

Sep 22, 15 8:32

MainClass.java

Page 1/2

```

1  package stockProvider;
2
3  import java.util.Iterator;
4  import java.util.UUID;
5  import java.util.ArrayList;
6  import java.lang.Math;
7  import java.lang.System;
8  import java.util.Random;
9  import java.util.concurrent.TimeoutException;
10 import java.io.IOException;
11 import java.util.Random;
12
13 import com.rabbitmq.client.ConnectionFactory;
14 import com.rabbitmq.client.Connection;
15 import com.rabbitmq.client.Channel;
16 import org.apache.commons.lang3.SerializationUtils;
17
18 import stockManager.StockDB;
19 import common.Product;
20 import configParser.ConfigParser;
21 import logger.Logger;
22 import logger.LogLevel;
23
24 public class MainClass {
25     public MainClass(String[] argv) throws IllegalArgumentException,
26         IOException {
27         config_ = ConfigParser.getInstance();
28         logger_ = Logger.getInstance();
29
30         config_.init(argv[1]);
31         this.initLogger(argv[0]);
32
33         String stockDBFile = config_.get("STOCK", "stock-db-file");
34         stockDB_ = new StockDB(stockDBFile);
35     }
36
37     public static void main(String[] argv) {
38         ConfigParser config = ConfigParser.getInstance();
39         Logger logger = Logger.getInstance();
40         try {
41             MainClass app = new MainClass(argv);
42             app.increaseStock();
43         }
44         catch (IllegalArgumentException e) {
45             // We couldn't open the logger. Just exit
46             System.out.println(e);
47             System.exit(-1);
48         }
49         catch (IOException e) {
50             logger.log(LogLevel.ERROR, e.toString());
51         }
52     }
53
54     private void initLogger(String processNumber)
55     throws IllegalArgumentException {
56         String logFileName = config_.get("MAIN", "log-file");
57         String logLevel = config_.get("MAIN", "log-level");
58
59         Logger logger = Logger.getInstance();
60         logger.init(logFileName, LogLevel.parse(logLevel));
61         logger.setPrefix("[STOCK_PROVIDER " + processNumber + "]");
62         logger.log(LogLevel.DEBUG, "Process started");
63     }
64
65     public void increaseStock() throws IllegalArgumentException,
66         IOException {
67         long globalIncrease = Long.parseLong(config_.get("STOCK-PROVIDER",
68             "global-increase"));
69         long type1Increase = Long.parseLong(config_.get("STOCK-PROVIDER",
70             "type-1-increase"));
71         long type2Increase = Long.parseLong(config_.get("STOCK-PROVIDER",
72             "type-2-increase"));
73         long type3Increase = Long.parseLong(config_.get("STOCK-PROVIDER",

```

Sep 22, 15 8:32

MainClass.java

Page 2/2

```

74         "type-3-increase"));
75         long type4Increase = Long.parseLong(config_.get("STOCK-PROVIDER",
76             "type-4-increase"));
77         long type5Increase = Long.parseLong(config_.get("STOCK-PROVIDER",
78             "type-5-increase"));
79
80         stockDB_.increaseStock(Product.TYPE_1, globalIncrease + type1Increase);
81         stockDB_.increaseStock(Product.TYPE_2, globalIncrease + type2Increase);
82         stockDB_.increaseStock(Product.TYPE_3, globalIncrease + type3Increase);
83         stockDB_.increaseStock(Product.TYPE_4, globalIncrease + type4Increase);
84         stockDB_.increaseStock(Product.TYPE_5, globalIncrease + type5Increase);
85     }
86
87     private Logger logger_;
88     private ConfigParser config_;
89     private StockDB stockDB_;
90 }

```

Sep 21, 15 5:43

StockManager.java

Page 1/2

```

1 package stockManager;
2
3 import com.rabbitmq.client.ConnectionFactory;
4 import com.rabbitmq.client.Connection;
5 import com.rabbitmq.client.Channel;
6 import com.rabbitmq.client.Consumer;
7 import com.rabbitmq.client.DefaultConsumer;
8 import com.rabbitmq.client.Envelope;
9 import com.rabbitmq.client.AMQP;
10 import org.apache.commons.lang3.SerializationUtils;
11
12 import configParser.ConfigParser;
13 import common.Order;
14 import common.OrderState;
15 import logger.Logger;
16 import logger.LogLevel;
17 import stockManager.StockDB;
18
19 import java.io.IOException;
20
21
22 public class StockManager extends DefaultConsumer {
23     public StockManager(Channel channel) throws IllegalArgumentException,
24         IOException {
25         super(channel);
26         logger_ = Logger.getInstance();
27         config_ = ConfigParser.getInstance();
28         this.initQueues();
29
30         String stockDBFile = config_.get("STOCK", "stock-db-file");
31         stockDB_ = new StockDB(stockDBFile);
32     }
33
34     @Override
35     public void handleDelivery(String consumerTag,
36         Envelope envelope,
37         AMQP.BasicProperties properties,
38         byte[] body) throws IOException {
39         Order newOrder = (Order) SerializationUtils.deserialize(body);
40         logger_.log(LogLevel.TRACE, "Order received: " + newOrder.toString());
41
42
43         elapsedTime_ = System.currentTimeMillis();
44         boolean enoughStock = stockDB_.decreaseStock(newOrder.productType(),
45             newOrder.amount());
46         elapsedTime_ = System.currentTimeMillis() - elapsedTime_;
47         logger_.log(LogLevel.NOTICE, "decreaseStock. Time: "
48             + elapsedTime_ + " ms.");
49
50         if (enoughStock) {
51             newOrder.state(OrderState.ACCEPTED);
52         }
53         else {
54             newOrder.state(OrderState.REJECTED);
55         }
56
57         logger_.log(LogLevel.INFO, "Order processed: "
58             + newOrder.toStringShort() + ". Sending it to the OrderManager.");
59
60         body = SerializationUtils.serialize(newOrder);
61         this.getChannel().basicPublish("", orderManagerQueueName_, null, body);
62     }
63
64     /**
65      * @brief Declare the queues. This is necessary because maybe they have not
66      * been created yet
67      */
68     private void initQueues() throws IOException {
69         Channel channel = this.getChannel();
70
71         orderManagerQueueName_ = config_.get("QUEUES", "order-manager-queue");
72         channel.queueDeclare(orderManagerQueueName_,
73             false,

```

Sep 21, 15 5:43

StockManager.java

Page 2/2

```

74             false,
75             false,
76             null);
77     }
78
79     private Logger logger_;
80     private ConfigParser config_;
81     private String orderManagerQueueName_;
82     private StockDB stockDB_;
83
84     // For performance stats
85     private long elapsedTime_;
86 }

```

Sep 22, 15 8:32

StockDB.java

Page 1/3

```

1 package stockManager;
2
3 import java.nio.channels.OverlappingFileLockException;
4 import java.lang.System;
5 import java.io.EOFException;
6 import java.nio.ByteBuffer;
7 import java.nio.channels.FileLock;
8 import java.util.Map;
9 import java.io.IOException;
10 import java.io.RandomAccessFile;
11 import java.nio.channels.FileChannel;
12 import java.io.FileOutputStream;
13 import java.io.File;
14 import java.util.HashMap;
15
16 import common.Product;
17 import logger.Logger;
18 import logger.LogLevel;
19
20 public class StockDB {
21     public StockDB(String dbFilePath) throws IOException {
22         logger_ = Logger.getInstance();
23
24         File file = new File(dbFilePath);
25         // Taken from the JavaDocs
26         // http://docs.oracle.com/javase/7/docs/api/java/io/File.
27         // html#createNewFile()
28         // Atomically creates a new, empty file named by this abstract
29         // pathname if and only if a file with this name does not yet
30         // exist. The check for the existence of the file and the creation
31         // of the file if it does not exist are a single operation that is
32         // atomic with respect to all other filesystem activities that
33         // might affect the file.
34         file.createNewFile();
35
36         // http://docs.oracle.com/javase/7/docs/api/java/io/
37         // RandomAccessFile.html#mode
38         // The "rwd" mode can be used to reduce the number of I/O operations
39         // performed. Using "rwd" only requires updates to the file's content
40         // to be written to storage; using "rws" requires updates to both the
41         // file's content and its metadata to be written, which generally
42         // requires at least one more low-level I/O operation.
43         file_ = new RandomAccessFile(dbFilePath, "rwd");
44         FileLock lock = file_.getChannel().lock();
45
46         if (file.length() == 0) {
47             this.createEmptyStockFile();
48             logger_.log(LogLevel.WARNING,
49                 "StockDB file doesn't exists."
50                 + " Proceed to create it. StockDB file: " + dbFilePath);
51         }
52         lock.release();
53     }
54
55     private void createEmptyStockFile() throws IOException {
56         HashMap<Product, Long> map = new HashMap<Product, Long>();
57         for (Product product : Product.values()) {
58             map.put(product, new Long(10000));
59         }
60
61         for (Map.Entry<Product, Long> entry : map.entrySet()) {
62             String key = String.format("%-10s", entry.getKey().toString());
63             file_.write(key.getBytes());
64
65             ByteBuffer b = ByteBuffer.allocate(8);
66             b.putLong(entry.getValue());
67             file_.write(b.array());
68         }
69     }
70
71     // This method is very long, but don't split it in functions because
72     // to performance problems
73     public boolean decreaseStock(Product product, Long amount)

