SOEN 423, Fall 2024, Assignment 2

Ties Schasfoort

22nd October 2024

Overall design:

For this assignment I implemented the distributed emergency response management system from Assignment 1 in CORBA using Java IDL. Most of the methods I use in this assignment are replicas from the code I wrote in A1, with the exception to the swapResource() method of course. Because these methods are basically the same as in A1, I won't discuss them in this documentation, for that I would refer to the documentation of A1. Instead, I will focus on the adjustments that were made to the code of A1 such that it fit the requirements of A2.

Differences:

Client setup:

Below is the code which would connect the client and ClientIDList in java RMI:

```
Registry registry = LocateRegistry.getRegistry("localhost", 1099);
ClientIDListInterface clientIDList = (ClientIDListInterface) registry.lookup("ClientIDList");
```

Here we connect the client to the ClientIDList by connecting to its portnumber. Then we get a reference to this list by looking up the remote object using its name, so that we can check if the id that is given by the client later on is already in use.

However, in CORBA we would do this:

```
String[] orbArgs = {"-ORBInitialHost", "localhost", "-ORBInitialPort", "1050"};
ORB orb = ORB.init(orbArgs, null);


org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);


// Access ClientIDList
ClientIDListInterface clientIDList = ClientIDListInterfaceHelper.narrow(ncRef.resolve_str("ClientIDList"));
```

Here we first set up the orb arguments, which are used to initialize the orb. Then we connect initialize the orb which connects the client to the orb at the server side on port 1050. After, we obtain a reference to the CORBA Name Service that allows us to look up objects by name. This is used to get a reference to the ClientIDList, like registry.lookup did in RMI.

This is how the client connects to its respective server (MTL, SHE, or QUE) in RMI:

```
Registry registryMain = LocateRegistry.getRegistry("localhost", port);
ServerInterface stub = (ServerInterface) registryMain.lookup(serverName);
```

We connect to the server like we connected to ClientIDList, but this time with another port number. Then we get a reference to the object by using lookup() again, such that the client is connected and can interact with the server.

This is how it is done in CORBA:

```
ServerInterface stub = ServerInterfaceHelper.narrow(ncRef.resolve_str(clientId.substring(0,
3).toUpperCase()));
```

We get a reference to the server by using the ncRef, which is used to access Name Service such that we can look up the server by its name, which is the prefix of the client's id. This way we can use the stub to interact with the server.

Server setup:

In RMI, we initialize the server's threads such that each one has one running for client-server communication and one for inter-server communication like this:

```
Thread serverMTL = new Thread(new ServerThread(7777,"MTL", 7877));
serverThreads.add(serverMTL);
serverMTL.start();
```

And this is the code that is executed when calling new ServerThread:

```
Registry registry = LocateRegistry.createRegistry(port);
Server server = new Server(udpPort, serverName.substring(0, 3));
registry.bind(serverName, server);
System.out.println("Server:" + serverName + "running on ports:" + port + "and" + udpPort);
server.startUDPListener();
```

In CORBA I implemented it like this for each server:

```
Thread serverMTL = new Thread(() -> {
   String[] orbArgs = {"-ORBInitialHost", "localhost", "-ORBInitialPort", "1050"};
   ORB orb = ORB.init(orbArgs, null);
   new ServerThread(7777, "MTL", 7877, orb).run();
});
serverThreads.add(serverMTL);
serverMTL.start();
```

As you can see, the only difference here is that the server first connects to the orb at port 1050, which is just like shown before, used to connect to the Name Service such that the client can look up the server.

