JCSync developer's guide

Autor:
Piotr Bucior

Version: v0.1

Contents

1 Getting started

1.1 Configuring and running Bootstrap server from source code

To configure Bootrstrap server, which will be used in our examples we need to create class similar to this:

```
1  package pl.edu.pjwstk.mteam.jcsync.samples.utils;
2
3  import pl.edu.pjwstk.p2pp.P2PPManager;
4  import pl.edu.pjwstk.p2pp.objects.P2POptions;
5  import pl.edu.pjwstk.p2pp.superpeer.SuperPeerBootstrapServer;
6  import pl.edu.pjwstk.p2pp.superpeer.SuperPeerConstants;
7  import pl.edu.pjwstk.p2pp.util.P2PPMessageFactory;
8  import pl.edu.pjwstk.p2pp.util.P2PPUtils;
9
10 public class BootstrapServerRunner extends Thread{
11
12     private P2PPManager manager;
13     private final int port;
14
15     public BootstrapServerRunner(int port){
16         this.port = port;
17     }
18     @Override
19     public void run(){
20         try{
21                 this.manager = new P2PPManager(0, this.port, 0, 0, 0, "", "", new↩
                        P2PPMessageFactory(), "myOverlayID".getBytes("UTF-8"));
22                 String hashAlgorithm = "SHA-1";
23                 byte hashLength = 20;
24                 byte hashBase = 2;
25                 String overlayID = "myOverlayID";
26                 this.manager.setOptions(new P2POptions(P2PPUtils.↩
                       convertHashAlgorithmName(hashAlgorithm), hashLength,
27                           P2PPUtils.convertP2PAlgorithmName(SuperPeerConstants.↩
                                 SUPERPEER_PROTOCOL_NAME), hashBase, overlayID.↩
                                 getBytes("UTF-8")));
28
29                 SuperPeerBootstrapServer server = new SuperPeerBootstrapServer();
30                 this.manager.addEntity(server);
31                 this.manager.start();
32                 }catch(Exception e){
33                     e.printStackTrace();
34                 }
35     }
36 }
```

Listing 1: BootstrapServerRunner.java

To run server we need to create it with given port number and just call *start()* method:

```
1         BootstrapServerRunner bs;
2         int bsPort=6060;
3         //creates simple bootstrap server
4         this.bs = new BootstrapServerRunner(bsPort);
5         // run bs
6         this.bs.start();
```

Listing 2: Creating and running an instance of BootstrapServerRunner

Now the bootstrap server is ready to work.

1.2 Configuring and running Bootstrap server from command line

In the current version of p2pm library there is also possible to run Bootstrap Server from command line by using *pl.edu.pjwstk.p2pp.launchers.CommandLineLauncher* tool. We will us it by using **run_p2pp.sh** bash script. Lets check available arguments:

```
./run\_p2pp.sh
09:14:56.350 INFO  [pl.edu.pjwstk.p2pp.launchers.CommandLineLauncher] - parsing ↩
    starts
09:14:56.388 ERROR [pl.edu.pjwstk.p2pp.launchers.CommandLineLauncher] - problem ↩
    with number
09:14:56.397 INFO  [pl.edu.pjwstk.p2pp.launchers.CommandLineLauncher] - Bad ↩
    arguments.
Bad arguments. Read help.
usage: Arguments depend on mode.
Example for bootstrap server:
-m 4 -udp port\_number -p overlay\_protocol_name -h hash\_algorithm\_name -o
overlay\_id -hl hash\_byte\_length -hb hash\_base -sra
server\_reflexive\_address -srp server\_reflexive\_port
Example for peer:
-p overlay\_protocol_name -m 1 -udp port\_number -id peer\_id -sra
server\_reflexive\_ip\_address -srp server\_reflexive\_port
 -debug          Enables debug mode
 -dtls <arg>     DTLS port to be used. If present, DTLS protocol will be
                 used. (NOT SUPPORTED)
 -h <arg>        Hash algorithm. Used by bootstrap server.
 -hb <arg>       Hash base. Used by bootstrap server.
 -hl <arg>       Length of hash. Used by bootstrap server.
 -id <arg>       UnhashedID used by peer or client.
 -keys <arg>     Path to a file with ssl keys
 -m <arg>        Mode. Peer(1), client(2), bootstrap server(4), diagnostics
                 server(8), enrollment and authentication server(16) and ↩
                     combinations of
                 them (5 is bootstrap server and peer).
 -o <arg>        ID of overlay. Used by bootstrap server.
 -p <arg>        Name of P2P protocol.
 -pass <arg>     Passphrase to unlock the keys
 -sf <arg>       File with social network (only with SocialCircle protocol)
 -sra <arg>      TEMPORARILY USED (ICE implementation is not ready). Server
                 reflexive address (in xxx.xxx.xxx.xxx form)
 -srp <arg>      TEMPORARILY USED (ICE implementation is not ready). Server
                 reflexive port
 -ssl <arg>      SSL port to be used. If present, SSL protocol will be used.
 -tcp <arg>      TCP port to be used. If present, TCP protocol will be used.
 -tls <arg>      TLS port to be used. If present, TLS protocol will be used.
 -udp <arg>      UDP port to be used. If present, UDP protocol will be used.
                 (only one supported at the moment)
```