```

Sep 22, 15 8:32

StockDB.java

Page 2/3

```

74     throws IOException {
75         byte[] buffer = new byte[PRODUCT_KEY_MAX_SIZE];
76         file_.seek(0);
77
78         try {
79             while(true) {
80                 int readBytes = file_.read(buffer, 0, PRODUCT_KEY_MAX_SIZE);
81                 if (readBytes == -1) {
82                     // lock.release();
83                     return false;
84                 }
85
86                 Product key = Product.valueOf(new String(buffer).trim());
87                 if (key != product) {
88                     // Jump to the next entry
89                     file_.skipBytes(Long.BYTES);
90                     continue;
91                 }
92
93                 // Product found, proceed to update value
94                 // Read the amount of the stock to update it, and go back to
95                 // the same position
96                 file_.read(buffer, 0, Long.BYTES);
97                 file_.seek(file_.getFilePointer() - Long.BYTES);
98
99                 // Check if there is stock of the file
100                 ByteBuffer b = ByteBuffer.wrap(buffer);
101                 long productStock = b.getLong();
102                 if (productStock < amount) {
103                     logger_.log(LogLevel.WARNING, "Order cannot be accepted. "
104                         + "Not enough stock of product " + product.toString()
105                         + ". ProductStock: " + productStock
106                         + " - OrderAmount: " + amount);
107                     return false;
108                 }
109
110                 // There is stock, update the StockDB
111                 long newStock = productStock - amount;
112                 ByteBuffer longBuf = ByteBuffer.allocate(Long.BYTES);
113                 longBuf.putLong(newStock);
114
115                 // Just lock the part of the file to me modified
116                 FileLock lock = file_.getChannel().lock(file_.getFilePointer(),
117                                                             Long.BYTES,
118                                                             false);
119
120                 file_.write(longBuf.array());
121                 lock.release();
122
123                 logger_.log(LogLevel.DEBUG, "Decreasing stock of product "
124                     + product.toString() + ". PreviousStock: "
125                     + productStock + " - UpdatedStock: " + newStock);
126                 break;
127             }
128         } catch (EOFException e) {
129             // If this happen, then the product does not exists and we have
130             // a bug in the system. ABORT!
131             logger_.log(LogLevel.ERROR, "Product does not exists. Product: "
132                 + product.toString());
133             System.exit(-1);
134         }
135
136         // lock.release();
137         return true;
138     }
139
140
141     // This method is very long, but don't split it in functions because
142     // to performance problems
143     public boolean increaseStock(Product product, Long amount)
144     throws IOException {
145         byte[] buffer = new byte[PRODUCT_KEY_MAX_SIZE];
146         file_.seek(0);

```

Sep 22, 15 8:32

StockDB.java

Page 3/3

```

147
148
149     try {
150         while(true) {
151             int readBytes = file_.read(buffer, 0, PRODUCT_KEY_MAX_SIZE);
152             if (readBytes == -1) {
153                 // lock.release();
154                 return false;
155             }
156
157             Product key = Product.valueOf(new String(buffer).trim());
158             if (key != product) {
159                 // Jump to the next entry
160                 file_.skipBytes(Long.BYTES);
161                 continue;
162             }
163
164             // Product found, proceed to update value
165             // Read the amount of the stock to update it, and go back to
166             // the same position
167             file_.read(buffer, 0, Long.BYTES);
168             file_.seek(file_.getFilePointer() - Long.BYTES);
169
170             // Check if there is stock of the file
171             ByteBuffer b = ByteBuffer.wrap(buffer);
172             long productStock = b.getLong();
173
174             // There is stock, update the StockDB
175             long newStock = productStock + amount;
176             ByteBuffer longBuf = ByteBuffer.allocate(Long.BYTES);
177             longBuf.putLong(newStock);
178
179             // Just lock the part of the file to me modified
180             FileLock lock = file_.getChannel().lock(file_.getFilePointer(),
181                                                     Long.BYTES,
182                                                     false);
183
184             file_.write(longBuf.array());
185             lock.release();
186
187             logger_.log(LogLevel.NOTICE, "Increasing stock of product "
188                     + product.toString() + ". PreviousStock: "
189                     + productStock + " - UpdatedStock: " + newStock);
190             break;
191         }
192     } catch (EOFException e) {
193         // If this happen, then the product does not exists and we have
194         // a bug in the system. ABORT!
195         logger_.log(LogLevel.ERROR, "Product does not exists. Product: "
196                 + product.toString());
197         System.exit(-1);
198     }
199
200     // lock.release();
201     return true;
202 }
203
204 private Logger logger_;
205 private RandomAccessFile file_;
206 private static final int PRODUCT_KEY_MAX_SIZE = 10;

```

Sep 20, 15 20:01

MainClass.java

Page 1/1

```

1  package stockManager;
2
3  // Program includes
4  import configParser.ConfigParser;
5  import logger.Logger;
6  import logger.LogLevel;
7
8  // External libraries includes
9  import com.rabbitmq.client.ConnectionFactory;
10 import com.rabbitmq.client.Connection;
11 import com.rabbitmq.client.Channel;
12 import com.rabbitmq.client.Consumer;
13 import com.rabbitmq.client.DefaultConsumer;
14 import com.rabbitmq.client.Envelope;
15 import com.rabbitmq.client.AMQP;
16
17 // Java includes
18 import java.lang.IllegalArgumentException;
19 import java.io.IOException;
20 import java.util.concurrent.TimeoutException;
21
22 public class MainClass {
23     public static void main(String[] argv) {
24         ConfigParser config = ConfigParser.getInstance();
25
26         Logger logger = Logger.getInstance();
27
28         try {
29             MainClass app = new MainClass();
30             config.init(argv[1]);
31             app.initLogger(config, argv[0]);
32
33             ConnectionFactory factory = new ConnectionFactory();
34             factory.setHost(config.get("MAIN", "server-address", "localhost"));
35             Connection connection = factory.newConnection();
36             Channel channel = connection.createChannel();
37
38             String stockQueue = config.get("QUEUES", "stock-manager-queue");
39             // To secure fairness between the processes
40             channel.basicQos(1);
41             channel.queueDeclare(stockQueue,
42                                 false,
43                                 false,
44                                 false,
45                                 null);
46
47             Consumer consumer = new StockManager(channel);
48             channel.basicConsume(stockQueue, true, consumer);
49         } catch (IllegalArgumentException e) {
50             // We couldn't open the logger. Just exit
51             System.out.println(e);
52             System.exit(-1);
53         } catch (TimeoutException e) {
54             logger.log(LogLevel.ERROR, e.toString());
55         } catch (IOException e) {
56             logger.log(LogLevel.ERROR, e.toString());
57         }
58     }
59
60
61     private void initLogger(ConfigParser config, String processNumber)
62     throws IllegalArgumentException {
63         String logFileName = config.get("MAIN", "log-file");
64         String logLevel = config.get("MAIN", "log-level");
65
66         Logger logger = Logger.getInstance();
67         logger.init(logFileName, LogLevel.parse(logLevel));
68         logger.setPrefix("[STOCK_MANAGER " + processNumber + " ]");
69         logger.log(LogLevel.DEBUG, "Process started");
70     }
71
72 }

```

Sep 20, 15 19:50

RequestDispatcher.java

Page 1/2

```

1 package requestDispatcher;
2
3 import com.rabbitmq.client.ConnectionFactory;
4 import com.rabbitmq.client.Connection;
5 import com.rabbitmq.client.Channel;
6 import com.rabbitmq.client.Consumer;
7 import com.rabbitmq.client.DefaultConsumer;
8 import com.rabbitmq.client.Envelope;
9 import com.rabbitmq.client.AMQP;
10 import org.apache.commons.lang3.SerializationUtils;
11
12 import configParser.ConfigParser;
13 import common.Order;
14 import common.OrderState;
15 import logger.Logger;
16 import logger.LogLevel;
17
18 import java.io.IOException;
19
20
21 public class RequestDispatcher extends DefaultConsumer {
22     public RequestDispatcher(Channel channel) throws IllegalArgumentException,
23                             IOException {
24         super(channel);
25         logger_ = Logger.getInstance();
26         config_ = ConfigParser.getInstance();
27         this.initQueues();
28     }
29
30     @Override
31     public void handleDelivery(String consumerTag,
32                               Envelope envelope,
33                               AMQP.BasicProperties properties,
34                               byte[] body) throws IOException {
35         Order newOrder = (Order) SerializationUtils.deserialize(body);
36
37         newOrder.state(OrderState.RECEIVED);
38         body = SerializationUtils.serialize(newOrder);
39
40         this.getChannel().basicPublish("", orderManagerQueueName_, null, body);
41         logger_.log(LogLevel.DEBUG, "Order received: " + newOrder.stringID());
42
43         // Send the order received to the auditLog
44         this.getChannel().basicPublish("", auditLogQueueName_, null, body);
45
46         // Send the order to the Stock Manager
47         this.getChannel().basicPublish("", stockManagerQueueName_, null, body);
48     }
49
50     /**
51      * @brief Declare the queues. This is necessary because maybe they have not
52      * been created yet
53      */
54     private void initQueues() throws IOException {
55         Channel channel = this.getChannel();
56         channel.basicQos(1);
57
58         auditLogQueueName_ = config_.get("QUEUES", "audit-log-queue");
59         channel.queueDeclare(auditLogQueueName_,
60                             false,
61                             false,
62                             false,
63                             null);
64
65         orderManagerQueueName_ = config_.get("QUEUES", "order-manager-queue");
66         channel.queueDeclare(orderManagerQueueName_,
67                             false,
68                             false,
69                             false,
70                             null);
71
72         stockManagerQueueName_ = config_.get("QUEUES", "stock-manager-queue");
73         channel.queueDeclare(stockManagerQueueName_,

```

Sep 20, 15 19:50

RequestDispatcher.java

Page 2/2

```

74         false,
75         false,
76         false,
77         null);
78     }
79
80     private Logger logger_;
81     private ConfigParser config_;
82     private String auditLogQueueName_;
83     private String stockManagerQueueName_;
84     private String orderManagerQueueName_;
85 }