And this is the code that is executed when calling new ServerThread:

```
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
rootPOA.the_POAManager().activate();

//Create the server object
ServerBase server = new ServerBase(orb,this.serverName,this.udpPort);
Object ref = rootPOA.servant_to_reference(server);

//This gives the server a name such that nameservice can be used to connect client to server
org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
NameComponent[] path = ncRef.to_name(this.serverName);
ncRef.rebind(path, ref);

System.out.println("Server: " + this.serverName + " is running on port: " + this.port + " and UDP port: " +
this.udpPort);

// Start the UDP listener in a separate thread
server.startUDPListener();

orb.run();
```

The difference with RMI here is that the server here first initializes its own POA. This POA is used to handle objects, in this case the line:

Object ref = rootPOA.servant_to_reference(server)

Here each server creates its own servant object (e.g., ServerBase server) and registers it with the server's POA. This ensures that each server has its own CORBA object reference that clients can use to invoke methods on that specific server.

Then ncRef.to_name(this.serverName); converts the server name into a path that can be used for naming.

rebind(path, ref) binds (or rebinds if the name already exists) the server's object reference to the name in the naming service. This way, when a client looks up the server's name, it gets in return the server object linked to that name. This ensures that the client can interact with its respective server.

Finally, orb.run() starts the orb's event loop. Once the orb is running, it waits for and processes incoming client requests.

Interfaces:

Since the interface are declared in the idl file, I declared both interfaces for the main server and the ClientIDList in one file. I created ResourceException to cover the exceptions thrown by methods. I also had to define StringSeq as a datatype since java 8 doesn't support List<String> as a return type.

```
interface ServerInterface {

    exception ResourceException {
       string message;  // Error message
    };
    // Responder methods
    string addResource(in string resourceID, in string resourceName, in long duration)
raises(ResourceException);
    string removeResource(in string resourceID, in long duration) raises(ResourceException);
    string listResourceAvailability(in string resourceName) raises(ResourceException);

    // Coordinator methods
    string requestResource(in string coordinatorID, in string resourceID, in long duration)
raises(ResourceException);
    string findResource(in string coordinatorID, in string resourceName) raises(ResourceException);
    string returnResource(in string coordinatorID, in string resourceID) raises(ResourceException);
    string swapResource(in string coordinatorID, in string oldResourceID, in string oldResourceType, in string
newResourceID,
    in string newResourceType) raises(ResourceException);
    void addToQueue(in string clientId, in string resourceId);
```

```
};



typedef sequence<string> StringSeq;
interface ClientIDListInterface {
   void addClientID(in string clientId);
   void removeClientID(in string clientId);
   StringSeq getClientIDs();
};
```

Running the code:

When I ran the code in RMI I just ran the following files in the following order:

1. ClientIDListServer.java and Server.java. The order doesn't matter.
2. Then I ran Client.java.

For CORBA this was a bit more complicated:

1. Run "idlj -fall ServerInterface.idl" to compile the interfaces.
2. Run "tnameserv -ORBInitialPort 1050" to start the Name Service such that it can be connected to on port 1050.
3. Run ServerBase.java and ClientIDListServer.java. Order doesn't matter.
4. Run Client.java.

Test cases and edge cases for swapResource():

Test cases:

First a responder would do this:

```
Enter your unique ID with 3 prefix letters MTL, SHE, or QUE followed by R or C meaning responder or coordinator, followed by 4 digits:
mtlr1111
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (add, rem, or list)?
add
ID (like MTL1111 or SHE1384):
she1111
Name (ambulance, firetruck, or personnel):
ambulance
Duration:
5
Resource added successfully
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (add, rem, or list)?
add
ID (like MTL1111 or SHE1384):
she1234
Name (ambulance, firetruck, or personnel):
ambulance
Duration:
5
Resource added successfully
```

Then the standard swap operation performed by the coordinator would be successful as shown below:

```
Enter your unique ID with 3 prefix letters MTL, SHE, or QUE followed by R or C meaning responder or coordinator, followed by 4 digits:
mtlc1111
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (req, find, return, or swap)?
req
ID (like MTL1111 or SHE1384):
she1111
Duration:
5
Resource was given
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (req, find, return, or swap)?
swap
Old resourceId (like MTL1111 or SHE1384):
she1111
New resourceId (like MTL1111 or SHE1384):
she1234
Successfully swapped resources
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (req, find, return, or swap)?
find
Name (ambulance, firetruck, or personnel):
ambulance
ambulance - she1234 1
Enter 'e' to disconnect or other key to continue with operations:
```

