- CompletionService Interface
- ThreadLocal
- ThreadLocalRandom
- CountDownLatch
- CyclicBarrier
- Concurrent Collections
- ConcurrentHashMap
- ConcurrentModificationException

Practice Mock Interview

CompletionException

Guide to understanding CompletionException

*If you are interviewing, consider buying our **number#1** course for **Java Multithreading Interviews**.*

Overview

The `CompletionException` extends from the `RuntimeException` and is thrown when a task hits an error or exception when executing.

The `CompletionException` and the `ExecutionException` might seem similar but there's a fundamental difference between the two. The `ExecutionException` is thrown only when an attempt is made to retrieve the result/outcome of a task, while the `CompletionException` can be thrown when waiting on a task to complete or when trying to retrieve a result computed by a task. In a sense `CompletionException` can be thought of as broader in scope than `ExecutionException`.

Example

We'll use the `CompletableFuture` class to demonstrate the `CompletionException`. The `CompletableFuture` class is used to program asynchronously in Java. It exposes a method `completeExceptionally` that takes in an instance of `Throwable` to indicate the failure experienced by the task. When the main thread attempts to wait for the task to complete by

by a task. In a sense `CompletionException` can be thought of as broader in scope than `ExecutionException`.

Example

We'll use the `CompletableFuture` class to demonstrate the `CompletionException`. The `CompletableFuture` class is used to program asynchronously in Java. It exposes a method `completeExceptionally` that takes in an instance of `Throwable` to indicate the failure experienced by the task. When the main thread attempts to wait for the task to complete by

invoking the `join()` method the `CompletionException` is thrown.

Java

main.java

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

class Demonstration {
    public static void main( String args[] ) {

        // create an executor service and a completable future
        ExecutorService executorService = Executors.newFixedThreadPool(5);
        CompletableFuture<Integer> completableFuture = new CompletableFuture<>();

        try {
            // create a command that marks the CompletableFuture instance complete exception
            Runnable command = new Runnable() {
```

Run

Note that in the above program, if we attempt to retrieve the result of the task, which should be an integer we'll get `ExecutionException` instead of the `CompletionException`.

Java

Press + to interact

main.java

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

class Demonstration {
    public static void main( String args[] ) {

        // create an executor service and a completable future
        ExecutorService executorService = Executors.newFixedThreadPool(5);
        CompletableFuture<Integer> completableFuture = new CompletableFuture<>();

        try {
            // create a command that marks the CompletableFuture instance complete exception
            Runnable command = new Runnable() {
```

Run

The `CompletableFuture` also offers another method `getNow()` that is non-blocking unlike the `get()` method. The `getNow()` method either returns the computed value if the future has completed or a default value that is passed-in at the time of the invocation of the method. The `getNow()` method throws the `CompletionExecution` instead of the `ExecutionException` in case the future is exceptionally completed as shown in the following program.

Java

Press + to interact

main.java

Saved

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

class Demonstration {
    public static void main( String args[] ) {

        // create an executor service and a completable future
        ExecutorService executorService = Executors.newFixedThreadPool(5);
        CompletableFuture<Integer> completableFuture = new CompletableFuture<>();

        try {
            // create a command that marks the CompletableFuture instance complete exception
            Runnable command = new Runnable() {
```

Run

Interestingly, if the `CompletableFuture` completes successfully but then throws an exception, the `join()` and `get()` methods don't throw any exception in the main thread as demonstrated by the following program.

Java

Press + to interact

main.java

Saved

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

class Demonstration {
    public static void main( String args[] ) {

        // create an executor service and a completable future
        ExecutorService executorService = Executors.newFixedThreadPool(5);
        CompletableFuture<Integer> completableFuture = new CompletableFuture<>();

        try {
            // create a command that marks the CompletableFuture instance complete exception
            Runnable command = new Runnable() {
```

Run