Hi Examiners,

Very interesting multithreading question and I did enjoy it.

Following is an assumption I made in the question:

ASSUMPTION
Following 2 statements in the question sheet are not compatible with each other:
1. Where possible, the message groups should not be interleaved ... except where resources are idle and other work can be done
2. For two resources, when two messages are received, both messages are sent to the gateway.

In case 2 messages from the same group are received while there are 2 resources, should these 2 messages be processed one after another?

I am taking it that messages from the same group need to be processed serially.
So when 2 messages from the same group are received while there are 2 resources available, only 1 message will be sent and the other one is queued until the first one completes.
ASSUMPTION

File list:

- ResourceScheduler_lib - library needed to run the program
- instructions.xml - XML file containing the instructions to simulate operations between scheduler and gateway. It shall be used after adding “sendMessage” operations to it.
- instructions_example.xml - an example of the “ready to go” instructions.xml
- setup.properties - configuration to generate random “sendMessage” operations instructions
- ResourceScheduler.jar - runnable JAR
- ResourceScheduler_src.jar - source code
- logViewing.png - example of viewing the output of the program
- Sequence_Diagram.png - sequence diagram
- Class_Diagram.png - class diagram of gateway and scheduler

The remaining of this readme consists of following sections:

1. The structure of the Resource Scheduler program
    a. main components
        i. Gateway
        ii. MessageProcessor
        iii. Scheduler
        iv. Strategy

    b. other components
        i. NeglectedResourceRecoverer
        ii. Monitor
        iii. Thread Exception Handler
    c. main classes
2. How to test it?
    a. generating messages
    b. viewing the output
3. How to run it?
    a. preparing the XML file
    b. run it
    c. view the log

The structure of Resource Scheduler

Main Components

Gateway

Gateway could be used by more than one scheduler.
Gateway maintains a pool of resources which in this case, message processors.
It provides interfaces for schedulers to acquire and use the resource to process the message sent over. It also provides interfaces to increase or decrease resources at runtime.
The approach of increasing resource is simple, it just allocate new message processors. However, the decreasing of resource is a bit different.
The gateway, whenever there is resource released by a scheduler after process of message, compares the new designated size and the current one. If the resource size is to be shrunk, idle resource will not be allocated again and will be freed or say, excluded from the pool.

MessageProcessor

Each message processor has its own thread to process the message.
Upon completion, it will

- inform the scheduler about the completion of the message via a call back
- notify the gateway that itself (the message processor) is now idle and available for next task

ResourceScheduler

A Resource Scheduler runs in its own thread.
Resource Scheduler takes messages from different producers (it is the main() in this case) and send the messages to the gateway for process when following conditions are met

- there is message to be sent in its queue (with some strategy enforced, there might be no message suitable for sending even the queue is not empty)

- gateway is available (with idle resource and not shutdown)
- resource allocated to the scheduler by gateway

When there is no message or no resource available, scheduler will be suspended by wait(). And it will be notified whenever following events occur:

- message process completes
- new message arrives
- resource status change (e.g. released, increased)

A message group can be cancelled using the scheduler and no more message from that group will be sent to the gateway.

Strategy

Strategy is implemented using Strategy design pattern.
There are 2 strategy implemented in this exercise:

- SimpleStrategy - as simple as FIFO
- GroupPrioritisedStrategy
  Since messages from the same group should not be interlevaed if possible, groups are prioritised via the order of first message received from them. And therefore, even when there are messages in the queue, there might be no message to send to the gateway, according to the strategy.
  Therefore, a message can be sent if
  1. the group is not terminated
  2. the group is not cancelled
  3. no other messages from the same group is being processed at the moment
  4. no other messages from the same group is received before this one

  For example,
  There are 4 resources and 2 of them are being used for processing messages from group 1 and group 2 respectively. If there are messages from these 2 groups in the queue in scheduler, they are not to be sent until the messages of these 2 groups have been fully processed. And only if messages from other groups arrives at this time, the message will be sent to the gateway.

Other components

NeglectedResourceRecoverer

Since messages from the same group should not be interleaved if possible, the decision on which is the next message will not be made until a resource is allocated for the processor. That means, a scheduler will acquire a resource before it decides which message to send the next time.
However, what if a resource is allocated while the scheduler never comes to use it (e.g. scheduler crashes for some reason)?