Listing 3: Possible arguments for CommandLineLauncher

Now to run bootstrap server with the same arguments as in the previous section simply call:

```
./run\_p2pp.sh -l bootlog.txt -m 4 -tcp 6060 -p SuperPeer -h SHA-1 -o ↩
    myOverlayID -hl 20 -hb 2 -sra 127.0.0.1 -srp 7080 -d
```

Listing 4: Running bootstrap server from command line

1.3 Getting p2p nodes ready to work

Listing below shows how to configure and connect to the bootsptrap server, which was already configured from source code in previous section.

```
1   import pl.edu.pjwstk.mteam.p2p.P2PNode;
2   // ...
3       P2PNode node1;
4       node1 = new P2PNode(null, P2PNode.RoutingAlgorithm.SUPERPEER);
5       //server reflexive address
6       node1.setServerReflexiveAddress("127.0.0.1");
7       //server reflexive port
8       node1.setServerReflexivePort(5050);
9       //boot IP address
10      node1.setBootIP("127.0.0.1");
11      //bootstrap server port
12      node1.setBootPort(6060);
13      //user name
14      node1.setUserName("node1");
```

```
15          //setting up UPD port number
16          node1.setUdpPort(4040);
17          //connect to the layer
18          node1.networkJoin();
19
20          //wait for connection
21          while(!node1.isConnected()){
22              try {
23                  Thread.sleep(100);
24              } catch (InterruptedException ex) {
25                  //...
26              }
27          }
```

Listing 5: Initialising p2p node

1.4 Getting JCSync instance ready to work

After the node was properly initialised and connected we can initialise JCSync layer instance.

```
1       import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
2       //...
3       private JCSyncCore core1;
4       //creates new jcsync core instance
5           core1 = new JCSyncCore(node1, 4444);
6
7           try {
8               core1.init();
9           } catch (Exception ex) {
10              // error handling
11          }
```

Listing 6: Initialising p2p node

Constructor of *core1* is called with two arguments, first of them is a node which will be related with this JCSync layer, the second is a port for JCSync layer.

1.5 Creating simple collection

Listing below shows how to create new HashMap.

```
1       import java.util.HashMap;
2       import java.util.logging.Level;
3       import java.util.logging.Logger;
4       import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
5       import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.↩
            JCSyncHashMap;
6       import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.↩
            SharedCollectionObject;
7       import pl.edu.pjwstk.mteam.jcsync.exception.ObjectExistsException;
8       import pl.edu.pjwstk.mteam.jcsync.exception.ObjectNotExistsException;
9       import pl.edu.pjwstk.mteam.jcsync.exception.OperationForbiddenException;
10      import pl.edu.pjwstk.mteam.jcsync.samples.utils.BootstrapServerRunner;
11      import pl.edu.pjwstk.mteam.p2p.P2PNode;
12      \\...
13
14      private JCSyncHashMap createHashMap(String testMap, JCSyncCore coreAlg) ↩
            throws ObjectExistsException, Exception {
15          // create collection instance
16          JCSyncHashMap map = new JCSyncHashMap();
17          // we need create an SharedCollectionObject, that will be assigned with ↩
                our collection.
18          SharedCollectionObject so_1 = new SharedCollectionObject(testMap, map, ↩
                coreAlg);
19          return map;
20      }
```

