Dheeraj B17041
Vipul B17069

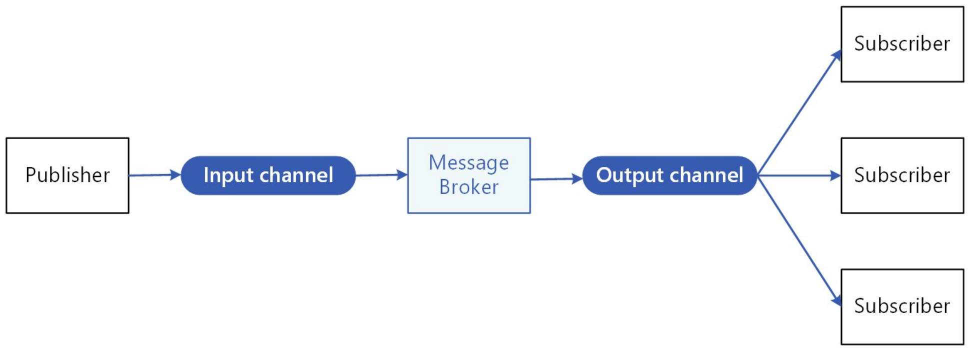
CS592
Reactive Design Patterns
Project Report

Publisher Subscriber Pattern

GitHub Repository: ***Publisher-Subscriber-Pattern***

Aim:

1. Enable an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers, using the Publisher Subscriber Pattern.
2. Enable master-slave configuration on the “Message Broker” Server.
3. Test the results using a verifier.



Work Distribution:

Dheeraj	Design and Implementation of Server, Subscribers and Publishers.
Vipul	Design and Implementation of Verifier along with the implementation of Server.

Implementation Details of Server-Subscriber-Publisher:

1. Communication is done using RMI.
2. Every server starts as a Slave and if there is no master, it promotes itself to Master.
3. Whenever a Slave promotes itself to Master, a new Slave is spawned.
4. The publisher object is not exported because it will not get any input from the server or Subscriber.
5. The UUID of each Publisher and Subscriber is of the form *pid@hostname*.
6. Each ReqID is of the form pid@hostname<reqCounter>.

Implementation Details of Verifier:

1. Verifier first creates a random input file and sends this input file to the various components from STDIN.
2. There is no guarantee that requests will be processed in the same order they are sent, so the order of execution is printed in the server logs.
3. Next, we create expected output files based on the contents of server.txt and this file is diffed with the real output of subscribes.

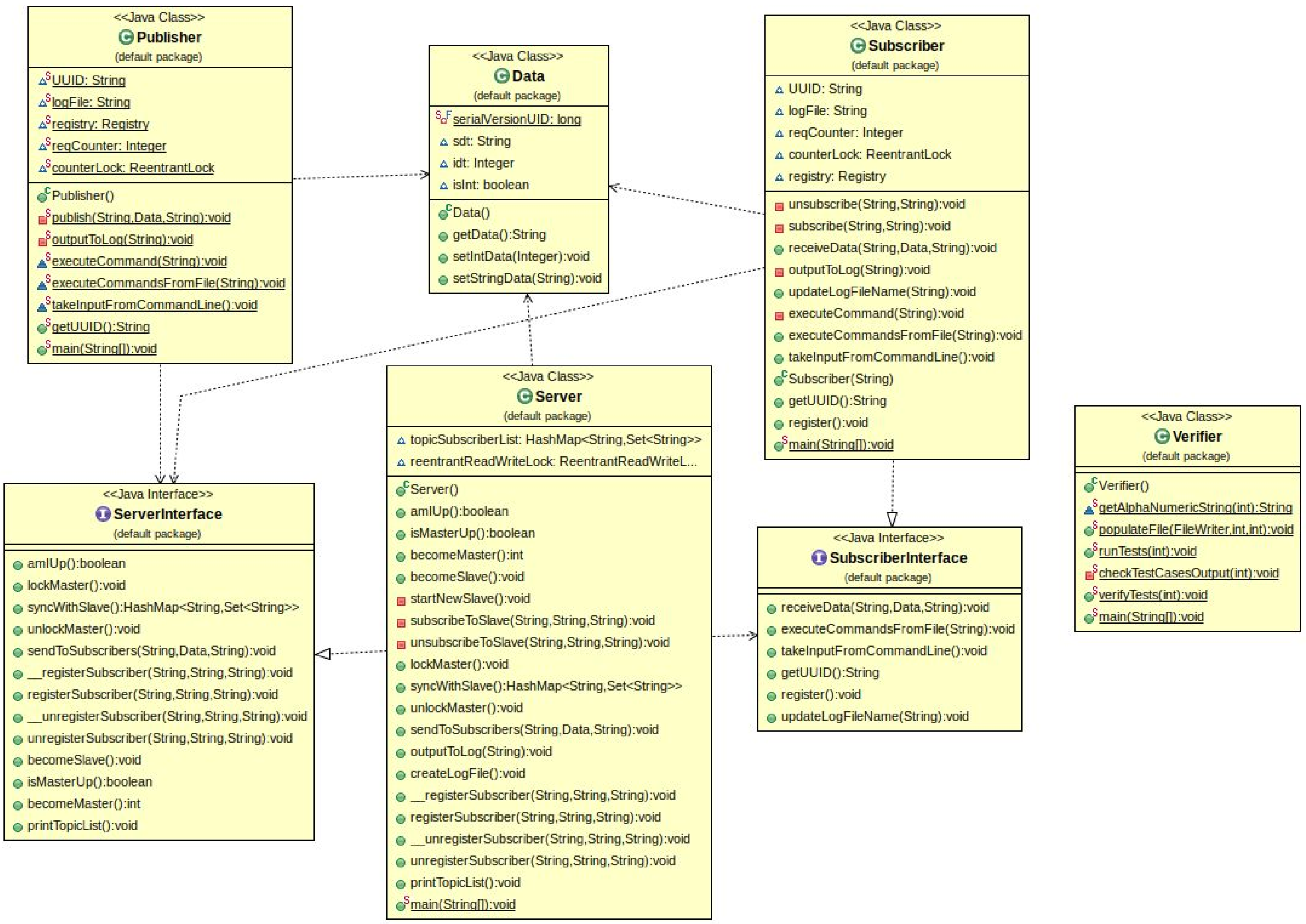
Patterns Used:

1. Publisher Subscriber Pattern - This is the overall theme of this project.
2. Heartbeat Pattern - The slave will ping the master to see if it is up every t_h second. If it detects that master is down, it automatically takes over as master and spawns another slave.
3. Master-Slave Replication Pattern
4. Event Stream Pattern - Verifier uses logs of Publisher, Server and Subscriber to simulate and verify the correctness

UML Diagram of System: Generated using NetBeans

Legend:

-	Red Box -	Function with private visibility.
-	Filled Blue Triangle -	Function with default visibility.
-	Green Circle -	Function with public visibility.
-	Unfilled Triangle -	Data members.



<<Java Class>>
Publisher
(default package)
UUID: String
logFile: String
registry: Registry
reqCounter: Integer
counterLock: ReentrantLock
Publisher()
publish(String,Data,String):void
outputToLog(String):void
executeCommand(String):void
executeCommandsFromFile(String):void
takeInputFromCommandLine():void
getUUID():String
main(String[]):void
<<Java Class>>
Data
(default package)
serialVersionUID: long
sdt: String
idt: Integer
isInt: boolean
Data()
getData():String
setIntData(Integer):void
setStringData(String):void
<<Java Class>>
Subscriber
(default package)
UUID: String
logFile: String
reqCounter: Integer
counterLock: ReentrantLock
registry: Registry
unsubscribe(String,String):void
subscribe(String,String):void
receiveData(String,Data,String):void
outputToLog(String):void
updateLogFileName(String):void
executeCommand(String):void
executeCommandsFromFile(String):void
takeInputFromCommandLine():void
Subscriber(String)
getUUID():String
register():void
main(String[]):void
<<Java Class>>
Server
(default package)
topicSubscriberList: HashMap<String,Set<String>>
reentrantReadWriteLock: ReentrantReadWriteL...
Server()
amIUp():boolean
isMasterUp():boolean
becomeMaster():int
becomeSlave():void
startNewSlave():void
subscribeToSlave(String,String,String):void
unsubscribeToSlave(String,String,String):void
lockMaster():void
syncWithSlave():HashMap<String,Set<String>>
unlockMaster():void
sendToSubscribers(String,Data,String):void
outputToLog(String):void
createLogFile():void
__registerSubscriber(String,String,String):void
registerSubscriber(String,String,String):void
__unregisterSubscriber(String,String,String):void
unregisterSubscriber(String,String,String):void
printTopicList():void
main(String[]):void
<<Java Interface>>
ServerInterface
(default package)
amIUp():boolean
lockMaster():void
syncWithSlave():HashMap<String,Set<String>>
unlockMaster():void
sendToSubscribers(String,Data,String):void
__registerSubscriber(String,String,String):void
registerSubscriber(String,String,String):void
__unregisterSubscriber(String,String,String):void
unregisterSubscriber(String,String,String):void
becomeSlave():void
isMasterUp():boolean
becomeMaster():int
printTopicList():void
<<Java Interface>>
SubscriberInterface
(default package)
receiveData(String,Data,String):void
executeCommandsFromFile(String):void
takeInputFromCommandLine():void
getUUID():String
register():void
updateLogFileName(String):void
<<Java Class>>
Verifier
(default package)
Verifier()
getAlphaNumericString(int):String
populateFile(FileWriter,int,int):void
runTests(int):void
checkTestCasesOutput(int):void
verifyTests(int):void
main(String[]):void

Small Test Case Invocation:

Input			**Output**		
Subscriber-1	**Subscriber-2**	**Publisher**	**Subscriber-1**	**Subscriber-2**	**Publisher**
S topic100	S topic100	-	Subscribe @topic100	Subscribe @topic100	-
-	-	topic100 Hi_first_message!	Received @topic100 Data: Hi_first_message! with reqID: 85618@dj-hp 0	Received @topic100 Data: Hi_first_message! with reqID: 85618@dj-hp 0	Publishing @topic100 Data: Hi_first_message! with Request ID: 85618@dj-hp 0
S topic2	-	-	Subscribe @topic2	-	-
		topic2 Hi_Sending_at_topic2!	Received @topic2 Data: Hi_Sending_at_topic2! with reqID: 85618@dj-hp 1		Publishing @topic2 Data: Hi_Sending_at_topic2! with Request ID: 85618@dj-hp 1
U topic100	-	-	UnSubscribe @topic100	-	-
-	-	topic100 third_message.	-	Received @topic100 Data: third_message. with reqID: 85618@dj-hp 2	Publishing @topic100 Data: third_message. with Request ID: 85618@dj-hp 2
SIGINT	SIGINT	SIGINT	-	-	-

Automated Test Cases in Verifier:

Number of Subscribers Used = 10	(given as command line argument to Verifier.java)
Number of Topics Used = 10	(can be changed in Verifier.java main fxn)
Lines in input.txt Test Case = 200	(can be changed in Verified.java populateFile fxn)

Conclusion:

1. Created a java application in which a publisher can announce events to multiple interested subscribers asynchronously, without coupling the senders to the receivers, using the Publisher Subscriber Pattern.
2. Enable master-slave configuration on the Server.
3. Tested the results manually and also using a Verifier script.