Edge cases:

What would happen if the coordinator passes the same resource ids to the swap operation? Having the responder do the same operations as in the test case above, the coordinator would do this:

```
Enter your unique ID with 3 prefix letters MTL, SHE, or QUE followed by R or C meaning responder or coordinator, followed by 4 digits:
mtlc1111
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (req, find, return, or swap)?
req
ID (like MTL1111 or SHE1384):
she1111
Duration:
5
Resource was given
Enter 'e' to disconnect or other key to continue with operations:
swa
What operation (req, find, return, or swap)?
swap
Old resourceId (like MTL1111 or SHE1384):
she1111
New resourceId (like MTL1111 or SHE1384):
she1111
No success in swapping resources
```

If coordinator tries swapping resources it already holds and are not available on the server:

```
Enter your unique ID with 3 prefix letters MTL, SHE, or QUE followed by R or C meaning responder or coordinator, followed by 4 digits:
mtlc1111
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (req, find, return, or swap)?
req
ID (like MTL1111 or SHE1384):
mtl1111
Duration:
5
Resource was given
Enter 'e' to disconnect or other key to continue with operations:
s
What operation (req, find, return, or swap)?
req
ID (like MTL1111 or SHE1384):
she1111
Duration:
5
Resource was given
Enter 'e' to disconnect or other key to continue with operations:
a
What operation (req, find, return, or swap)?
swap
Old resourceId (like MTL1111 or SHE1384):
mtl1111
New resourceId (like MTL1111 or SHE1384):
she1111
No success in swapping resources
```

Concurrency:

Besides the improvements I made to concurrency by widening the critical sections as explained in issue 2 below, I also added the swapresource method in ConcurrencyTest.java to test that method's concurrency. The result can be seen in the

pictures below. These show that concurrency is handled, since after requesting a resource, the client is always successful in swapping the resource. This means that even though multiple clients are performing operations simultaneously, the system properly synchronizes the access to shared resources, ensuring that a swap can only happen if the requested resource is available and the swap conditions are met. This demonstrates that the server is correctly handling concurrent requests to avoid race conditions or data corruption.

For example, if one client requests a resource and another client attempts to add or remove a resource at the same time, the system ensures that these actions do not interfere with each other.

```
Client MTLR1005remove: Resource couldn't be removed
Client MTLR1005 add1: Resource added successfully
Client MTLR1005 add2: Resource added successfully
Client MTLR1005 req: Resource was given
Client MTLR1005 swap: Successfully swapped resources
Client MTLR1001remove: Resource couldn't be removed
Client MTLR1001 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1001 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1001 req: Resource was given
Client MTLR1001 swap: Successfully swapped resources
Client MTLR1002remove: Resource couldn't be removed
Client MTLR1002 add1: Duration increased successfully
Client MTLR1002 add2: Duration increased successfully
Client MTLR1002 req: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1002 swap: No success in swapping resources
Client MTLR1008remove: Resource couldn't be removed
Client MTLR1008 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1008 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1008 req: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1008 swap: No success in swapping resources
Client MTLR1004remove: Resource couldn't be removed
Client MTLR1004 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1004 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1004 req: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1004 swap: No success in swapping resources
Client MTLR1007remove: Resource couldn't be removed
Client MTLR1007 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1007 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1007 req: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1007 swap: No success in swapping resources
Client MTLR1000remove: Resource couldn't be removed
Client MTLR1000 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1000 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1000 req: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1000 swap: No success in swapping resources
Client MTLR1006remove: Resource couldn't be removed
Client MTLR1006 add1: Resource couldn't be added since given duration was lower than duration for already present resource
```

```
Client MTLR1006 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1006 req: Resource was given
Client MTLR1006 swap: Successfully swapped resources
Client MTLR1009remove: Resource couldn't be removed
Client MTLR1009 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1009 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1009 req: Resource was given
Client MTLR1009 swap: Successfully swapped resources
Client MTLR1003remove: Resource couldn't be removed
Client MTLR1003 add1: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1003 add2: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1003 req: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1003 swap: No success in swapping resources
All clients finished.
```

