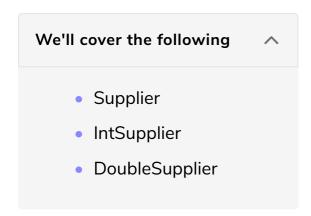
Supplier Functional Interface

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



Supplier is an interface that does not take in any argument but produces a value when the <code>get()</code> 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></t> | Represents a supplier of results (reference type) | T get() |
| DoubleSupplier | A supplier of double- value results | <pre>double getAsDouble()</pre> |
| IntSupplier | A supplier of int-value results | <pre>int getAsInt()</pre> |
| LongSupplier | A supplier of long-value results | <pre>long getAsLong()</pre> |
| 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 <code>DoubleSupplier</code> interface has a method <code>getAsDouble()</code>, which applies the given operation on its argument and returns a double value. It is similar to using an object of type <code>Supplier<Double></code>.

```
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());
   }
}
```





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.