CompletableFuture: Combining Results of Futures

This lesson explains how we can combine the results of an arbitrary number of futures together.

We'll cover the following

In the previous lesson, we used the `thenCombine()` and `thenCompose()` methods to combine the result of two futures.

If we need to run multiple futures in parallel and combine their result then we can use the `allOf()` and `anyOf()` methods.

1) `allOf()`

Here are a few important points regarding `allOf()` method:

1. It returns a new CompletableFuture that is completed when all of the given CompletableFutures are completed.
2. If any of the given CompletableFutures complete exceptionally, the returned CompletableFuture also completes, with a CompletionException holding this exception as its cause.
3. The results, if any, of the given CompletableFutures are not reflected in the returned CompletableFuture, but they may be obtained by inspecting them individually.
4. If no CompletableFutures are provided, it returns a CompletableFuture completed with the value null.

```
import java.util.concurrent.*;

public class CompletableFutureDemo {

    public static void main(String args[]) {
```

```
    public static void main(String args[]) {

        CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() -> 50);
        CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() -> 40);
        CompletableFuture<Integer> future3 = CompletableFuture.supplyAsync(() -> 30);

        CompletableFuture<Void> finalFuture = CompletableFuture.allOf(future1, future2, future3);

        try {
            finalFuture.get();
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("Code that should be executed after all the futures complete.");
    }
}
```

2) `join()`

Since the `allOf()` method returns a `CompletableFuture<Void>`, we can't combine the result of all the futures. We need to manually get the result of all the futures.

We can use the `join()` method to combine the result of all futures. The join method returns the result value when complete, or it throws an (unchecked) exception if completed exceptionally.

```
import java.util.Optional;
import java.util.concurrent.*;
import java.util.stream.Stream;

public class CompletableFutureDemo {

    public static void main(String args[]) {

        CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() -> 50);
        CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() -> 40);
        CompletableFuture<Integer> future3 = CompletableFuture.supplyAsync(() -> 30);

        Optional<Integer> maxElement = Stream.of(future1, future2, future3)
                .map(CompletableFuture::join) // This will return the stream of results of all fut
                .max(Integer::compareTo);

        System.out.println("The max element is " + maxElement);
    }
}
```

3) `anyOf()`

Here are a few important points regarding the `anyOf()` method:

1. It returns a new CompletableFuture that is completed when any of the given CompletableFutures complete with the same result.

2. If it is completed exceptionally, the returned CompletableFuture also does so, with a CompletionException holding this exception as its cause.

3. If no CompletableFutures are provided, it returns an incomplete CompletableFuture.

```java
import java.util.concurrent.*;

public class CompletableFutureDemo {

    public static void main(String args[]) {

        CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() -> 50);
        CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() -> 40);
        CompletableFuture<Integer> future3 = CompletableFuture.supplyAsync(() -> 30);

        //The first completed future will be returned.
        CompletableFuture<Object> firstCompletedFuture = CompletableFuture.anyOf(future1, future2,

        try {
            System.out.println("The first completed future value is " + firstCompletedFuture.get()
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("Code that should be executed after any of the futures complete.");
    }
}
```

The next lesson will introduce you to a new kind of `Lock` class called the `StampedLock`.