Issues:

Issue 1:

IOExceptions weren't available in java 8, so I had to translate the exception to the ResourceException defined in the idl interface file. I translated it in the server code like this, where sendUDPMessage could possible throw an IOException due to its socket use:

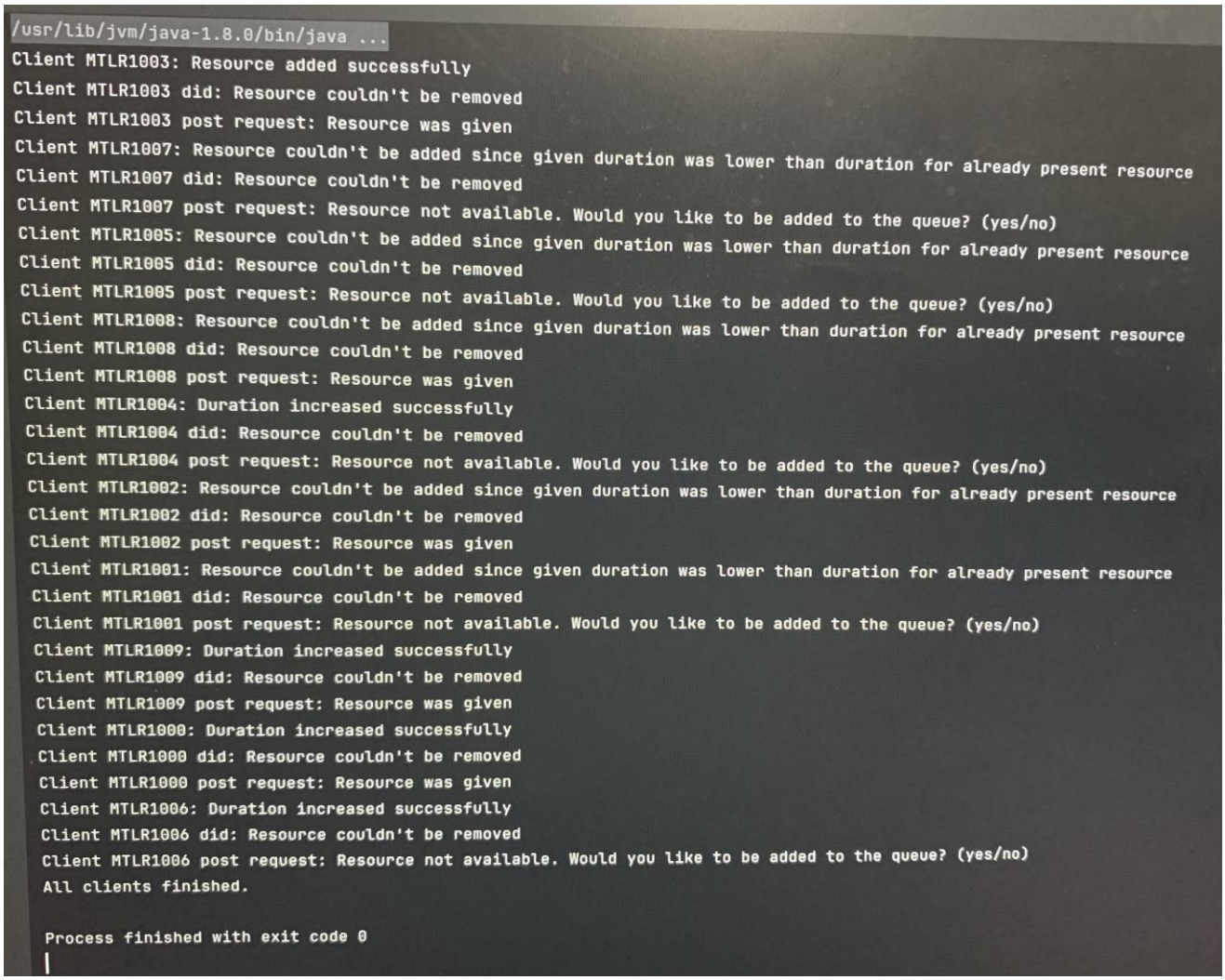
```
try{
   String response = sendUDPMessage("listResources:" + resourceName, "localhost", port);


}
catch (IOException e) { //Translate IOException to ResourceException
   throw new ResourceException("Network error while adding resource: " + e.getMessage());
}
```

