Supplier Functional Interface

In this lesson, we will look at the supplier functional interface.

We'll cover the following

`Supplier` is an interface that does not take in any argument but produces a value when the `get()` function is invoked. Suppliers are useful when we don't need to supply any value and obtain a result at the same time.

Below are some of the functional interfaces, which can be categorized as a supplier.

Interface Name	Description	Abstract Method
`Supplier<T>`	Represents a supplier of results (reference type)	`T get()`
`DoubleSupplier`	A supplier of double-value results	`double getAsDouble()`
`IntSupplier`	A supplier of int-value results	`int getAsInt()`
`LongSupplier`	A supplier of long-value results	`long getAsLong()`
`BooleanSupplier`	A supplier of boolean-value results	`boolean getAsBoolean()`

`Supplier<T>`

The `Supplier<T>` interface supplies a result of type `T`. In the previous lesson, we were passing a person object and a predicate to our `isPersonEligibleForVoting()` method.

In this example, we will provide a `Supplier<Person>` instead of the `Person` object. The `isPersonEligibleForVoting()` method will, itself, fetch the `Person` object from the supplier. Here is the code for this.

```
import java.util.function.Predicate;
import java.util.function.Supplier;

public class SupplierTest {

  static boolean isPersonEligibleForVoting(
      Supplier<Person> supplier, Predicate<Person> predicate) {
    return predicate.test(supplier.get());
  }

  public static void main(String args[]) {
    Supplier<Person> supplier = () -> new Person("Alex", 23);
    Predicate<Person> predicate = (p) -> p.age > 18;
    boolean eligible =
        isPersonEligibleForVoting(supplier, predicate);
    System.out.println("Person is eligible for voting: " + eligible);
  }
}

class Person {
  String name;
  int age;

  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

The `Supplier<T>` interface does not contain any default or static methods. Let us look at some of the primitive specializations of the supplier interface.

`IntSupplier`

The `IntSupplier` interface has a method `getAsInt()`, which applies the given operation on its argument and returns an int value. It is similar to using an object

of type `Supplier<Integer>`.

```
import java.util.function.IntSupplier;

public class SupplierDemo {

  public static void main(String args[]) {

        IntSupplier supplier = () -> (int)(Math.random() * 10);

        System.out.println(supplier.getAsInt());
  }
}
```

DoubleSupplier

The `DoubleSupplier` interface has a method `getAsDouble()`, which applies the given operation on its argument and returns a double value. It is similar to using an object of type `Supplier<Double>`.

```
import java.util.function.DoubleSupplier;

public class SupplierDemo {

  public static void main(String args[]) {

        DoubleSupplier supplier = () -> (int)(Math.random() * 10);

        System.out.println(supplier.getAsDouble());
  }
}
```

1 What does `T` represent in `Supplier<T>` ?

2

What is the default method of `IntSupplier`?

Retake Quiz

In the next lesson, we will look at the `Consumer` functional interfaces. These interfaces are the opposite of suppliers.