```

Sep 20, 15 20:01

MainClass.java

Page 1/2

```

1  package requestDispatcher;
2
3  // Program includes
4  import configParser.ConfigParser;
5  import logger.Logger;
6  import logger.LogLevel;
7  import requestDispatcher.RequestDispatcher;
8
9  // External libraries includes
10 import com.rabbitmq.client.ConnectionFactory;
11 import com.rabbitmq.client.Connection;
12 import com.rabbitmq.client.Channel;
13 import com.rabbitmq.client.Consumer;
14 import com.rabbitmq.client.DefaultConsumer;
15 import com.rabbitmq.client.Envelope;
16 import com.rabbitmq.client.AMQP;
17
18 // Java includes
19 import java.lang.IllegalArgumentException;
20 import java.io.IOException;
21 import java.util.concurrent.TimeoutException;
22
23 public class MainClass {
24     public static void main(String[] argv) {
25         ConfigParser config = ConfigParser.getInstance();
26
27         Logger logger = Logger.getInstance();
28
29         try {
30             MainClass app = new MainClass();
31             config.init(argv[1]);
32             app.initLogger(config, argv[0]);
33
34             ConnectionFactory factory = new ConnectionFactory();
35             factory.setHost(config.get("MAIN", "server-address", "localhost"));
36             Connection connection = factory.newConnection();
37             Channel channel = connection.createChannel();
38
39             String clientQueue = config.get("QUEUES", "client-queue");
40             // To secure fairness between the processes
41             channel.basicQos(1);
42             channel.queueDeclare(clientQueue,
43                                 false,
44                                 false,
45                                 false,
46                                 null);
47
48             Consumer consumer = new RequestDispatcher(channel);
49             channel.basicConsume(clientQueue, true, consumer);
50
51             catch (IllegalArgumentException e) {
52                 // We couldn't open the logger. Just exit
53                 System.out.println(e);
54                 System.exit(-1);
55             }
56             catch (TimeoutException e) {
57                 logger.log(LogLevel.ERROR, e.toString());
58             }
59             catch (IOException e) {
60                 logger.log(LogLevel.ERROR, e.toString());
61             }
62         }
63
64         private void initLogger(ConfigParser config, String processNumber)
65         throws IllegalArgumentException {
66             String logFileName = config.get("MAIN", "log-file");
67             String logLevel = config.get("MAIN", "log-level");
68
69             Logger logger = Logger.getInstance();
70             logger.init(logFileName, LogLevel.parse(logLevel));
71             logger.setPrefix("[REQUEST_DISPATCHER " + processNumber + " ]");
72             logger.log(LogLevel.DEBUG, "Process started");
73         }

```

Sep 20, 15 20:01

MainClass.java

Page 2/2

```

73 }

```

Sep 22, 15 8:32

QuerySolver.java

Page 1/1

```

1 package querySolver;
2
3 import java.util.UUID;
4 import com.rabbitmq.client.ConnectionFactory;
5 import com.rabbitmq.client.Connection;
6 import com.rabbitmq.client.Channel;
7 import com.rabbitmq.client.Consumer;
8 import com.rabbitmq.client.DefaultConsumer;
9 import com.rabbitmq.client.Envelope;
10 import com.rabbitmq.client.AMQP;
11 import org.apache.commons.lang3.SerializationUtils;
12
13 import configParser.ConfigParser;
14 import common.Order;
15 import common.OrderState;
16 import common.OrderDB;
17 import logger.Logger;
18 import logger.LogLevel;
19
20 import java.io.IOException;
21
22
23 public class QuerySolver extends DefaultConsumer {
24     public QuerySolver(Channel channel) throws IllegalArgumentException,
25         IOException {
26         super(channel);
27         logger_ = Logger.getInstance();
28         config_ = ConfigParser.getInstance();
29         orderDB_ = new OrderDB(config_.get("ORDER", "order-db-directory"));
30     }
31
32     @Override
33     public void handleDelivery(String consumerTag,
34         Envelope envelope,
35         AMQP.BasicProperties properties,
36         byte[] body) throws IOException {
37         UUID orderKey = (UUID) SerializationUtils.deserialize(body);
38         logger_.log(LogLevel.DEBUG, "Query received. Key: "
39             + orderKey.toString());
40
41         Order order = orderDB_.get(orderKey);
42         if (order != null) {
43             logger_.log(LogLevel.NOTICE, "Order " + orderKey.toString()
44                 + "-State: " + order.state().toString());
45         }
46         else {
47             logger_.log(LogLevel.WARNING, "Order " + orderKey.toString()
48                 + " was not processed yet.");
49         }
50     }
51
52     private Logger logger_;
53     private ConfigParser config_;
54     private OrderDB orderDB_;
55 }

```

Sep 20, 15 22:40

MainClass.java

Page 1/2

```

1 package querySolver;
2
3 // Program includes
4 import configParser.ConfigParser;
5 import logger.Logger;
6 import logger.LogLevel;
7 import querySolver.QuerySolver;
8
9 // External libraries includes
10 import com.rabbitmq.client.ConnectionFactory;
11 import com.rabbitmq.client.Connection;
12 import com.rabbitmq.client.Channel;
13 import com.rabbitmq.client.Consumer;
14 import com.rabbitmq.client.DefaultConsumer;
15 import com.rabbitmq.client.Envelope;
16 import com.rabbitmq.client.AMQP;
17
18 // Java includes
19 import java.lang.IllegalArgumentException;
20 import java.io.IOException;
21 import java.util.concurrent.TimeoutException;
22
23 public class MainClass {
24     public static void main(String[] argv) {
25         ConfigParser config = ConfigParser.getInstance();
26
27         Logger logger = Logger.getInstance();
28
29         try {
30             MainClass app = new MainClass();
31             config.init(argv[1]);
32             app.initLogger(config, argv[0]);
33
34             ConnectionFactory factory = new ConnectionFactory();
35             factory.setHost(config.get("MAIN", "server-address", "localhost"));
36             Connection connection = factory.newConnection();
37             Channel channel = connection.createChannel();
38
39             String queryQueue = config.get("QUEUES", "query-queue");
40             // To secure fairness between the processes
41             channel.basicQos(1);
42             channel.queueDeclare(queryQueue,
43                 false,
44                 false,
45                 false,
46                 null);
47
48             Consumer consumer = new QuerySolver(channel);
49             channel.basicConsume(queryQueue, true, consumer);
50
51             catch (IllegalArgumentException e) {
52                 // We couldn't open the logger. Just exit
53                 System.out.println(e);
54                 System.exit(-1);
55             }
56
57             catch (TimeoutException e) {
58                 logger.log(LogLevel.ERROR, e.toString());
59             }
60
61             catch (IOException e) {
62                 logger.log(LogLevel.ERROR, e.toString());
63             }
64
65         }
66
67         private void initLogger(ConfigParser config, String processNumber)
68         throws IllegalArgumentException {
69             String logFileName = config.get("MAIN", "log-file");
70             String logLevel = config.get("MAIN", "log-level");
71
72             Logger logger = Logger.getInstance();
73             logger.init(logFileName, LogLevel.parse(logLevel));
74             logger.setPrefix("[QUERY_SOLVER " + processNumber + " ]");
75             logger.log(LogLevel.DEBUG, "Process started");
76         }

```

Sep 20, 15 22:40

MainClass.java

Page 2/2

73 }

Sep 22, 15 8:32

OrderManager.java

Page 1/2

```

1  package orderManager;
2
3  import java.lang.System;
4  import com.rabbitmq.client.ConnectionFactory;
5  import com.rabbitmq.client.Connection;
6  import com.rabbitmq.client.Channel;
7  import com.rabbitmq.client.Consumer;
8  import com.rabbitmq.client.DefaultConsumer;
9  import com.rabbitmq.client.Envelope;
10 import com.rabbitmq.client.AMQP;
11 import org.apache.commons.lang3.SerializationUtils;
12
13 import configParser.ConfigParser;
14 import common.Order;
15 import common.OrderDB;
16 import common.OrderState;
17 import logger.Logger;
18 import logger.LogLevel;
19
20 import java.io.IOException;
21
22
23 public class OrderManager extends DefaultConsumer {
24     public OrderManager(Channel channel) throws IllegalArgumentException,
25                                     IOException {
26         super(channel);
27         logger_ = Logger.getInstance();
28         config_ = ConfigParser.getInstance();
29
30         orderDB_ = new OrderDB(config_.get("ORDER", "order-db-directory"));
31         this.initQueues();
32     }
33
34     @Override
35     public void handleDelivery(String consumerTag,
36                               Envelope envelope,
37                               AMQP.BasicProperties properties,
38                               byte[] body) throws IOException {
39         Order newOrder = (Order) SerializationUtils.deserialize(body);
40         logger_.log(LogLevel.DEBUG, "Order received: " + newOrder.stringID());
41
42         OrderState state = newOrder.state();
43         elapsedTime_ = System.currentTimeMillis();
44
45         switch(newOrder.state()) {
46             case RECEIVED:
47                 // Add the order to the DB
48                 orderDB_.add(newOrder);
49                 elapsedTime_ = System.currentTimeMillis() - elapsedTime_;
50                 logger_.log(LogLevel.NOTICE, "OrderDB::add. Time: "
51                                     + elapsedTime_ + " ms.");
52                 break;
53             case DELIVERED:
54             case REJECTED:
55                 orderDB_.alter(newOrder);
56                 elapsedTime_ = System.currentTimeMillis() - elapsedTime_;
57                 logger_.log(LogLevel.NOTICE, "OrderDB::alter. Time: "
58                                     + elapsedTime_ + " ms.");
59                 break;
60             case ACCEPTED:
61                 orderDB_.alter(newOrder);
62                 elapsedTime_ = System.currentTimeMillis() - elapsedTime_;
63                 logger_.log(LogLevel.NOTICE, "OrderDB::alter. Time: "
64                                     + elapsedTime_ + " ms.");
65                 this.getChannel().basicPublish("",
66                                     deliveryQueueName_,
67                                     null,
68                                     body);
69                 break;
70         }
71
72         logger_.log(LogLevel.INFO, "Order processed: " + newOrder.stringID());
73