Issue 2:

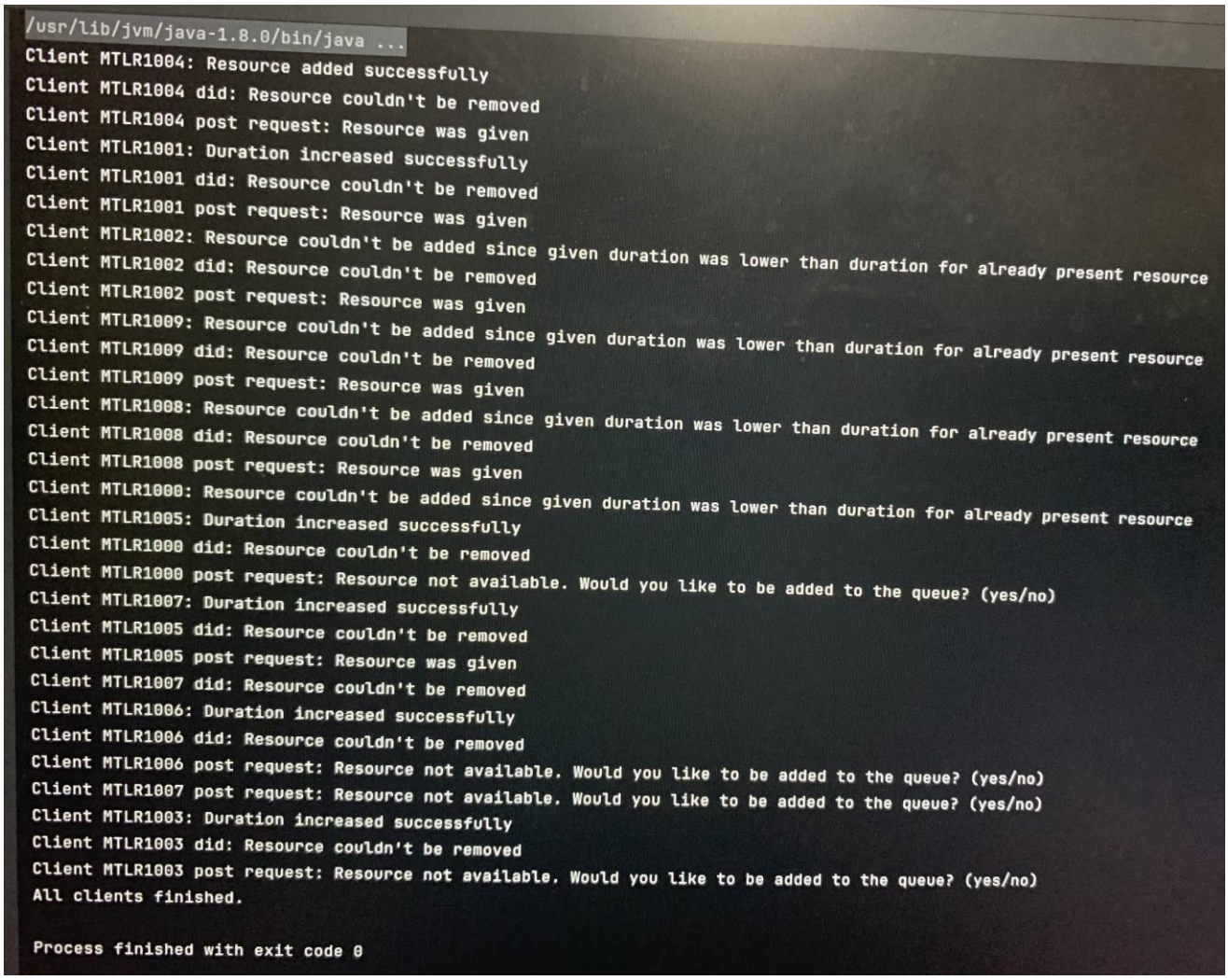
Implementing concurrency.

