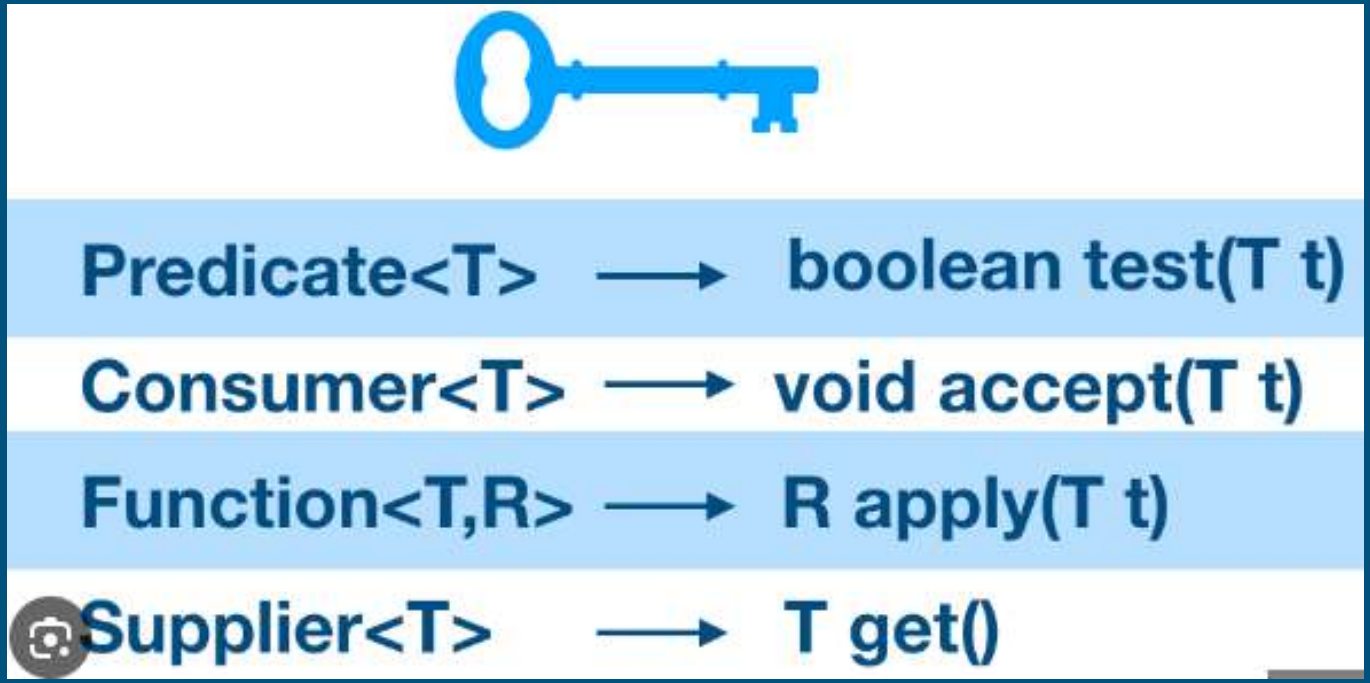
★ Functional Interface

- Key points about the functional interface: (functional programming approach.)

1. An Interface that contains exactly one abstract method is known as a *functional interface*. Aka *Single Abstract Method Interfaces or SAM Interfaces*
1. One or more *`default methods,`*
2. One or more *`static`* methods
3. Only ONE abstract method
4. Functional Interface is also known as . It is a new feature in Java 8, which helps to achieve a
5. A functional interface can extend another interface only when it does not have any abstract method.
6. Example :: *Runnable, Callable, Comparator, ActionListener*, and others

Java Predefined-Functional Interfaces




Predicate<T> → boolean test(T t)
Consumer<T> → void accept(T t)
Function<T,R> → R apply(T t)
Supplier<T> → T get()

Introduction

- Java 8 release is the greatest thing in the Java world since Java 5 (released quite a while ago, back in 2004)
- Language
- • compiler
- • libraries
- • tools
- • runtime (JVM)

Lambdas and Functional Interfaces

- Many languages on JVM platform (Groovy, Scala, . . .) have had lambdas since day one, but Java developers had no choice but hammer the lambdas with boilerplate anonymous classes
- new concise and compact language construct

```
Arrays.asList( "a", "b", "d" ).forEach( e -> System.out.println( e ) );
```

Please notice the type of argument **e** is being inferred by the compiler. Alternatively, you may explicitly provide the type of the parameter, wrapping the definition in brackets. For example:

```
Arrays.asList( "a", "b", "d" ).forEach( ( String e ) -> System.out.println( e ) );
```

In case lambda's body is more complex, it may be wrapped into square brackets, as the usual function definition in Java. For example:

```
Arrays.asList( "a", "b", "d" ).forEach( e -> {
    System.out.print( e );
    System.out.print( e );
} );
```

- Lambdas may reference the class members and local variables (implicitly making them effectively final if they are not). For example, those two snippets are equivalent:

```
String separator = ",";
 Arrays.asList( "a", "b", "d" ).forEach( ( String e ) -> System.out.print( e + separator ) );
```

```
final String separator = ",";
Arrays.asList( "a", "b", "d" ).forEach(
    ( String e ) -> System.out.print( e + separator ) );
```