Listing 7: Creating new collection

By calling *JCSyncHashMap map = new JCSyncHashMap()* we create new instance of collection, but now it isn't initialised yet and can't be used.
Collection is initialised by creating new *SharedCollectionObject* object instance.
Lets look at the given arguments in the *SharedCollectionObject* constructor:
- *testMap* - is a collection identifier in the layer
- *map* - instance of collection, which will be related and managed by this this *SharedCollectionObject* instance
- *coreAlg* - an *JCSyncCore* instance of the JCSync layer

To create this collection we use code below.

```
1         //creates (blank) new hashmap with identifier "testMap"
2         String collID = "testMap";
3         try {
4             JCSyncHashMap hs_core1 = createHashMap(collID,this.core1);
5         } catch (ObjectExistsException ex) {
6             System.out.println("Ooops, collection with given name already exists.↩
                  ");
7         } catch (Exception ex) {
8             System.out.println("Ooops, an error occurred.");
9         }
10        // wait for data propagation
11        snooze(500);
```

Listing 8: Creating new collection

If we catch *ObjectExistsException* that means the collection with *testMap* identifier is already created in the network layer. Subscribe operation is described in the next section.

To create an instance of *ArrayList* implementation code should be as below.
In this sample *JCSyncArrayList* is created with some initial values provided by initValues argument:

```
1         private JCSyncArrayList<String> createArrayList(String testMap, ↩
              JCSyncCore coreAlg, ArrayList<String> initValues) throws ↩
              ObjectExistsException, Exception {
2         // create collection instance
3         JCSyncArrayList<String> arr = new JCSyncArrayList<String>(initValues);
4         // we need create an SharedCollectionObject, that will be assigned with ↩
              our collection.
5         SharedCollectionObject so_1 = new SharedCollectionObject(testMap, arr, ↩
              coreAlg);
6         return (JCSyncArrayList<String>) so_1.getNucleusObject();
7     }
```

Listing 9: Creating new ArrayList implementation

Now to get new instance of JCSyncArrayList simply use the same code as for creating JCSyncHashMap.

From next, to create an instance of *TreeMap* implementation we can simply use the same code:

```
1         private JCSyncTreeMap createTreeMap(String testMap, JCSyncCore coreAlg) ↩
              throws ObjectExistsException, Exception {
2         // create collection instance
3         JCSyncTreeMap map = new JCSyncTreeMap();
4         // we need create an SharedCollectionObject, that will be assigned with ↩
              our collection.
5         SharedCollectionObject so_1 = new SharedCollectionObject(testMap, map, ↩
              coreAlg);
6         return (JCSyncTreeMap) so_1.getNucleusObject();
7     }
```

Listing 10: Creating new TreeMap implementation

1.6 Subscribing to collection

To subscribe with already defined collection we must invoke method called:

SharedCollectionObject.getFromOverlay(collectionName, coreAlg);

which will create an instance of already defined *SharedCollectionObject*.

```
1          JCSyncHashMap hs_core2;
2          //try to create the same collection on node 2
3          try {
4              hs_core2 = createHashMap(collID,this.core2);
5              // it will throw ObjectExistsException (code below)
6          } catch (ObjectExistsException ex) {
7              try {
8                  // in this case, try to subscribe with this collection
9                  hs_core2 = (JCSyncHashMap) subscribeCollection(collID, core2).↩
                       getNucleusObject();
10             } catch (ObjectNotExistsException ex1) {
11                 System.out.println("Ooops, collection not exists.");
12             } catch (OperationForbiddenException ex1) {
13                 System.out.println("Ooops, you cannot subscribe to this ↩
                       collection.");
14             } catch (Exception ex1) {
15                 System.out.println("Ooops, an error occurred.");
16             }
17         } catch (Exception ex) {
18             System.out.println("Ooops, an error occurred.");
19         }
20         // wait for data propagation
21         snooze(500);
22
23         //...
24         private SharedCollectionObject subscribeCollection(String collectionName,↩
               JCSyncCore coreAlg)
25             throws ObjectNotExistsException, OperationForbiddenException, ↩
                   Exception {
26
27         SharedCollectionObject so =
28                 (SharedCollectionObject) SharedCollectionObject.getFromOverlay(↩
                       collectionName, coreAlg);
29         return so;
30     }
```