A NeglectedResourceRecoverer, running as a daemon thread will be periodically checking if there is any resource allcoated for a scheduler from which it has not heard from for a certain period, then resource will be recovered and put back to the idle pool.
The gateway, in order to have this resource recover running, will have a keep track of the last communication with the scheduler. This is achieved via something like heartbeat().

Monitor

In order to give a better understanding of the current state of the program, including gateway and schedulers, 2 daemon threads periodically collect the status information of gateway and scheduler respectively and send them to either the log or console, or both.
Monitorable components (gateway and scheduler) implement an interface named Monitorable so the daemon thread will be able to, via this interface, capture the state of these components.
There are 2 options to collect this information, one is weak and the other one is strong.
The weak one collects information without putting a lock at the monitorable components, making that the information collected might be inconsistent, however it does not affect the performance much.
On the other hand, the strong one does put a lock at the monitorable components and is able to get a consistent state of it, and of course, at a cost of performance degradation.

Thread Exception Handler

Since scheduler and message processors have their own executing threads, exception coming from them are not catchable in main(). An exception handler is implemented to take care of this.

Main classes

Gateway

`org.vs.resourcescheduler.gateway.IGateway`
`org.vs.resourcescheduler.gateway.AbsGateway`
`org.vs.resourcescheduler.gateway.Gateway`

MessageProcessor

`org.vs.resourcescheduler.processor.IProcessor`
`org.vs.resourcescheduler.processor.AbsMessageProcessor`
`org.vs.resourcescheduler.processor.MessageProcessor`

Scheduler

`org.vs.resourcescheduler.scheduler.IResourceScheduler`
`org.vs.resourcescheduler.scheduler.AbsResourceScheduler`

```
org.vs.resourcescheduler.scheduler.ResourceScheduler
```

Strategy

```
org.vs.resourcescheduler.scheduler.strategy.IStrategy
org.vs.resourcescheduler.scheduler.strategy.SimpleStrategy
org.vs.resourcescheduler.scheduler.strategy.GroupPrioritisedStrategy
```

NeglectedResourceRecoverer

```
org.vs.resourcescheduler.gateway.Gateway$NeglectedResourceRecoverer
```

Monitor

```
org.vs.resourcescheduler.monitor.IMonitor
org.vs.resourcescheduler.monitor.DaemonMonitor
org.vs.resourcescheduler.monitor.IMonitorable
```

Thread Exception Handler

```
org.us.resourcescheduler.SimpleThreadFactory
```

How to test it?

Since the program is not an interactive one, instructions like send(message) need to be specified in an XML file in advance.

Generating Messages

To test this program, an XML file could be used as an input of the messages that are to be sent to the scheduler and gateway. An example is given in file instructions_example.xml

The XML has a schema like:

==========

```
<case>
  <config>
    <gateway size="2"/>
    <strategies>
      <strategy id="0" type="SimpleStrategy"/>
      <strategy id="1" type="GroupPrioritisedStrategy"/>
    </strategies>
    <schedulers>
```

```
      <scheduler id="0" type="ResourceScheduler" strategyId="1"/>
    </schedulers>
  </config>
  <instructions>
    <instruction subject="scheduler" id="0" action="setStrategy" strategyId="1"/>
  </instructions>
</case>
```
==========

In the <config> element of the XML file, following component could be configured:

- Gateway and its size;
- Strategy
- Scheduler

The other component is a list of <instruction> in element <instructions>.
In this element list, following instructions could be defined:

subject	**action**	**object**
main	wait	
main	sendMessage	scheduler
gateway	setSize	
gateway	shutdown	
scheduler	setStrategy	
scheduler	shutdown	
scheduler	cancel	

However, except for the wait operation, all these operations are non-blocking from the perspective of the main(), and they will be executed right after each other regardless what the workload of a message is. It therefore may seem that following example

==========
```
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”0”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”1”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”2”/>
<instruction subject="gateway" action="setSize" size=”5”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”3”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”4”/>
```
==========

is executed in a order like:

```
=======================
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”0”/>
<instruction subject="gateway" action="setSize" size=”5”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”1”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”2”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”3”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”4”/>
=======================
```

Because when messageId=”0” is being processed by gateway (after being sent to scheduler and then sent to gateway), all other messages are sent to the scheduler since they are not blocking operation and therefore the gateway’s setSize operation is executed also.