```


Sep 22, 15 8:32

OrderManager.java

Page 2/2

```

74 }
75
76 /**
77  * @brief Declare the queues. This is necessary because maybe they have not
78  * been created yet
79  */
80 private void initQueues() throws IOException {
81     Channel channel = this.getChannel();
82     channel.basicQos(1);
83
84     deliveryQueueName_ = config_.get("QUEUES", "delivery-queue");
85     channel.queueDeclare(deliveryQueueName_,
86                         false,
87                         false,
88                         false,
89                         null);
90 }
91
92 private Logger logger_;
93 private ConfigParser config_;
94 private OrderDB orderDB_;
95 private long elapsedTime_;
96 private String deliveryQueueName_;
97 }

```

Sep 20, 15 20:02

MainClass.java

Page 1/2

```

1  package orderManager;
2
3  // Program includes
4  import configParser.ConfigParser;
5  import logger.Logger;
6  import logger.LogLevel;
7  import orderManager.OrderManager;
8
9  // External libraries includes
10 import com.rabbitmq.client.ConnectionFactory;
11 import com.rabbitmq.client.Connection;
12 import com.rabbitmq.client.Channel;
13 import com.rabbitmq.client.Consumer;
14 import com.rabbitmq.client.DefaultConsumer;
15 import com.rabbitmq.client.Envelope;
16 import com.rabbitmq.client.AMQP;
17
18 // Java includes
19 import java.lang.IllegalArgumentException;
20 import java.io.IOException;
21 import java.util.concurrent.TimeoutException;
22
23 public class MainClass {
24     public static void main(String[] argv) {
25         ConfigParser config = ConfigParser.getInstance();
26
27         Logger logger = Logger.getInstance();
28
29         try {
30             MainClass app = new MainClass();
31             config.init(argv[1]);
32             app.initLogger(config, argv[0]);
33
34             ConnectionFactory factory = new ConnectionFactory();
35             factory.setHost(config.get("MAIN", "server-address", "localhost"));
36             Connection connection = factory.newConnection();
37             Channel channel = connection.createChannel();
38
39             String orderQueue = config.get("QUEUES", "order-manager-queue");
40             // To secure fairness between the processes
41             channel.basicQos(1);
42             channel.queueDeclare(orderQueue,
43                                 false,
44                                 false,
45                                 false,
46                                 null);
47
48             Consumer consumer = new OrderManager(channel);
49             channel.basicConsume(orderQueue, true, consumer);
50
51             catch (SecurityException e) {
52                 logger.log(LogLevel.ERROR, "Cannot create OrderDB Directory. "
53                     + "Change folder permissions or point out to another path.");
54             }
55             catch (IllegalArgumentException e) {
56                 // We couldn't open the logger. Just exit
57                 System.out.println(e);
58                 System.exit(-1);
59             }
60             catch (TimeoutException e) {
61                 logger.log(LogLevel.ERROR, e.toString());
62             }
63             catch (IOException e) {
64                 logger.log(LogLevel.ERROR, e.toString());
65             }
66         }
67
68         private void initLogger(ConfigParser config, String processNumber)
69         throws IllegalArgumentException {
70             String logFileName = config.get("MAIN", "log-file");
71             String logLevel = config.get("MAIN", "log-level");
72
73             Logger logger = Logger.getInstance();

```

Sep 20, 15 20:02

MainClass.java

Page 2/2

```

73     logger.init(logFileName, LogLevel.parse(logLevel));
74     logger.setPrefix("[ORDER_MANAGER " + processNumber + "]);
75     logger.log(LogLevel.DEBUG, "Process started");
76 }
77 }

```

Sep 22, 15 8:32

LogLevel.java

Page 1/2

```

1  package logger;
2
3  public enum LogLevel {
4      ERROR(1),
5      CRITIC(2),
6      WARNING(3),
7      NOTICE(4),
8      INFO(5),
9      DEBUG(6),
10     TRACE(7);
11
12     LogLevel(int level) {
13         if (this.isLevelValid(level)) {
14             this.level_ = level;
15         }
16         else {
17             throw new IllegalArgumentException("LogLevel is out of bounds.");
18         }
19     }
20
21     public static LogLevel parse(String level) {
22         String lowerLogLevel = level.toLowerCase().trim();
23
24         if (lowerLogLevel.equals("error")) {
25             return LogLevel.ERROR;
26         }
27         else if (lowerLogLevel.equals("critic")) {
28             return LogLevel.CRITIC;
29         }
30         else if (lowerLogLevel.equals("warning")) {
31             return LogLevel.WARNING;
32         }
33         else if (lowerLogLevel.equals("info")) {
34             return LogLevel.INFO;
35         }
36         else if (lowerLogLevel.equals("notice")) {
37             return LogLevel.NOTICE;
38         }
39         else if (lowerLogLevel.equals("debug")) {
40             return LogLevel.DEBUG;
41         }
42         else if (lowerLogLevel.equals("trace")) {
43             return LogLevel.TRACE;
44         }
45
46         throw new IllegalArgumentException("Invalid LogLevel introduced: "
47             + level);
48     }
49
50     public int level() {
51         return level_;
52     }
53
54     public Boolean isLevelValid(int level) {
55         // TODO: This enum are shit. I try, but this is the best a can do
56         return level ≥ 1 ∧ level ≤ 7;
57         /*return level ≥ LogLevel.ERROR &&
58            level ≤ LogLevel.TRACE;*/
59     }
60
61     public String prefix(LogLevel verbosity) {
62         int level = verbosity.level();
63
64         if (this.isLevelValid(level)) {
65             return PREFIX_ARRAY[level - 1];
66         }
67
68         throw new IllegalArgumentException("LogLevel is out of bounds.");
69     }
70
71     private final int level_;
72     private final String PREFIX_ARRAY[] = {"[ERROR]",
73         "[CRITIC]",

```

Sep 22, 15 8:32

LogLevel.java

Page 2/2

```

74         "[WARNING]",
75         "[NOTICE]",
76         "[INFO]",
77         "[DEBUG]",
78         "[TRACE]" };
79     }

```

Sep 21, 15 5:50

Logger.java

Page 1/2

```

1  package logger;
2
3  import java.io.*;
4  import java.nio.channels.FileLock;
5  import java.text.DateFormat;
6  import java.text.SimpleDateFormat;
7  import java.util.concurrent.locks.Lock;
8  import java.util.concurrent.locks.ReentrantLock;
9  import java.util.Date;
10
11 import logger.LogLevel;
12
13
14
15 public class Logger {
16     private Logger() {}
17
18     public static Logger getInstance() {
19         if (logger_ == null) {
20             logger_ = new Logger();
21         }
22
23         return logger_;
24     }
25
26     public void init(String filePath, LogLevel verbosity) {
27         verbosity_ = verbosity;
28         dateFormat_ = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
29         lock_ = new ReentrantLock();
30
31         try {
32             // Open file in append mode
33             fstream_ = new FileOutputStream(filePath, true);
34         }
35         catch (IOException e) {
36             System.err.println("[LOGGER] Error calling init() method.");
37             System.err.println(e);
38         }
39     }
40
41     public void setPrefix(String prefix) {
42         try {
43             lock_.lock();
44             prefix_ = prefix;
45         }
46         finally {
47             lock_.unlock();
48         }
49     }
50
51     public void terminate() throws IOException {
52         FileLock lock = null;
53         try {
54             lock = fstream_.getChannel().lock();
55             fstream_.close();
56         }
57         catch (IOException e) {
58             System.err.println("[LOGGER] Error calling terminate() method.");
59             System.err.println(e);
60             System.exit(-1);
61         }
62         finally {
63             lock.release();
64         }
65     }
66
67     public void log(LogLevel verbosity, String msg) {
68         if (verbosity_.level() >= verbosity.level()) {
69             this.write(verbosity_.prefix(verbosity) + " " + msg);
70         }
71     }
72
73     private void write(String msg) {

```

Sep 21, 15 5:50

Logger.java

Page 2/2

```

74     try {
75         Date date = new Date();
76         msg = dateFormat_.format(date) + " " + prefix_ + " " + msg + "\n";
77
78         FileLock lock = fstream_.getChannel().lock();
79         fstream_.write(msg.getBytes());
80         fstream_.flush();
81         // Remove sync to enhance performance
82         // fstream_.getFD().sync();
83         lock.release();
84     }
85     catch (IOException e) {
86         System.err.println( "[LOGGER] Error calling write() method." );
87         System.err.println(e);
88         System.exit(-1);
89     }
90 }
91
92 private static Logger logger_ = null;
93 private DateFormat dateFormat_;
94 private LogLevel verbosity_;
95 private FileOutputStream fstream_;
96 private Lock lock_;
97 private String prefix_;
98 }

```

Sep 22, 15 23:58

Launcher.py

Page 1/2

```

1  import subprocess
2  import sys
3  import os
4  import ConfigParser
5
6
7  class Launcher(object):
8      def __init__(self):
9          self._config = ConfigParser.RawConfigParser()
10         self._config.read('launcher.ini')
11         self._absolute_path = self._config.get("MAIN", "absolute-path")
12         self._processes_pid_list = []
13         self._manual_processes = {}
14         self._processes_config_file = self._config.get("MAIN",
15                                                         "processes-config-file")
16
17         self._common_classpath = ""
18         self._common_classpath = self.process_classpath("MAIN")
19
20     def process_classpath(self, section):
21         # Process the classpath
22         classpath = self._config.get(section, "classpath")
23
24         ret_classpath = ""
25         for lib in classpath.split(":"):
26             ret_classpath += self._absolute_path + lib + ":"
27         return self._common_classpath + ret_classpath
28
29     def init_system_processes(self):
30         for section in self._config.sections():
31             if section == "MAIN":
32                 continue
33
34             if not self._config.getboolean(section, "run"):
35                 continue
36
37             classpath = self.process_classpath(section)
38             classname = self._config.get(section, "class-name")
39             amount_processes = self._config.getint(section, "amount")
40
41             for i in range(1, amount_processes + 1):
42                 print "Proceed to execute instance with ID {0} of program {1}" \
43                     .format(str(i), classname.split(".")[0])
44
45                 call_args = []
46                 call_args.extend(["java",
47                                 "-cp",
48                                 classpath[:-1],
49                                 classname,
50                                 str(i),
51                                 self._processes_config_file])
52                 process = subprocess.Popen(call_args, shell=False)
53                 if self._config.getboolean(section, "kill"):
54                     self._processes_pid_list.append(process)
55
56     def wait_for_events(self):
57         # Wait for an input
58         prompt = "Write 'STOP' to terminate the "\
59                 "system. Write the section name "\
60                 "of a process to run a instance of it.\n"
61
62         while 1:
63             user_input = raw_input(prompt)
64             input_args = user_input.split(" ")
65
66             if input_args[0] == "STOP" or user_input == "STOP":
67                 if len(input_args) == 3:
68                     # Try to kill a specific process
69                     self.kill_process(input_args[1:])
70                 else:
71                     # Kill all scheduled processes
72                     self.kill_processes()
73                     break
74             elif user_input in self._config.sections():