Listing 11: Subscribing to collection

1.7 Full source code

```
1  package pl.edu.pjwstk.mteam.jcsync.samples;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.logging.Level;
6  import java.util.logging.Logger;
7  import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
8  import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.JCSyncArrayList↩
       ;
9  import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.JCSyncHashMap;
10 import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.JCSyncTreeMap;
11 import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.↩
       SharedCollectionObject;
12 import pl.edu.pjwstk.mteam.jcsync.exception.ObjectExistsException;
13 import pl.edu.pjwstk.mteam.jcsync.exception.ObjectNotExistsException;
14 import pl.edu.pjwstk.mteam.jcsync.exception.OperationForbiddenException;
15 import pl.edu.pjwstk.mteam.jcsync.samples.utils.BootstrapServerRunner;
16 import pl.edu.pjwstk.mteam.p2p.P2PNode;
17
18 /**
19  * Sample class, that shows how to use implemented collections.
20  * @author Piotr Bucior
```

```
21  */
22 public class BasicCollectionUsage {
23
24     private BootstrapServerRunner bs;
25     private JCSyncCore core1;
26     private JCSyncCore core2;
27
28
29     public BasicCollectionUsage(){
30         // first of all - run bootstrap and creates two jcsync instances
31         initBootstrapServer(7080);
32         // first node that will be used in this sample
33         initNode1(6060,7080);
34         // 2nd node
35         initNode2(6070,7080);
36
37         //creates (blank) new hashmap with identifier "testMap"
38         String collID = "testMap";
39         try {
40             JCSyncHashMap hs_core1 = createHashMap(collID,this.core1);
41         } catch (ObjectExistsException ex) {
42             System.out.println("Ooops, collection with given name already exists.↩
                   ");
43         } catch (Exception ex) {
44             System.out.println("Ooops, an error occurred.");
45         }
46         // wait for data propagation
47         snooze(500);
48
49         JCSyncHashMap hs_core2;
50         //try to create the same collection on node 2
51         try {
52             hs_core2 = createHashMap(collID,this.core2);
53             // it will throw ObjectExistsException (code below)
54         } catch (ObjectExistsException ex) {
55             try {
56                 // in this case, try to subscribe with this collection
57                 hs_core2 = (JCSyncHashMap) subscribeCollection(collID, core2).↩
                       getNucleusObject();
58             } catch (ObjectNotExistsException ex1) {
59                 System.out.println("Ooops, collection not exists.");
60             } catch (OperationForbiddenException ex1) {
61                 System.out.println("Ooops, you cannot subscribe to this ↩
                       collection.");
62             } catch (Exception ex1) {
63                 System.out.println("Ooops, an error occurred.");
64             }
65         } catch (Exception ex) {
66             System.out.println("Ooops, an error occurred.");
67         }
68         // wait for data propagation
69         snooze(500);
70
71         // now we do own stuff on the collection, e.g. adding, removing, editing ↩
               elements
72
73
74     }
75
76     /**
77      * Method body shows how to create new blank collection instance
78      * @param testMap collection identifier in overlay.
79      * @return created collection
80      */
81     private JCSyncHashMap createHashMap(String testMap, JCSyncCore coreAlg) ↩
           throws ObjectExistsException, Exception {
82         // create collection instance
83         JCSyncHashMap map = new JCSyncHashMap();
84         // we need create an SharedCollectionObject, that will be assigned with ↩
               our collection.
85         SharedCollectionObject so_1 = new SharedCollectionObject(testMap, map, ↩
               coreAlg);
86         return map;
87     }
88
```

```
89      private JCSyncArrayList<String> createArrayList(String testMap, JCSyncCore ↩
            coreAlg, ArrayList<String> initValues) throws ObjectExistsException, ↩
            Exception {
90          // create collection instance
91          JCSyncArrayList<String> arr = new JCSyncArrayList<String>(initValues);
92          // we need create an SharedCollectionObject, that will be assigned with ↩
                our collection.
93          SharedCollectionObject so_1 = new SharedCollectionObject(testMap, arr, ↩
                coreAlg);
94          return (JCSyncArrayList<String>) so_1.getNucleusObject();
95      }
96
97      private JCSyncTreeMap createTreeMap(String testMap, JCSyncCore coreAlg) ↩
            throws ObjectExistsException, Exception {