What can be done to simulate a gateway’s setSize operation, and something else like scheduler.setStrategy is:

```
=======================
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”0”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”1”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”2”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”3”/>
<instruction subject="main" schedulerId="0" action="sendMessage" messageId=”4”/>
<instruction subject="main" action=”wait” length=”10000”/>
<instruction subject="gateway" action="setSize" size=”5”/>
=======================
```

A “wait” operation here could delay the gateway’s setSize operation, but at which point it is executed is not precisely known / controlled as it is just a rough estimation.

Viewing the output

There is a dedicated logger configured for each of the components described above:

- Gateway - verbose information of gateway operations
- GatewayMonitor - periodically state information
- NeglectedResourceRecoverer - information when scanning / recovering neglected resources
- Scheduler - verbose information of scheduler operations
- SchedulerMonitor - periodically state information
- Processor - verbose information of message processors
- ThreadExceptionHandler - Exception information

Location of these log files can be found in the log4j.properties file.

How to run it?

Preparing the XML file

Because manually writing the XML file is not very funny, an InstructionGenerator is implemented to do this job. Briefly, it will

1. configure how many message groups are there, and how many message are there for each of these groups
2. generate the message as a list of instructions of send(message)
3. put the generated content (XML elements) in an pre-written XML file which has been prepared for other activities like configuring gateway and scheduler
4. run it.

Following is a more detailed breakdown of how to run it:

1. prepare the input messages
    a. configure setup.properties
    (example provided)
    There are 4 properties in the setup.properties file:
        i. output - this is the filename of the generated message file
        ii. groupSize - the value of this properties is a string separated by semi-colon.
        Each part of it specifies the size of a group
        and the number of parts implies the number of groups.
        e.g.
        groupSize=20;30;40;60;100
        indicates there are 5 groups in total, and they have 20, 30, 40, 60 and 100 messages respectively.
        iii. workloadUOM - each message is assigned with a random workload to simulate the work needed by the gateway to process it. This properties designates the Unit of Measure of the workload, for example, 500 milliseconds.
        iv. maxWorkloadUnit - This is the max workload unit that would be assigned to a message. The exactly time needed to process a message is the production of the UOM and this one.
        e.g.
        workloadUOM=500
        maxWorkloadUnit=5
        then the workload assigned to a message would be randomly picked among
        500, 1000, 1500, 2000, 2500
    b. generate the messages XML file
        i. run

```
$ java -jar ResourceScheduler.jar prepare $FILENAME
```

e.g.

```
$ java -jar ResourceScheduler.jar prepare setup.properties
```

2. edit the instructions.xml
   The XML files that contains messages has only messages, but no initialisations of other components that are needed to process them.
   There is an XML file named
   instructions.xml
   comes along with the package which can be used as a template.
   It already has some predefined gateway, strategy and scheduler and they are ready to use.
   Please place the content of the generated XML file between the
   <instructions> </instructions>
   elements in this instructions.xml.
   For a merged version of the XML file, please refer to
   instructions_example.xml

run it

```
$ java ResourceScheduler.jar run $XML_FILENAME
```

e.g.

```
$ java ResourceScheduler.jar run instructions_example.xml
```

or

```
$ java ResourceScheduler.jar run instructions.xml
```

View the log

It is more convenient to have multiple terminal console opened and each of them showing one of the logs via command

```
$ tail -f /tmp/ResourceScheduler/gateway.log
$ tail -f /tmp/ResourceScheduler/gatewayMonitor.log
$ tail -f /tmp/ResourceScheduler/gatewayNRR.log
$ tail -f /tmp/ResourceScheduler/scheduler.log
$ tail -f /tmp/ResourceScheduler/schedulerMonitor.log
$ tail -f /tmp/ResourceScheduler/processor.log
$ tail -f /tmp/ResourceScheduler/threadExceptionHandler.log
```