Practices for Lesson 15: Concurrency

Chapter 15

Practices for Lesson 15: Overview

Practices Overview

In these practices, you will use the `java.util.concurrent` package and sub-packages of the Java programming language.

Practice 15-1: Summary Level: Using the `java.util.concurrent` Package

Overview

In this practice, you will modify an existing project to use an `ExecutorService` from the `java.util.concurrent` package.

Assumptions

You have reviewed the sections covering the use of the `java.util.concurrent package`.

Summary

You will create a simple multithreaded counting application. Instead of manually creating threads, you will leverage an `ExecutorService` from the `java.util.concurrent` package.

Tasks

1. Open the `ConCount15-01Prac` project as the main project.
   a. Select File > Open Project.
   b. Browse to /home/oracle/labs/15-Concurrency/practices/practice1.
   c. Select ConCount15-01Prac and click the Open Project button.
2. Expand the project directories.
3. Open the `CountRunnable` class in the `com.example` package.
4. Create a constructor to initialize the `count` and `threadName` variables.
5. Uncomment the `count` and `threadName` variables.
6. In the run method, setup a `for` loop to print out the thread name and each number counted.
7. Open the `Main` class in the `com.example` package.
8. Setup the `ExecutorService` in the main method using the `Executors` class and the `newCachedThreadPool` method.
9. Setup three `CountRunnable` objects to count to 20, named threads A, B, and C.
10. Shut down the `ExecutorService`.
11. Run the project. You should see each thread count to 20. Because of out of order processing, the counts of the three threads should be all jumbled together.

Practice 15-2: Detailed Level: Using the `java.util.concurrent` Package

Overview

In this practice, you will modify an existing project to use an `ExecutorService` from the `java.util.concurrent` package.

Assumptions

You have reviewed the sections covering the use of the `java.util.concurrent package`.

Summary

You will create a simple multithreaded counting application . Instead of manually creating threads, you will leverage an `ExecutorService` from the `java.util.concurrent` package.

Tasks

1. Open the `ConCount15-01Prac` project as the main project.
   a. Select File > Open Project.
   b. Browse to /home/oracle/labs/15-Concurrency/practices/practice1.
   c. Select ConCount15-01Prac and click the Open Project button.
2. Expand the project directories.
3. Open the `CountRunnable` class in the `com.example` package.
4. Create a constructor to initialize the `count` and `threadName` variables.

```
 public CountRunnable(int count, String name){
   this.count = count;
   this.threadName = name;
 }
```

5. Uncomment the `count` and `threadName` variables.

```
 final int count;
 final String threadName;
```

6. In the run method, set up a `for` loop to print out the thread name and each number counted.

```
   for (int i = 1; i <= count; i++){
     System.out.println("Thread " + threadName +
       ": " + i);
   }
```

7. Open the `Main` class in the `com.example` package.
8. Setup the `ExecutorService` in the main method using the `Executors` class and the `newCachedThreadPool` method.

```
   ExecutorService es = Executors.newCachedThreadPool();
```

9. Setup three `CountRunnable` objects to count to 20, named threads A, B, and C.

```
   es.submit(new CountRunnable(20,"A"));
   es.submit(new CountRunnable(20,"B"));
   es.submit(new CountRunnable(20,"C"));
```

10. Shut down the `ExecutorService`.

```
    es.shutdown();
```

11. Run the project. You should see each thread count to 20. Because of out of order processing, the counts of the three threads should be all jumbled together.

Practice 15-2: Summary Level: Create a Network Client using the `java.util.concurrent` Package

Overview

In this practice, you will modify an existing project to use an `ExecutorService` from the `java.util.concurrent` package.

Assumptions

You have reviewed the sections covering the use of the `java.util.concurrent` package.

Summary

You will create a multithread networking client that will rapidly read the price of a shirt from several different servers. Instead of manually creating threads, you will leverage an `ExecutorService` from the `java.util.concurrent` package.

Tasks

1. Open the `ExecutorService15-02Prac` project as the main project.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/15-Concurrency/practices/practice2`.
   c. Select `ExecutorService15-02Prac` and click the Open Project button.
2. Expand the project directories.
3. Run the `NetworkServerMain` class in the `com.example.server` package by right-clicking the class and selecting Run File.
4. Open the `NetworkClientMain` class in the `com.example.client` package.
5. Run the `NetworkClientMain` class package by right-clicking the class and selecting Run File. Notice the amount of time it takes to query all the servers sequentially.
6. Create a `NetworkClientCallable` class in the `com.example.client` package.
   a. Add a constructor and a field to receive and store a `RequestResponse` reference.
   b. Implement the `Callable` interface with a generic type of `RequestResponse`.

```
public class NetworkClientCallable implements
Callable<RequestResponse>
```

   c. Complete the `call` method by using a `java.net.Socket` and a `java.util.Scanner` to read the response from the server. Store the result in the `RequestResponse` object and return it.

      Note: You may want to use a `try-with-resource` statement to ensure that the `Socket` and `Scanner` objects are closed.
7. Modify the `main` method of the `NetworkClientMain` class to query the servers concurrently by using an `ExecutorService`.
   a. Comment out the contents of the `main` method.
   b. Obtain an `ExecutorService` that reuses a pool of cached threads.
   c. Create a Map that will be used to tie a request to a future response.

```
Map<RequestResponse, Future<RequestResponse>> callables = new
HashMap<>();
```

   d. Code a loop that will create a `NetworkClientCallable` instance for each network request.

e. The servers should be running on localhost, ports 10000–10009.
f. Submit each `NetworkClientCallable` to the ExecutorService. Store each `Future` in the `Map` created in step 7c.
g. Shut down the `ExecutorService`.
h. Await the termination of all threads within the `ExecutorService` for 5 seconds.
i. Loop through the `Future` objects stored in the `Map` created in step 7c. Print out the servers' response or an error message with the server details if there was a problem communicating with a server.