98          // create collection instance
99          JCSyncTreeMap map = new JCSyncTreeMap();
100         // we need create an SharedCollectionObject, that will be assigned with ↩
                our collection.
101         SharedCollectionObject so_1 = new SharedCollectionObject(testMap, map, ↩
                coreAlg);
102         return (JCSyncTreeMap) so_1.getNucleusObject();
103     }
104
105     /**
106      * Method body shows how to create new collection instance with already ↩
             stored some data
107      * @param testMap collection identifier in overlay.
108      * @return created collection
109      */
110     private JCSyncHashMap createHashMap(String testMap, JCSyncCore coreAlg, ↩
            HashMap coreMap) throws ObjectExistsException, Exception {
111         // create collection instance
112         JCSyncHashMap map = new JCSyncHashMap(coreMap);
113         // we need create an SharedCollectionObject, that will be assigned with ↩
                our collection.
114         SharedCollectionObject so_1 = new SharedCollectionObject(testMap, map, ↩
                coreAlg);
115         return map;
116     }
117
118
119     private SharedCollectionObject subscribeCollection(String collectionName, ↩
            JCSyncCore coreAlg)
120             throws ObjectNotExistsException, OperationForbiddenException, ↩
                    Exception {
121
122         SharedCollectionObject so =
123                 (SharedCollectionObject) SharedCollectionObject.getFromOverlay(↩
                        collectionName, coreAlg);
124         return so;
125     }
126     /**
127      * initialise bootstrap server
128      * @param i port
129      */
130     private void initBootstrapServer(int i) {
131         //creates simple bootstrap server
132         this.bs = new BootstrapServerRunner(i);
133         // run bs
134         this.bs.start();
135     }
136     /**
137      * init first node and jcsync instance
138      * @param i port
139      * @param bootPort bootstrap server port
140      */
141     private void initNode1(int i, int bootPort) {
142         P2PNode node1;
143         node1 = new P2PNode(null, P2PNode.RoutingAlgorithm.SUPERPEER);
144         node1.setServerReflexiveAddress("127.0.0.1");
145         node1.setServerReflexivePort(bootPort);
146         node1.setBootIP("127.0.0.1");
147         node1.setBootPort(bootPort);
148         node1.setUserName("node1");
149         node1.setUdpPort(i);
150         node1.networkJoin();
```

```
151
152         //wait for connection
153         while(!node1.isConnected()){
154             try {
155                 Thread.sleep(100);
156             } catch (InterruptedException ex) {
157                 Logger.getLogger(BasicCollectionUsage.class.getName()).log(Level.↩
                        SEVERE, null, ex);
158             }
159         }
160         //creates new jcsync core instance
161         core1 = new JCSyncCore(node1, i+2);
162
163         try {
164             core1.init();
165         } catch (Exception ex) {
166             Logger.getLogger(BasicCollectionUsage.class.getName()).log(Level.↩
                    SEVERE, null, ex);
167         }
168     }
169     /**
170      * init first node and jcsync instance
171      * @param i port
172      */
173     private void initNode2(int i, int bootPort) {
174         P2PNode node2;
175         node2 = new P2PNode(null, P2PNode.RoutingAlgorithm.SUPERPEER);
176         node2.setServerReflexiveAddress("127.0.0.1");
177         node2.setServerReflexivePort(bootPort);
178         node2.setBootIP("127.0.0.1");
179         node2.setBootPort(bootPort);
180         node2.setUserName("node1");
181         node2.setUdpPort(i);
182         node2.networkJoin();
183
184         //wait for connection
185         while(!node2.isConnected()){
186             try {
187                 Thread.sleep(100);
188             } catch (InterruptedException ex) {
189                 Logger.getLogger(BasicCollectionUsage.class.getName()).log(Level.↩
                        SEVERE, null, ex);
190             }
191         }
192         //creates new jcsync core instance
193         core2 = new JCSyncCore(node2, i+2);
194
195         try {
196             core2.init();
197         } catch (Exception ex) {
198             Logger.getLogger(BasicCollectionUsage.class.getName()).log(Level.↩
                    SEVERE, null, ex);
199         }
200     }
201     private void snooze(long time){
202         try {
203             Thread.sleep(time);
204         } catch (InterruptedException ex) {
205             Logger.getLogger(BasicCollectionUsage.class.getName()).log(Level.↩
                    SEVERE, null, ex);
206         }
207     }
208
209
210 }
```