```

Sep 22, 15 23:58

Launcher.py

Page 2/2

```

74         # Process created who will terminate by their own
75         self.run_process(user_input)
76         elif len(input_args) > 1 ^ input_args[0]\
77         in self._config.sections():
78             self.run_process(input_args[0], input_args[1])
79
80
81     def run_process(self, section, key=None):
82         classpath = self.process_classpath(section)
83         classname = self._config.get(section, "class-name")
84         print "Proceed to run program " + classname
85
86         call_args = []
87         call_args.extend(["java",
88                         "-cp",
89                         classpath[:-1],
90                         classname,
91                         "X" if key == None else key,
92                         self._processes_config_file])
93
94         process = subprocess.Popen(call_args, shell=False)
95         if key != None:
96             self._manual_processes[section, key] = process
97
98     def kill_processes(self):
99         for process in self._processes_pid_list:
100             print "Killing process with PID " + str(process.pid)
101             os.system("kill -15 " + str(process.pid))
102
103     def kill_process(self, key):
104         key = tuple(key)
105         if self._manual_processes[key] != None:
106             print "Killing process " + key[0] + " with PID " + \
107                 str(self._manual_processes[key].pid)
108             os.system("kill -15 " + str(self._manual_processes[key].pid))
109             del self._manual_processes[key]
110         else:
111             print "Process was not registered: " + key
112
113
114     def main():
115         launcher = Launcher()
116         launcher.init_system_processes()
117         launcher.wait_for_events()
118
119
120 if __name__ == '__main__':
121     main()

```

Sep 22, 15 23:58

MainClass.java

Page 1/2

```

1  package employer;
2
3  // Program includes
4  import java.lang.Thread;
5  import java.lang.Runtime;
6  import configParser.ConfigParser;
7  import logger.Logger;
8  import logger.LogLevel;
9  import requestDispatcher.RequestDispatcher;
10
11 // External libraries includes
12 import com.rabbitmq.client.ConnectionFactory;
13 import com.rabbitmq.client.Connection;
14 import com.rabbitmq.client.Channel;
15 import com.rabbitmq.client.Consumer;
16 import com.rabbitmq.client.DefaultConsumer;
17 import com.rabbitmq.client.Envelope;
18 import com.rabbitmq.client.AMQP;
19
20 // Java includes
21 import java.lang.IllegalArgumentException;
22 import java.io.IOException;
23 import java.util.concurrent.TimeoutException;
24
25 public class MainClass extends Thread {
26     public MainClass() {
27         logger_ = Logger.getInstance();
28     }
29
30     public static void main(String[] argv) {
31         ConfigParser config = ConfigParser.getInstance();
32
33         Logger logger = Logger.getInstance();
34
35         try {
36             MainClass app = new MainClass();
37             Runtime.getRuntime().addShutdownHook(app);
38
39             config.init(argv[1]);
40             app.initLogger(config, argv[0]);
41
42             ConnectionFactory factory = new ConnectionFactory();
43             factory.setHost(config.get("MAIN", "server-address", "localhost"));
44             Connection connection = factory.newConnection();
45             Channel channel = connection.createChannel();
46
47             String deliveryQueue = config.get("QUEUES", "delivery-queue");
48             // To secure fairness between the processes
49             channel.basicQos(1);
50             channel.queueDeclare(deliveryQueue,
51                                 false,
52                                 false,
53                                 false,
54                                 null);
55
56             Consumer consumer = new Employer(channel, deliveryQueue);
57             channel.basicConsume(deliveryQueue, true, consumer);
58
59             catch (IllegalArgumentException e) {
60                 // We couldn't open the logger. Just exit
61                 System.out.println(e);
62                 System.exit(-1);
63             }
64
65             catch (TimeoutException e) {
66                 logger.log(LogLevel.ERROR, e.toString());
67             }
68
69             catch (IOException e) {
70                 logger.log(LogLevel.ERROR, e.toString());
71             }
72
73         }
74
75         private void initLogger(ConfigParser config, String processNumber)
76         throws IllegalArgumentException {

```

Sep 22, 15 23:58

MainClass.java

Page 2/2

```

73     String logFileName = config.get("MAIN", "log-file");
74     String logLevel = config.get("MAIN", "log-level");
75
76     Logger logger = Logger.getInstance();
77     logger.init(logFileName, LogLevel.parse(logLevel));
78     logger.setPrefix("[EMPLOYER " + processNumber + "];");
79     logger.log(LogLevel.DEBUG, "Process started");
80 }
81
82 public void run() {
83     logger_.log(LogLevel.NOTICE, "Program finished by signal.");
84 }
85
86 private Logger logger_;
87 }

```

Sep 22, 15 23:58

Employer.java

Page 1/2

```

1  package employer;
2
3  import com.rabbitmq.client.ConnectionFactory;
4  import com.rabbitmq.client.Connection;
5  import com.rabbitmq.client.Channel;
6  import com.rabbitmq.client.Consumer;
7  import com.rabbitmq.client.DefaultConsumer;
8  import com.rabbitmq.client.Envelope;
9  import com.rabbitmq.client.AMQP;
10 import org.apache.commons.lang3.SerializationUtils;
11
12 import configParser.ConfigParser;
13 import common.Order;
14 import common.OrderState;
15 import logger.Logger;
16 import logger.LogLevel;
17
18 import java.io.IOException;
19 import java.util.concurrent.TimeoutException;
20
21 public class Employer extends DefaultConsumer {
22     public Employer(Channel channel,
23                     String channelName) throws IllegalArgumentException,
24                                     IOException {
25         super(channel);
26         logger_ = Logger.getInstance();
27         config_ = ConfigParser.getInstance();
28         this.initQueues();
29
30         channelName_ = channelName;
31         channelClosed_ = false;
32
33         amountOrdersToProcess_ =
34             Integer.parseInt(config_.get("EMPLOYER",
35                                         "amount-orders-to-process"));
36     }
37
38     @Override
39     public void handleDelivery(String consumerTag,
40                               Envelope envelope,
41                               AMQP.BasicProperties properties,
42                               byte[] body) throws IOException {
43         if (channelClosed_) {
44             return;
45         }
46
47         Order newOrder = (Order) SerializationUtils.deserialize(body);
48         // Change the state a we should have process the order and be ready
49         // to deliver it
50         newOrder.state(OrderState.DELIVERED);
51         body = SerializationUtils.serialize(newOrder);
52
53         logger_.log(LogLevel.DEBUG, "Order delivered: " + newOrder.stringID());
54         this.getChannel().basicPublish("", orderManagerQueueName_, null, body);
55     }
56
57     /**
58      * @brief Declare the queues. This is necessary because maybe they have not
59      * been created yet
60      */
61     private void initQueues() throws IOException {
62         Channel channel = this.getChannel();
63         channel.basicQos(1);
64
65         orderManagerQueueName_ = config_.get("QUEUES", "order-manager-queue");
66         channel.queueDeclare(orderManagerQueueName_,
67                             false,
68                             false,
69                             false,
70                             null);
71     }
72
73     private Logger logger_;

```

Sep 22, 15 23:58

Employer.java

Page 2/2

```

74 private ConfigParser config_;
75 private String orderManagerQueueName_;
76 private String channelName_;
77 private int amountOrdersToProcess_;
78 private boolean channelClosed_;
79 }

```

Sep 19, 15 21:26

ConfigParser.java

Page 1/1

```

1 package configParser;
2
3 import java.lang.IllegalArgumentException;
4 import java.io.FileReader;
5 import java.io.IOException;
6
7 // External imports
8 import org.ini4j.Ini;
9
10 import logger.Logger;
11 import logger.LogLevel;
12
13
14 public class ConfigParser {
15     private ConfigParser() {}
16
17     public void init(String filePath) {
18         // TODO: Receive the config file from an argument
19         String configFileName = filePath;
20         config_ = new Ini();
21
22         try {
23             config_.load(new FileReader(configFileName));
24         }
25         catch (IOException e) {
26             System.err.println("[CONFIGPARSER] Could not open config file. ");
27             System.err.println(e);
28             System.exit(-1);
29         }
30     }
31
32     public static ConfigParser getInstance() {
33         if (configParser_ == null) {
34             configParser_ = new ConfigParser();
35         }
36
37         return configParser_;
38     }
39
40     public String get(String section, String key, String defaultValue) {
41         String value = config_.get(section, key);
42         if (value != null) {
43             return value;
44         }
45
46         Logger.getInstance().log(LogLevel.INFO,
47             "[CONFIGPARSER] Key (" + section + ", " + key
48             + ") was not found. Using default value: " + defaultValue);
49         return defaultValue;
50     }
51
52     public String get(String section, String key) {
53         String value = config_.get(section, key);
54         if (value != null) {
55             return value;
56         }
57
58         String msg = "Value doesn't exists in Config File. Section: "
59             + section + " - Key: " + key;
60         throw new IllegalArgumentException(msg);
61     }
62
63     private static ConfigParser configParser_ = null;
64     private Ini config_;
65 }