8. Run the `NetworkClientMain` class by right-clicking the class and selecting Run File. Notice the amount of time it takes to query all the servers concurrently.
9. When done testing your client, be sure to select the ExecutorService output tab and terminate the server application.

Practice 15-2: Detailed Level: Create a Network Client using the `java.util.concurrent` Package

Overview

In this practice, you will modify an existing project to use an `ExecutorService` from the `java.util.concurrent` package.

Assumptions

You have reviewed the sections covering the use of the `java.util.concurrent` package.

Summary

You will create a multithread networking client that will rapidly read the price of a shirt from several different servers. Instead of manually creating threads, you will leverage an `ExecutorService` from the `java.util.concurrent` package.

Tasks

1. Open the `ExecutorService15-02Prac` project as the main project.
   a. Select File > Open Project.
   b. Browse to /home/oracle/labs/15-Concurrency/practices/practice2.
   c. Select ExecutorService15-02Prac and click the Open Project button.
2. Expand the project directories.
3. Run the `NetworkServerMain` class in the `com.example.server` package by right-clicking the class and selecting Run File.
4. Open the `NetworkClientMain` class in the `com.example.client` package.
5. Run the `NetworkClientMain` class package by right-clicking the class and selecting Run File. Notice the amount of time it takes to query all the servers sequentially.
6. Create a `NetworkClientCallable` class in the `com.example.client` package that implements the Callable interface. Use the notation for generics to define the Callable as of type RequestResponse.

```
public class NetworkClientCallable implements
Callable<RequestResponse>
```

   NetBeans shortcut: Right-click and select Fix Imports to add the necessary import statement.

   a. Add a constructor and a field named lookup of type `RequestResponse` to receive and store a `RequestResponse` reference during construction.

   NetBeans shortcut: Add the field first, as a private class field, then right-click and select Insert Code. Then Select Constructor. Select the lookup field and click Generate.

   b. Implement the `Callable` interface with a generic type of `RequestResponse`.

   NetBeans shortcut: Select the light bulb beside the class signature and click Implement all abstract methods.

   c. Remove the line of code in the generated `call` method.

d. Complete the `call` method by using a `java.net.Socket` and a `java.util.Scanner` to read the response from the server. Store the result in the `RequestResponse` object and return it.

   Note: You may want to use a `try-with-resource` statement to ensure that the `Socket` and `Scanner` objects are closed.

```
try (Socket sock = new Socket(lookup.host, lookup.port);
  Scanner scanner = new Scanner(sock.getInputStream())) {
  lookup.response = scanner.next();
  return lookup;
}
```

e. Use the NetBeans hint above to add the necessary import statements.

f. Note: Click the lightbulb with the caution triangle next to the class field to add `final` to the class field instance.

g. Save the file.

7. Modify the `main` method of the `NetworkClientMain` class to query the servers concurrently by using an `ExecutorService`.

   a. Comment out the contents of the `main` method.

   b. Obtain an `ExecutorService` that reuses a pool of cached threads.

```
ExecutorService es = Executors.newCachedThreadPool();
```

   c. Create a `Map` that will be used to tie a request to a future response.

```
Map<RequestResponse, Future<RequestResponse>> callables = new
HashMap<>();
```

   d. Copy the following lines of the `for` loop and code that creates an instance of a `RequestResponse` from the commented out code:

```
String host = "localhost";
for (int port = 10000; port < 10010; port++) {
  RequestResponse lookup = new RequestResponse(host, port);
```

   e. Add a line of code that creates an instance of a `NetworkClientCallable` and passes the instance of the `RequestResponse` object to it for each network request.

   f. Submit each `NetworkClientCallable` to the ExecutorService. Store each `Future` in the `Map` created above.

   g. Your complete for loop should look like this:

```
for (int port = 10000; port < 10010; port++) {
  RequestResponse lookup = new RequestResponse(host, port);
  NetworkClientCallable callable =
    new NetworkClientCallable(lookup);
  Future<RequestResponse> future = es.submit(callable);
  callables.put(lookup, future);
}
```

   h. Shut down the `ExecutorService`.

i. Await the termination of all threads within the `ExecutorService` for 5 seconds. Recall from the lesson that `awaitTermination` method throws an `InterruptedException`, so use a try-catch block.

```
es.shutdown();

try {
  es.awaitTermination(5, TimeUnit.SECONDS);
} catch (InterruptedException ex) {
  System.out.println("Stopped waiting early");
}
```

j. Loop through the `Future` objects stored in the `Map` created above. Use the `keyset` method to return and `Iterable` that contains the `RequestResponse` object.

k. Get the `Future<RequestResponse>` object from the `RequestResponse` object retrieved from the `Map`.

l. Print out the servers' response or an error message with the server details if there was a problem communicating with a server.

m. Your code should look similar to this:

```
for (RequestResponse lookup : callables.keySet()) {
  Future<RequestResponse> future = callables.get(lookup);
  try {
    lookup = future.get();
    System.out.println(lookup.host + ":" + lookup.port + " " +
                       lookup.response);
  } catch (ExecutionException | InterruptedException ex) {
    System.out.println("Error talking to " + lookup.host +
                       ":" + lookup.port);
  }
}
```

8. Run the `NetworkClientMain` class by right-clicking the class and selecting Run File. Notice the amount of time it takes to query all the servers concurrently.
9. When done testing your client, be sure to select the `ExecutorService` output tab and terminate the server application.