Listing 12: Full source code

1.8 Access restriction

This section will describe how to manage object access restrictions. As an example the collection will be used.

1.8.1 How to set the private access for collection

By default created collection is initialised with public access for all nodes. To change it to private access we must modify access control rules provides by *Publish-Subscribe* layer. To make it we must add users to the access control rules:

```
1         AccessControlLists acRules = new AccessControlLists(t);
2         acRules.getRule(PubSubConstants.OPERATION_SUBSCRIBE).addUser(↩
              PubSubConstants.EVENT_ALL,subscriber);
```

Above code means that only user defined as a *subscriber* object will be allowed to subscribe with specific object. For next, we can give this *acRules* as an arguments when we create new object. Created object will be allowed only for its owner.

```
1     import pl.edu.pjwstk.mteam.jcsync.core.AccessControlLists;
2     import pl.edu.pjwstk.mteam.jcsync.core.JCSyncAbstractSharedObject;
3     import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
4     import pl.edu.pjwstk.mteam.jcsync.exception.OperationForbiddenException;
5     import pl.edu.pjwstk.mteam.p2p.P2PNode;
6     import pl.edu.pjwstk.mteam.pubsub.core.PubSubConstants;
7     import pl.edu.pjwstk.mteam.pubsub.core.Subscriber;
8     import pl.edu.pjwstk.mteam.pubsub.core.Topic;
9     import pl.edu.pjwstk.p2pp.P2PPManager;
10    import pl.edu.pjwstk.p2pp.objects.P2POptions;
11    import pl.edu.pjwstk.p2pp.superpeer.SuperPeerBootstrapServer;
12    import pl.edu.pjwstk.p2pp.superpeer.SuperPeerConstants;
13    import pl.edu.pjwstk.p2pp.util.P2PPMessageFactory;
14    import pl.edu.pjwstk.p2pp.util.P2PPUtils;
15    // creating nodes, running bootstrap
16
17        String name = "SCO_private";
18        SharedCollectionObject s1 = null;
19        JCSyncHashMap hs = new JCSyncHashMap();
20        // section needed to create a Subscriber object
21        Topic t = new Topic(name);
22        Subscriber subscriber = new Subscriber("node1", t);
23        t.setOwner(subscriber);
24        // modify acRules
25        AccessControlLists acRules = new AccessControlLists(t);
26        acRules.getRule(PubSubConstants.OPERATION_SUBSCRIBE).addUser(↩
              PubSubConstants.EVENT_ALL,subscriber);
27        //give modified acRules as an arguments
28        s1 = new SharedCollectionObject(name, hs, core, acRules);
29        // wait for data propagation
30        Thread.sleep(500);
```

Listing 13: Modifying AccessControlLists to make private access to the object

To go back to the public access for this object see code below:

```
1     // remove earlier added user from acRules
2     acRules.getRule(PubSubConstants.OPERATION_SUBSCRIBE).removeUser(↩
          PubSubConstants.EVENT_ALL,subscriber);
3     // informs JCsync layer about it
4     core.modifyAccessControlLists(name, acRules);
5     // wait for data propagation
6     Thread.sleep(500);
```

Listing 14: Making public access for object

1.8.2 How to add new user to object with private access

To add new user ("node2") to our private object defined in previous section:

```
Subscriber subscriber2 = new Subscriber("node2", t);
        acRules.getRule(PubSubConstants.OPERATION_SUBSCRIBE).addUser(↩
            PubSubConstants.EVENT_ALL, subscriber2);
        core.modifyAccessControlLists(name, acRules);
```