```

Sep 20, 15 1:19

Product.java

Page 1/1

```
1 package common;
2
3 import java.util.Arrays;
4 import java.util.List;
5 import java.util.Collections;
6 import java.util.Random;
7
8 public enum Product {
9     TYPE_1,
10    TYPE_2,
11    TYPE_3,
12    TYPE_4,
13    TYPE_5;
14
15    private static final List<Product> VALUES =
16        Collections.unmodifiableList(Arrays.asList(values()));
17    private static final int SIZE = VALUES.size();
18    private static final Random RANDOM = new Random();
19
20    public static Product randomProduct() {
21        return VALUES.get(RANDOM.nextInt(SIZE));
22    }
23 }
24
25
26
27
28
```

Sep 20, 15 12:22

OrderState.java

Page 1/1

```
1 package common;
2
3 public enum OrderState {
4     TO_BE_PROCESSED,
5     RECEIVED,
6     REJECTED,
7     ACCEPTED,
8     DELIVERED;
9 }
```


Sep 20, 15 19:12

Order.java

Page 1/2

```

1 package common;
2
3 import common.OrderState;
4 import common.Product;
5
6 import java.io.Serializable;
7 import java.util.UUID;
8
9
10 public class Order implements Serializable {
11     public Order(Product productType, Long amount) {
12         state_ = OrderState.TO_BE_PROCESSED;
13         productType_ = productType;
14         amount_ = amount;
15         uuid_ = UUID.randomUUID();
16     }
17
18     public Order(UUID uuid,
19                 Product productType,
20                 Long amount,
21                 OrderState state) {
22         uuid_ = uuid;
23         productType_ = productType;
24         amount_ = amount;
25         state_ = state;
26     }
27
28     public UUID id() {
29         return uuid_;
30     }
31
32     public OrderState state() {
33         return state_;
34     }
35
36     public void state(OrderState state) {
37         state_ = state;
38     }
39
40     public String stringID() {
41         return uuid_.toString();
42     }
43
44     public Product productType() {
45         return productType_;
46     }
47
48     public Long amount() {
49         return amount_;
50     }
51
52     public String toString() {
53         String aux = "";
54         // Reduce the size of the UUID to better log size comprehension
55         aux += "Order ID: " + uuid_.toString().substring(0,6) + "-";
56         aux += "State: " + state_.toString() + "-";
57         aux += "Product Type: " + productType_.toString() + "-";
58         aux += "Amount: " + amount_;
59         return aux;
60     }
61
62     /**
63      * @brief Return a representation of the order just with a truncated
64      * UUID and the state of the order
65      */
66     public String toStringShort() {
67         String aux = "";
68         // Reduce the size of the UUID to better log size comprehension
69         aux += "Order ID: " + uuid_.toString().substring(0,6) + "-";
70         aux += "State: " + state_.toString() + "-";
71         aux += "Product Type: " + productType_.toString() + "-";
72         aux += "Amount: " + amount_;
73         return aux;

```

Sep 20, 15 19:12

Order.java

Page 2/2

```

74     }
75
76     public String toStringFull() {
77         String aux = "";
78         // Reduce the size of the UUID to better log size comprehension
79         aux += "Order ID: " + uuid_.toString() + "-";
80         aux += "State: " + state_.toString() + "-";
81         aux += "Product Type: " + productType_.toString() + "-";
82         aux += "Amount: " + amount_;
83         return aux;
84     }
85
86     private OrderState state_;
87     private final UUID uuid_;
88     // FIXME: Create a enum or something like that to represent this
89     private Product productType_;
90     private Long amount_;
91     public static final long serialVersionUID = 123L;
92 }

```

Sep 22, 15 8:32

OrderDB.java

Page 1/3

```

1 package common;
2
3 import java.lang.System;
4 import java.io.EOFException;
5 import java.util.UUID;
6 import java.nio.ByteBuffer;
7 import java.nio.channels.FileLock;
8 import java.io.RandomAccessFile;
9 import common.Order;
10 import common.OrderDBEntry;
11 import common.Product;
12 import logger.Logger;
13 import logger.LogLevel;
14
15 import java.lang.IllegalArgumentException;
16 import java.io.IOException;
17 import java.io.RandomAccessFile;
18 import java.io.File;
19 import java.lang.SecurityException;
20
21 public class OrderDB {
22     public OrderDB(String dirPath) throws SecurityException,
23         IOException,
24         IllegalArgumentException {
25         logger_ = Logger.getInstance();
26
27         dirPath_ = dirPath;
28         File file = new File(dirPath);
29         if (!file.isDirectory()) {
30             // Use mkdirs instead of mkdir, to create intermediate
31             // directories if they does not exists
32             file.mkdirs();
33         }
34     }
35
36     public void add(Order order) throws IOException {
37         RandomAccessFile file = this.getOrderFile(order, "rwd");
38         FileLock lock = file.getChannel().lock();
39         OrderDBEntry entry = new OrderDBEntry(order);
40
41         // Go to the end of the file
42         file.seek(file.length());
43         file.write(entry.getBytes());
44         lock.release();
45         file.close();
46     }
47
48     public void alter(Order order) throws IOException {
49         RandomAccessFile file = this.getOrderFile(order, "rwd");
50         FileLock lock = file.getChannel().lock();
51         long offset = this.getOffsetToEntry(file, order.id());
52
53         // Sanity check
54         if (offset == -1) {
55             // This should not happen. Stop program
56             logger_.log(LogLevel.ERROR, "Order doesn't exists in alter");
57             System.exit(-1);
58         }
59
60         // Do a have to do this or the file is in the correct offset?
61         file.seek(offset);
62         OrderDBEntry entry = new OrderDBEntry(order);
63         file.write(entry.getBytes());
64
65         lock.release();
66         file.close();
67     }
68
69     public Order get(UUID orderKey) throws IOException {
70         RandomAccessFile file = this.getOrderFile(orderKey, "rwd");
71         FileLock lock = file.getChannel().lock();
72         long offset = this.getOffsetToEntry(file, orderKey);
73     }

```

Sep 22, 15 8:32

OrderDB.java

Page 2/3

```

74     if (offset == -1) {
75         lock.release();
76         file.close();
77         return null;
78     }
79
80     byte[] entryBuffer = new byte[OrderDBEntry.ENTRY_SIZE];
81     file.seek(offset);
82     file.read(entryBuffer, 0, OrderDBEntry.ENTRY_SIZE);
83     OrderDBEntry entry = new OrderDBEntry(entryBuffer);
84
85     lock.release();
86     file.close();
87
88     return entry.order();
89 }
90
91 private long getOffsetToEntry(RandomAccessFile file, UUID orderKey)
92 throws IOException {
93     byte[] buffer = new byte[OrderDBEntry.UUID_SIZE];
94     try {
95         file.seek(0);
96         while(true) {
97             int readBytes = file.read(buffer, 0, OrderDBEntry.UUID_SIZE);
98             if (readBytes == -1) {
99                 // EOF reached
100                 return -1;
101             }
102
103             // Create a UUID
104             ByteBuffer bb = ByteBuffer.wrap(buffer);
105             UUID uuid = new UUID(bb.getLong(), bb.getLong());
106
107             if (uuid.equals(orderKey)) {
108                 break;
109             }
110
111             // Jump to the next entry
112             file.skipBytes(OrderDBEntry.ENTRY_SIZE -
113                 OrderDBEntry.UUID_SIZE);
114         }
115     } catch (EOFException e) {
116         // If this happen, then the product does not exists and we have
117         // a bug in the system. ABORT!
118         logger_.log(LogLevel.ERROR, "Order does not exists in OrderDB. "
119             + "Order key: " + orderKey.toString());
120         System.exit(-1);
121     }
122
123     // We must sustract the key that was read in the last comparison
124     return file.getFilePointer() - OrderDBEntry.UUID_SIZE;
125 }
126
127 /**
128  * @brief Get the file where the order must be stored in the DB
129  */
130 private RandomAccessFile getOrderFile(Order order, String mode)
131 throws IOException {
132     String subUuid = order.stringID().substring(0, 2);
133     return this.getOrderFile(subUuid, mode);
134 }
135
136 private RandomAccessFile getOrderFile(UUID uuid, String mode)
137 throws IOException {
138     String subUuid = uuid.toString().substring(0, 2);
139     return this.getOrderFile(subUuid, mode);
140 }
141
142 private RandomAccessFile getOrderFile(String subUuid, String mode)
143 throws IOException {
144     String fileName = dirPath_ + "/" + subUuid;
145 }
146

```

Sep 22, 15 8:32

OrderDB.java

Page 3/3

```

147
148      // Again, we cannot check if the file exists. Just try to create it
149      File orderFile = new File(fileName);
150      orderFile.createNewFile();
151      return new RandomAccessFile(fileName, mode);
152  }
153
154  private String dirPath_;
155  private Logger logger_;
156  }

