First Iteration - A command-line Application - Part 3

Back To Test-Driven Implementation

If you review our JUnitTest, we added a test (previous article) that expects to queue a request, and then be able to retrieve the list of pending requests, and assert that there is only 1 request pending.

That test fails at the moment because we need to make our ShapeCalculatorServiceImpl do some real work to make that happen.

Here is what I chose to add...

ShapeCalculatorServiceImpl:

```
// .... package name and imports here....

@Component

public class ShapeCalculatorServiceImpl implements ShapeCalculatorService {

private PendingRequests pendingRequests;

private CalculatedResults calculatedResults;

// ....some irrelevant code here....

@Override

public void queueCalculationRequest(

        ShapeName shapeName, CalcType calcType, double dimension) {


    if (null==shapeName) { throw new IllegalArgumentException('ShapeName can not
be null'); }

    if (null==calcType) { throw new IllegalArgumentException("CalcType can not be
null"); }

    if (0>dimension) { throw new IllegalArgumentException("dimension must be zero or
positive"); }

    CalculationRequest request = new
CalculationRequest(shapeName,calcType,dimension);

    if (calculatedResults.containsRequest(request)) { return; }

    pendingRequests.putRequest(request);
```

```
}
```

I added two private members, and some more logic to the "queueCalculationRequest()" method.

If you review our requirements (see Javadoc in the interface of this implementation), then we know that once we have run a calculation, there is no need to re-run it, since it will yield the same result. Also, re-adding a pending request to the list should have no effect (if we code it correctly).

The two new members ("PendingRequests" and "CalculatedResults") I chose to be interfaces.

Why? For now, we could just use some Java collection... but our goal is to have this service persist the requests and results, and we want to use JPA / Hibernate. Thus the interfaces.

Further, I want Spring to load up these members for us, in our continuing attempt to make all this as de-coupled as possible.

There are many ways to do something, and many reasons for and against - I just chose the following as a way to continue to introduce more features. My focus is NOT to make this service production-ready, even though I may add some robustness here and there.

Since our ShapeCalculatorServiceImpl has been annotated as a @Component, that means Spring context knows and manages it. Thus, we can also have Spring introduce or inject other components into it.

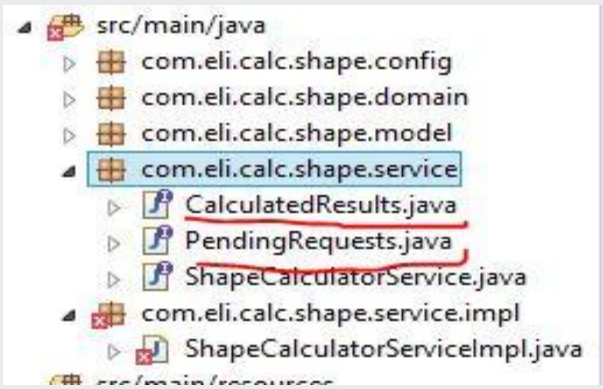
First, though, I would like to change the current @Component annotation. Replace it with @Service. This is just a more specialized stereotype of @Component.

The next thing is to inform Spring to set up our new internal members. For that we use @Autowired.

So now we have:

```
@Service
public class ShapeCalculatorServiceImpl implements ShapeCalculatorService {
@Autowired
private PendingRequests pendingRequests;
@Autowired
private CalculatedResults calculatedResults;
```

I had already created the two new interfaces:



We need to start off the implementations of those two, and add the @Component annotation to each, so that Spring will find them when scanning, and autowire instances of them into our service implementation.

PendingRequests:

```
package com.eli.calc.shape.service;
import java.util.List;
import com.eli.calc.shape.domain.CalculationRequest;
public interface PendingRequests {
List<CalculationRequest> getRequests();
void putRequest(CalculationRequest request);
void removeRequest(CalculationRequest request);
long getNumRequests();
}
```

CalculatedResults:

```
package com.eli.calc.shape.service;
import java.util.List;
import com.eli.calc.shape.domain.CalculationRequest;
import com.eli.calc.shape.domain.CalculationResult;
public interface CalculatedResults {
void deleteAllResults();
void putResult(CalculationResult result);
void removeResult(CalculationResult result);
boolean containsRequest(CalculationRequest request);
List<CalculationResult> listResults();
```

```
}
```

PendingRequestsImpl:

```
package com.eli.calc.shape.service.impl;
import java.util.List;
import org.springframework.stereotype.Component;
import com.eli.calc.shape.domain.CalculationRequest;
import com.eli.calc.shape.service.PendingRequests;
@Component
public class PendingRequestsImpl implements PendingRequests {
@Override
public List<CalculationRequest> getRequests() {
// TODO Auto-generated method stub
return null;
}
@Override
public void putRequest(CalculationRequest request) {
// TODO Auto-generated method stub
}
@Override
public void removeRequest(CalculationRequest request) {
// TODO Auto-generated method stub
}
@Override
public long getNumRequests() {
// TODO Auto-generated method stub
return 0;
}
}
```