I added try-finally blocks around the sections where resources where locked, this way the lock is ensured to be released. I have also increased the scope of the critical sections to ensure that the locks stay in the possession of the client while they're changing the shared resource and no one else can access them. Below I have two pictures showing what would happen if I ran the ConcurrencyTest.java file before and after implementing these changes. It can be concluded that it worked in handling concurrency since before the changes, more than two clients were able to acquire the same resource. Even though only two should be able to since the resource is first added, then increased its duration due to another client trying to add the same resource. Then requesting half of the duration should be possible only twice, which is the case in the picture which reflects the results of adding the try-finally blocks and enlarging the critical sections.

Pre:


/usr/lib/jvm/java-1.8.0/bin/java ...
Client MTLR1003: Resource added successfully
Client MTLR1003 did: Resource couldn't be removed
Client MTLR1003 post request: Resource was given
Client MTLR1007: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1007 did: Resource couldn't be removed
Client MTLR1007 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1005: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1005 did: Resource couldn't be removed
Client MTLR1005 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1008: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1008 did: Resource couldn't be removed
Client MTLR1008 post request: Resource was given
Client MTLR1004: Duration increased successfully
Client MTLR1004 did: Resource couldn't be removed
Client MTLR1004 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1002: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1002 did: Resource couldn't be removed
Client MTLR1002 post request: Resource was given
Client MTLR1001: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1001 did: Resource couldn't be removed
Client MTLR1001 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1009: Duration increased successfully
Client MTLR1009 did: Resource couldn't be removed
Client MTLR1009 post request: Resource was given
Client MTLR1000: Duration increased successfully
Client MTLR1000 did: Resource couldn't be removed
Client MTLR1000 post request: Resource was given
Client MTLR1006: Duration increased successfully
Client MTLR1006 did: Resource couldn't be removed
Client MTLR1006 post request: Resource not available. Would you like to be added to the queue? (yes/no)
All clients finished.
Process finished with exit code 0

Post:


/usr/lib/jvm/java-1.8.0/bin/java ...
Client MTLR1004: Resource added successfully
Client MTLR1004 did: Resource couldn't be removed
Client MTLR1004 post request: Resource was given
Client MTLR1001: Duration increased successfully
Client MTLR1001 did: Resource couldn't be removed
Client MTLR1001 post request: Resource was given
Client MTLR1002: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1002 did: Resource couldn't be removed
Client MTLR1002 post request: Resource was given
Client MTLR1009: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1009 did: Resource couldn't be removed
Client MTLR1009 post request: Resource was given
Client MTLR1008: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1008 did: Resource couldn't be removed
Client MTLR1008 post request: Resource was given
Client MTLR1000: Resource couldn't be added since given duration was lower than duration for already present resource
Client MTLR1005: Duration increased successfully
Client MTLR1000 did: Resource couldn't be removed
Client MTLR1000 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1007: Duration increased successfully
Client MTLR1005 did: Resource couldn't be removed
Client MTLR1005 post request: Resource was given
Client MTLR1007 did: Resource couldn't be removed
Client MTLR1006: Duration increased successfully
Client MTLR1006 did: Resource couldn't be removed
Client MTLR1006 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1007 post request: Resource not available. Would you like to be added to the queue? (yes/no)
Client MTLR1003: Duration increased successfully
Client MTLR1003 did: Resource couldn't be removed
Client MTLR1003 post request: Resource not available. Would you like to be added to the queue? (yes/no)
All clients finished.
Process finished with exit code 0