```

Sep 21, 15 0:16

OrderDBEntry.java

Page 1/2

```

1  package common;
2
3  import java.nio.ByteBuffer;
4  import java.util.UUID;
5  import common.Order;
6  import common.OrderState;
7  import common.Product;
8
9  /**
10   * DB ENTRY serialization structure:
11   * UUID - 16 bytes
12   * PRODUCT_TYPE - 10 bytes (pad with spaces)
13   * AMOUNT - 8 bytes (long type)
14   * ORDER_STATE - 15 bytes (pad with spaces)
15   */
16
17  public class OrderDBEntry {
18      public OrderDBEntry(Order order) {
19          order_ = order;
20      }
21
22      /**
23       * @brief Receives a DB entry in bytes, and deserialize it to get an Order
24       */
25      public OrderDBEntry(byte[] entry) {
26          ByteBuffer bb = ByteBuffer.wrap(entry);
27
28          // UUID;
29          UUID uuid = new UUID(bb.getLong(), bb.getLong());
30
31          // Product
32          byte[] productBuffer = new byte[PRODUCT_SIZE];
33          bb.get(productBuffer, 0, PRODUCT_SIZE);
34          Product product = Product.valueOf(new String(productBuffer).trim());
35
36          // Amount
37          long amount = bb.getLong();
38
39          // Order State
40          byte[] stateBuffer = new byte[STATE_SIZE];
41          bb.get(stateBuffer, 0, STATE_SIZE);
42          OrderState state = OrderState.valueOf(new String(stateBuffer).trim());
43
44          order_ = new Order(uuid, product, amount, state);
45      }
46
47      /**
48       * @brief Serializes the Order stored as it should be stored in the DB
49       * @return The order serialized as it would be stored in the DB
50       */
51      public byte[] getBytes() {
52          ByteBuffer bb = ByteBuffer.allocate(ENTRY_SIZE);
53
54          // UUID
55          bb.putLong(order_.id().getMostSignificantBits());
56          bb.putLong(order_.id().getLeastSignificantBits());
57
58          // Product
59          String product = String.format("%-10s",
60                                          order_.productType().toString());
61          bb.put(product.getBytes());
62
63          // Amount
64          bb.putLong(order_.amount());
65
66          // Order State
67          String state = String.format("%-15s", order_.state().toString());
68          bb.put(state.getBytes());
69
70          return bb.array();
71      }
72
73      public Order order() {

```

Sep 21, 15 0:16

OrderDBEntry.java

Page 2/2

```

74     return order_;
75 }
76
77 private Order order_;
78 // 16 = UUID size (This shouldn't change)
79 public static final int UUID_SIZE = 16;
80 public static final int PRODUCT_SIZE = 10;
81 // 8 = Long size (I don't expect this to change)
82 private static final int AMOUNT_SIZE = 8;
83 private static final int STATE_SIZE = 15;
84 public static final int ENTRY_SIZE = PRODUCT_SIZE +
85                                     STATE_SIZE +
86                                     UUID_SIZE +
87                                     AMOUNT_SIZE;
88 }

```

Sep 22, 15 23:58

MainClass.java

Page 1/3

```

1  package client;
2
3  import java.util.concurrent.locks.ReentrantLock;
4  import java.util.concurrent.locks.Lock;
5  import java.lang.Thread;
6  import java.lang.Runtime;
7  import java.util.Iterator;
8  import java.util.UUID;
9  import java.util.ArrayList;
10 import java.lang.Math;
11 import java.lang.System;
12 import java.util.Random;
13 import java.util.concurrent.TimeoutException;
14 import java.io.IOException;
15 import java.util.Random;
16
17 import com.rabbitmq.client.ConnectionFactory;
18 import com.rabbitmq.client.Connection;
19 import com.rabbitmq.client.Channel;
20 import org.apache.commons.lang3.SerializationUtils;
21
22 import common.Order;
23 import common.OrderState;
24 import common.Product;
25 import configParser.ConfigParser;
26 import logger.Logger;
27 import logger.LogLevel;
28
29 public class MainClass extends Thread {
30     public MainClass(String[] argv) {
31         randomGenerator_ = new Random(System.currentTimeMillis());
32         config_ = ConfigParser.getInstance();
33         logger_ = Logger.getInstance();
34         lock_ = new ReentrantLock();
35         ordersKeys_ = new ArrayList<UUID>();
36
37         config_.init(argv[1]);
38         this.initLogger(argv[0]);
39     }
40
41     public static void main(String[] argv) throws InterruptedException {
42         ConfigParser config = ConfigParser.getInstance();
43         Logger logger = Logger.getInstance();
44         try {
45             MainClass app = new MainClass(argv);
46             Runtime.getRuntime().addShutdownHook(app);
47
48             app.initRabbit();
49             logger.log(LogLevel.INFO,
50                 "Proceed to create and send orders");
51             app.sendOrders();
52
53             int sleepTime = Integer.parseInt(config.get("CLIENT",
54                 "sleep-between-orders-and-queries", "0"));
55
56             if (sleepTime > 0) {
57                 logger.log(LogLevel.INFO,
58                     "Proceed to sleep before send queries to the system");
59                 Thread.sleep(sleepTime * 1000);
60             }
61
62             logger.log(LogLevel.INFO,
63                 "Proceed to send queries associated with the orders created");
64             app.queryOrders();
65             app.terminate();
66         }
67         catch (IllegalArgumentException e) {
68             // We couldn't open the logger. Just exit
69             System.out.println(e);
70             System.exit(-1);
71         }
72         catch (TimeoutException e) {
73             logger.log(LogLevel.ERROR, e.toString());

```

Sep 22, 15 23:58

MainClass.java

Page 2/3

```

74     }
75     catch (IOException e) {
76         logger.log(LogLevel.ERROR, e.toString());
77     }
78 }
79
80 private void initLogger(String processNumber)
81 throws IllegalArgumentException {
82     String logFileName = config_.get("MAIN", "log-file");
83     String logLevel = config_.get("MAIN", "log-level");
84
85     Logger logger = Logger.getInstance();
86     logger.init(logFileName, LogLevel.parse(logLevel));
87     logger.setPrefix("[CLIENT " + processNumber + "]");
88     logger.log(LogLevel.DEBUG, "Process started");
89 }
90
91 public void initRabbit() throws IOException,
92     TimeoutException,
93     IllegalArgumentException {
94     ConnectionFactory factory = new ConnectionFactory();
95     factory.setHost(config_.get("MAIN", "server-address", "localhost"));
96     connection_ = factory.newConnection();
97     channel_ = connection_.createChannel();
98
99     clientQueue_ = config_.get("QUEUES", "client-queue");
100    channel_.queueDeclare(clientQueue_,
101        false,
102        false,
103        false,
104        null);
105
106    queryQueue_ = config_.get("QUEUES", "query-queue");
107    channel_.queueDeclare(queryQueue_,
108        false,
109        false,
110        false,
111        null);
112 }
113
114 public void terminate() throws IOException, TimeoutException {
115     channel_.close();
116     connection_.close();
117 }
118
119 public void sendOrders() throws IOException {
120     int ordersToCreate =
121         Integer.parseInt(config_.get("CLIENT",
122             "amount-orders-to-simulate",
123             "1"));
124     logger_.log(LogLevel.DEBUG, "Orders to simulate: "
125         + ordersToCreate);
126
127     for (int i = 0; i < ordersToCreate; ++i) {
128         Order order = this.generateRandomOrder();
129         byte[] data = SerializationUtils.serialize(order);
130
131         // Store the UUID generated to then make a query to the system
132         ordersKeys_.add(order.id());
133
134         logger_.log(LogLevel.DEBUG, "Sending order: " + order.stringID());
135         channel_.basicPublish("", clientQueue_, null, data);
136     }
137 }
138
139 /**
140  * @brief Sends as much queries as the parameter amount-queries-to-simulate
141  * @details If amount-queries-to-simulate is bigger than
142  * amount-orders-to-simulate, a round-robin algorithm is used to keep
143  * querying orders
144  */
145 public void queryOrders() throws IOException {
146     int amountQueries =

```

Sep 22, 15 23:58

MainClass.java

Page 3/3

```

147     Integer.parseInt(config_.get("CLIENT",
148         "amount-queries-to-simulate",
149         "1"));
150
151     Iterator<UUID> it = ordersKeys_.iterator();
152     while (amountQueries > 0) {
153         --amountQueries;
154
155         UUID key = it.next();
156         byte[] data = SerializationUtils.serialize(key);
157         logger_.log(LogLevel.DEBUG, "Querying order: " + key.toString());
158         channel_.basicPublish("", queryQueue_, null, data);
159
160         if (!it.hasNext()) {
161             it = ordersKeys_.iterator();
162         }
163     }
164 }
165
166 private Order generateRandomOrder() {
167     long amount = Math.abs(randomGenerator_.nextInt() % 10) + 1;
168     return new Order(Product.randomProduct(), amount);
169 }
170
171 public void run() {
172     try {
173         this.terminate();
174     }
175     catch (TimeoutException e) {
176     }
177     catch (IOException e) {
178     }
179 }
180
181 private Logger logger_;
182 private ConfigParser config_;
183 private Random randomGenerator_;
184 private Channel channel_;
185 private Connection connection_;
186 private String clientQueue_;
187 private String queryQueue_;
188 private Lock lock_;
189 private ArrayList<UUID> ordersKeys_;
190 }

```

Sep 20, 15 20:02

MainClass.java

Page 1/2

```

1 package auditLogger;
2
3 import java.lang.Runtime;
4 import java.lang.Thread;
5 import java.util.concurrent.TimeoutException;
6 import java.io.IOException;
7
8 // External libraries includes
9 import com.rabbitmq.client.ConnectionFactory;
10 import com.rabbitmq.client.Connection;
11 import com.rabbitmq.client.Channel;
12 import com.rabbitmq.client.Consumer;
13 import com.rabbitmq.client.DefaultConsumer;
14 import com.rabbitmq.client.Envelope;
15 import com.rabbitmq.client.AMQP;
16 import org.apache.commons.lang3.SerializationUtils;
17
18 import auditLogger.AuditLogger;
19 import common.Order;
20 import configParser.ConfigParser;
21 import logger.Logger;
22 import logger.LogLevel;
23
24
25 public class MainClass {
26     public static void main(String[] argv) {
27         ConfigParser config = ConfigParser.getInstance();
28
29         Logger logger = Logger.getInstance();
30
31         try {
32             MainClass app = new MainClass();
33             config.init(argv[1]);
34             app.initLogger(config, argv[0]);
35
36             Channel channel = app.initRabbit(config);
37             String auditLogFile = config.get("AUDIT", "audit-log-file");
38             String auditLogQueue = config.get("QUEUES", "audit-log-queue");
39             Consumer consumer = new AuditLogger(channel, auditLogFile);
40             channel.basicConsume(auditLogQueue, true, consumer);
41         }
42         catch (IllegalArgumentException e) {
43             // We couldn't open the logger. Just exit
44             System.out.println(e);
45             System.exit(-1);
46         }
47         catch (TimeoutException e) {
48             logger.log(LogLevel.ERROR, e.toString());
49         }
50         catch (IOException e) {
51             logger.log(LogLevel.ERROR, e.toString());
52         }
53     }
54
55     private void initLogger(ConfigParser config, String processNumber)
56     throws IllegalArgumentException {
57         String logFileName = config.get("MAIN", "log-file");
58         String logLevel = config.get("MAIN", "log-level");
59
60         Logger logger = Logger.getInstance();
61         logger.init(logFileName, LogLevel.parse(logLevel));
62         logger.setPrefix("[AUDIT_LOG " + processNumber + "]");
63         logger.log(LogLevel.DEBUG, "Process started");
64     }
65
66     private Channel initRabbit(ConfigParser config)
67     throws IllegalArgumentException,
68         IOException,
69         TimeoutException {
70         ConnectionFactory factory = new ConnectionFactory();
71         factory.setHost(config.get("MAIN", "server-address", "localhost"));
72         Connection connection = factory.newConnection();
73         Channel channel = connection.createChannel();