CalculatedResultsImpl:

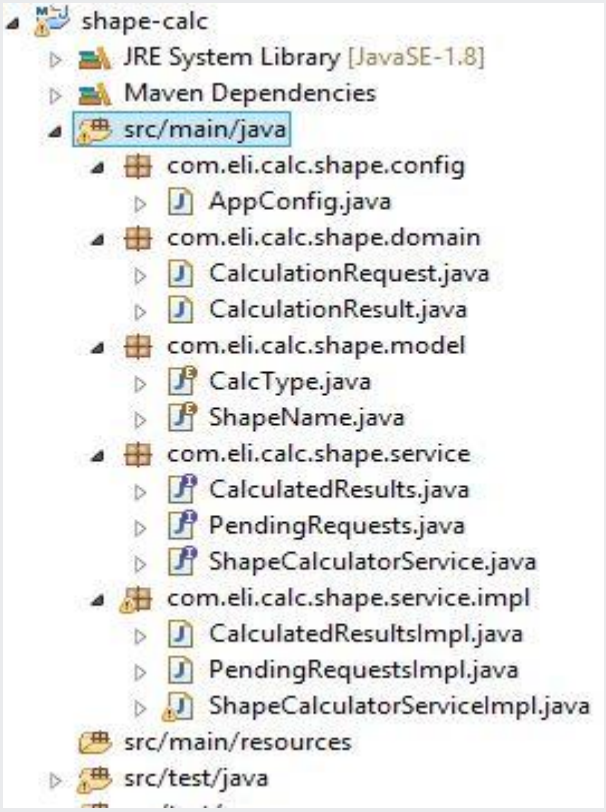
```
package com.eli.calc.shape.service.impl;
import java.util.List;
import org.springframework.stereotype.Component;
import com.eli.calc.shape.domain.CalculationRequest;
import com.eli.calc.shape.domain.CalculationResult;
import com.eli.calc.shape.service.CalculatedResults;
@Component
public class CalculatedResultsImpl implements CalculatedResults {
@Override
public void deleteAllResults() {
// TODO Auto-generated method stub
}
@Override
public void putResult(CalculationResult result) {
// TODO Auto-generated method stub
}
@Override
public void removeResult(CalculationResult result) {
// TODO Auto-generated method stub
}
@Override
public boolean containsRequest(CalculationRequest request) {
// TODO Auto-generated method stub
return false;
}
@Override
public List<CalculationResult> listResults() {
// TODO Auto-generated method stub
return null;
}
}
```

We also need to, based on our code in the queue request operation of our service implementation, add a constructor to the CalculationRequest class.

CalculationRequest:

```
package com.eli.calc.shape.domain;

import com.eli.calc.shape.model.CalcType;

import com.eli.calc.shape.model.ShapeName;

public final class CalculationRequest {

   public CalculationRequest(ShapeName shapeName, CalcType calcType, double dimension) {

   }

}
```

Here is our current package structure once more:



Our JUnitTest still fails because PendingRequestsImpl do not deal with any real collection. So let's add one (this will likely change when we later begin to employ Hibernate).

I fleshed out the PendingRequestsImpl:

```
package com.eli.calc.shape.service.impl;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.HashSet;
import org.springframework.stereotype.Component;
import com.eli.calc.shape.domain.CalculationRequest;
import com.eli.calc.shape.service.PendingRequests;
@Component
public class PendingRequestsImpl implements PendingRequests {
private final Set<CalculationRequest> requests = new
HashSet<CalculationRequest>();
@Override
public List<CalculationRequest> getRequests() {
return new ArrayList<CalculationRequest>(requests);
}
@Override
public void putRequest(CalculationRequest request) {
requests.add(request);
}
@Override
public void removeRequest(CalculationRequest request) {
requests.remove(request);
}
@Override
public long getNumRequests() {
return requests.size();
}
}
```

The tests all passed.

I next added another interesting test. To queue the same request multiple times:

```
@Test
public void testQueueRequestAndRetrievePendingMultipleSameRequests() {
calculator.deleteAllPendingRequests();
double dimension = 0;
calculator.queueCalculationRequest(ShapeName.CIRCLE, CalcType.CALC_AREA,
dimension);
calculator.queueCalculationRequest(ShapeName.CIRCLE, CalcType.CALC_AREA,
dimension);
calculator.queueCalculationRequest(ShapeName.CIRCLE, CalcType.CALC_AREA,
dimension);
List<CalculationRequest> requests = calculator.getAllPendingRequests();
assertNotNull(requests);
assertEquals(1,requests.size());
calculator.deleteAllPendingRequests();
requests = calculator.getAllPendingRequests();
assertNotNull(requests);
assertEquals(0,requests.size());
}
```

The test failed. Expected 1, but got back 3.

I went to CalculationRequest, and using Eclipse, auto-generated Getters and Setters, also toString(), but more importantly - hashCode() and equals().

Ran the test again. This time it passed.

Continued in next article....