Functional Interfaces

- The function interface is an interface with just one single method.
- The java.lang.Runnable and java.util.concurrent.Callable are two great examples of functional interfaces
- In practice, the functional interfaces are fragile: if someone adds just one another method to the interface definition, it will not be functional anymore and compilation process will fail.
- Java 8 adds special annotation @FunctionalInterface

```
@FunctionalInterface
public interface Functional {
    void method();
}
```

One thing to keep in mind: default and static methods do not break the functional interface contract and may be declared:

```
@FunctionalInterface
public interface FunctionalDefaultMethods {
    void method();

    default void defaultMethod() {
    }
}
```

Lambdas are the largest selling point of Java 8. It has all the potential to attract more and more developers to this great platform and provide state of the art support for functional programming concepts in pure Java. For more details please refer to official documentation

Interface’s Default and Static Methods

- Java 8 extends interface declarations with two new concepts: default and static methods
- The difference between default methods and abstract methods is that abstract methods are required to be implemented. But default methods are not.
- Each interface must provide so called default implementation and all the implementers will inherit it by default (with a possibility to override this default implementation if needed). Let us take a look on example below.

```
private interface Defaulable {
    // Interfaces now allow default methods, the implementer may or
    // may not implement (override) them.
    default String notRequired() {
        return "Default implementation";
    }
}

private static class DefaultableImpl implements Defaulable {
}
```

Method References

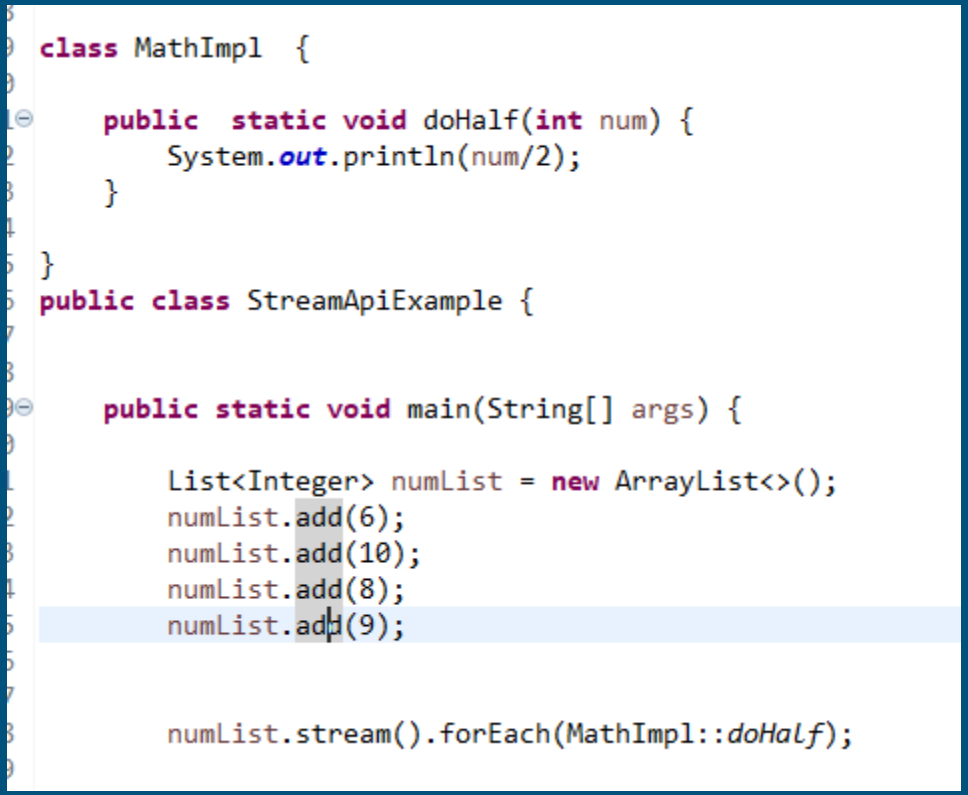
- Method references provide the useful syntax to refer directly to exiting methods or constructors of Java classes or objects (instances).
- With conjunction of Lambdas expressions, method references make the language constructs look compact and concise, leaving off boilerplate.
- Method references help to point to methods by their names. A method reference is described using "::" symbol. A method reference can be used to point the following types of methods:
  - Static methods
  - Instance methods
  - Constructors using new operator (TreeSet::new)

If a Lambda expression is like:

```
// If a lambda expression just call a static method of a class
(args) -> Class.staticMethod(args)
```

Then method reference is like:

```
// Shorthand if a lambda expression just call a static method of a class
Class::staticMethod
```



```
class MathImpl  {

    public  static void doHalf(int num) {
        System.out.println(num/2);
    }

}
public class StreamApiExample {


    public static void main(String[] args) {

        List<Integer> numList = new ArrayList<>();
        numList.add(6);
        numList.add(10);
        numList.add(8);
        numList.add(9);


        numList.stream().forEach(MathImpl::doHalf);

```

Thank you

Contact us - 7758094241