```

Sep 20, 15 20:02

MainClass.java

Page 2/2

```

73
74         String auditLogQueue = config.get("QUEUES", "audit-log-queue");
75         // To secure fairness between the processes
76         channel.basicQos(1);
77         channel.queueDeclare(auditLogQueue,
78                             false,
79                             false,
80                             false,
81                             null);
82
83         return channel;
84     }
85 }

```

Sep 20, 15 14:07

AuditLogger.java

Page 1/1

```

1 package auditLogger;
2
3 import java.text.SimpleDateFormat;
4 import java.util.Date;
5 import java.text.DateFormat;
6 import java.io.File;
7 import com.rabbitmq.client.ConnectionFactory;
8 import com.rabbitmq.client.Connection;
9 import com.rabbitmq.client.Channel;
10 import com.rabbitmq.client.Consumer;
11 import com.rabbitmq.client.DefaultConsumer;
12 import com.rabbitmq.client.Envelope;
13 import com.rabbitmq.client.AMQP;
14 import org.apache.commons.lang3.SerializationUtils;
15
16 import common.Order;
17 import logger.Logger;
18 import logger.LogLevel;
19
20 import java.io.IOException;
21 import java.io.FileWriter;
22
23
24 public class AuditLogger extends DefaultConsumer {
25     public AuditLogger(Channel channel,
26         String auditLogFile) throws IOException {
27         super(channel);
28         logger_ = Logger.getInstance();
29         // Open log file in append mode
30         File file = new File(auditLogFile);
31         dateFormat_ = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
32
33         if (!file.exists()) {
34             logger_.log(LogLevel.WARNING,
35                 "Audit log doesn't exists."
36                 + " Proceed to create it. AuditLogFile: " + auditLogFile);
37             file.createNewFile();
38         }
39         writer_ = new FileWriter(auditLogFile, true);
40     }
41
42     @Override
43     public void handleDelivery(String consumerTag,
44         Envelope envelope,
45         AMQP.BasicProperties properties,
46         byte[] body) throws IOException {
47         Order newOrder = (Order) SerializationUtils.deserialize(body);
48         logger_.log(LogLevel.DEBUG, "Order received: " + newOrder.stringID());
49         writer_.write(this.generateAuditEntry(newOrder) + "\n");
50         writer_.flush();
51     }
52
53     private String generateAuditEntry(Order order) {
54         Date date = new Date();
55         String entry = dateFormat_.format(date) + " - ";
56         entry += "OrderID: " + order.stringID();
57         return entry;
58     }
59
60     private Logger logger_;
61     private FileWriter writer_;
62     private DateFormat dateFormat_;
63 }

```

Sep 22, 15 23:58

launcher.ini

Page 1/1

```

1 [MAIN]
2 absolute-path = /media/Datos/Facultad/75.61 Taller III/Order_Dispatcher/
3 processes-config-file = /media/Datos/Facultad/75.61 Taller III/Order_Dispatcher/
  configuration.ini
4 classpath = lib/ini4j-0.5.4/ini4j-0.5.4.jar:lib/rabbitmq-java-client-bin-3.5.4/r
  abbitmq-client.jar:lib/ini4j-0.5.4/ini4j-0.5.4.jar:lib/commons-lang3-3.4/commons
  -lang3-3.4.jar
5
6 # All the paths in the classpaths classes must be relative paths to the
7 # absolute-path given in the main section
8 # Also, the paths who have spaces must not be escaped. The app will do it
9 [REQUEST-DISPATCHER]
10 classpath = build/jar/RequestDispatcher.jar
11 class-name = requestDispatcher.MainClass
12 run = true
13 kill = true
14 amount = 1
15
16 [CLIENT]
17 classpath = build/jar/Client.jar
18 class-name = client.MainClass
19 run = false
20 kill = true
21 amount = 1
22
23 [EMPLOYER]
24 classpath = build/jar/Employer.jar
25 class-name = employer.MainClass
26 run = false
27 kill = true
28 amount = 1
29
30 [AUDIT-LOGGER]
31 classpath = build/jar/AuditLogger.jar
32 class-name = auditLogger.MainClass
33 run = true
34 kill = true
35 # This cannot be a value higher than one
36 amount = 1
37
38 [STOCK-MANAGER]
39 classpath = build/jar/StockManager.jar
40 class-name = stockManager.MainClass
41 run = true
42 kill = true
43 amount = 1
44
45 [ORDER-MANAGER]
46 classpath = build/jar/OrderManager.jar
47 class-name = orderManager.MainClass
48 run = true
49 kill = true
50 amount = 1
51
52 [QUERY-SOLVER]
53 classpath = build/jar/QuerySolver.jar
54 class-name = querySolver.MainClass
55 run = true
56 kill = true
57 amount = 1

```

Sep 22, 15 23:58

configuration.ini

Page 1/1

```

1 [MAIN]
2 server-address = localhost
3 log-file = /tmp/OrderDispatcher.log
4 log-level = DEBUG
5
6 [QUEUES]
7 client-queue = CLIENT-QUEUE
8 audit-log-queue = LOG-QUEUE
9 stock-manager-queue = STOCK-QUEUE
10 order-manager-queue = ORDER-QUEUE
11 query-queue = QUERY-QUEUE
12 delivery-queue = DELIVERY-QUEUE
13
14 [AUDIT]
15 audit-log-file = /tmp/Audit.log
16
17 [STOCK]
18 stock-db-file = /tmp/Stock.db
19
20 [CLIENT]
21 amount-orders-to-simulate = 1000
22 amount-queries-to-simulate = 1000
23 # In seconds
24 sleep-between-orders-and-queries = 20
25
26 [ORDER]
27 order-db-directory = /tmp/OrderDB
28
29 [STOCK-PROVIDER]
30 global-increase = 10
31 type-1-increase = 1
32 type-2-increase = 2
33 type-3-increase = 3
34 type-4-increase = 4
35 type-5-increase = 5
36
37 [EMPLOYER]
38 amount-orders-to-process = 100

```

Sep 24, 15 1:24

Table of Content

Page 1/1

1	Table of Contents				
2	1	MainClass.java.....	sheets	1 to 1 (1)	pages 1- 2 91 lines
3	2	StockManager.java...	sheets	2 to 2 (1)	pages 3- 4 87 lines
4	3	StockDB.java.....	sheets	3 to 4 (2)	pages 5- 7 207 lines
5	4	MainClass.java.....	sheets	4 to 4 (1)	pages 8- 8 73 lines
6	5	RequestDispatcher.java	sheets	5 to 5 (1)	pages 9- 10 86 lines
7	6	MainClass.java.....	sheets	6 to 6 (1)	pages 11- 12 74 lines
8	7	QuerySolver.java....	sheets	7 to 7 (1)	pages 13- 13 56 lines
9	8	MainClass.java.....	sheets	7 to 8 (2)	pages 14- 15 74 lines
10	9	OrderManager.java...	sheets	8 to 9 (2)	pages 16- 17 98 lines
11	10	MainClass.java.....	sheets	9 to 10 (2)	pages 18- 19 78 lines
12	11	LogLevel.java.....	sheets	10 to 11 (2)	pages 20- 21 80 lines
13	12	Logger.java.....	sheets	11 to 12 (2)	pages 22- 23 99 lines
14	13	Launcher.py.....	sheets	12 to 13 (2)	pages 24- 25 122 lines
15	14	MainClass.java.....	sheets	13 to 14 (2)	pages 26- 27 88 lines
16	15	Employer.java.....	sheets	14 to 15 (2)	pages 28- 29 80 lines
17	16	ConfigParser.java...	sheets	15 to 15 (1)	pages 30- 30 66 lines
18	17	Product.java.....	sheets	16 to 16 (1)	pages 31- 31 29 lines
19	18	OrderState.java.....	sheets	16 to 16 (1)	pages 32- 32 10 lines
20	19	Order.java.....	sheets	17 to 17 (1)	pages 33- 34 93 lines
21	20	OrderDB.java.....	sheets	18 to 19 (2)	pages 35- 37 157 lines
22	21	OrderDBEntry.java...	sheets	19 to 20 (2)	pages 38- 39 89 lines
23	22	MainClass.java.....	sheets	20 to 21 (2)	pages 40- 42 191 lines
24	23	MainClass.java.....	sheets	22 to 22 (1)	pages 43- 44 86 lines
25	24	AuditLogger.java....	sheets	23 to 23 (1)	pages 45- 45 64 lines
26	25	launcher.ini.....	sheets	23 to 23 (1)	pages 46- 46 58 lines
27	26	configuration.ini...	sheets	24 to 24 (1)	pages 47- 47 39 lines