Listing 15: Adding new user to private object

1.8.3 How to disable write access for specific user

To disable write access for user "node2":

```
import pl.edu.pjwstk.mteam.jcsync.core.AccessControlLists;
import pl.edu.pjwstk.mteam.jcsync.core.JCSyncAbstractSharedObject;
import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
import pl.edu.pjwstk.mteam.jcsync.exception.OperationForbiddenException;
import pl.edu.pjwstk.mteam.p2p.P2PNode;
import pl.edu.pjwstk.mteam.pubsub.core.Event;
import pl.edu.pjwstk.mteam.pubsub.core.PubSubConstants;
import pl.edu.pjwstk.mteam.pubsub.core.Subscriber;
import pl.edu.pjwstk.mteam.pubsub.core.Topic;
import pl.edu.pjwstk.p2pp.P2PPManager;
import pl.edu.pjwstk.p2pp.objects.P2POptions;
import pl.edu.pjwstk.p2pp.superpeer.SuperPeerBootstrapServer;
import pl.edu.pjwstk.p2pp.superpeer.SuperPeerConstants;
import pl.edu.pjwstk.p2pp.util.P2PPMessageFactory;
import pl.edu.pjwstk.p2pp.util.P2PPUtils;
import static pl.edu.pjwstk.mteam.jcsync.operation.RegisteredOperations.*;
// ...
    acRules.getRule(PubSubConstants.OPERATION_PUBLISH).addUser(↩
        OP_REQ_WRITE_METHOD, subscriber);
    core.modifyAccessControlLists(name, acRules);
```

Listing 16: Modifying write access to the collection

In this case only user *subscriber* is allowed to publishing changes on the collection. To add other users simply add them to the acRules.

1.8.4 Predefined events in access control lists

Table below shows defined events, which are described and manages by AccessControlLists.

Name	Description
OP_REQ_TRANSFER_OBJECT	Used to transport shared object over the layer to requesting node
OP_IND_TRANSFER_OBJECT	Transfer object indication. It is sends only to the request publisher
OP_REQ_LOCK_APPLY	Informs that the publisher wants to get exclusive access to the shared object
OP_IND_LOCK_APPLY	Lock apply indication. Sends only to the request publisher
OP_REQ_LOCK_RELEASE	Informs that the publisher is releasing exclusive access to the shared object
OP_IND_LOCK_RELEASE	Lock release indication. Sends only to the request publisher
OP_IND_WRITE_METHOD	'Write' type method indication
OP_REQ_WRITE_METHOD	Request to call 'write' type method
OP_REQ_READ_METHOD	Request to call a 'read' type method
OP_IND_READ_METHOD	'Read' type method indication

Table 1: Predefined JCSync events

1.9 The Observable-Observers mechanism

1.9.1 Introduce

This section describes Observable[1] mechanism implemented in Java and its extension by JCSync.

Lets see the documentation of Observable from JDK[2]:

"This class represents an observable object, or "data" in the model-view paradigm. It can be subclassed to represent an object that the application wants to have observed.

An observable object can have one or more observers. An observer may be any object that implements interface Observer. After an observable instance changes, an application calling the Observable's notifyObservers method causes all of its observers to be notified of the change by a call to their update method.

The order in which notifications will be delivered is unspecified. The default implementation provided in the Observerable class will notify Observers in the order in which they registered interest, but subclasses may change this order, use no guaranteed order, deliver notifications on separate threads, or may guarantee that their subclass follows this order, as they choose. [...]"

It says that by calling **notifyObservers(Object arg)** on *Observable* object we can pass any object (**arg**) to all of connected observers with this *Observable* object.
JCSync extension allows the same feature with possibility to informing the observer over the network layer.

[1] http://en.wikipedia.org/wiki/Observer_pattern
[2] JDK Observable specification

1.9.2 Example of usage

To get some impression how to use the *Observable* mechanism lets see example below.

This is a small and very simple chat application which is using *Observable* and *Observer* to notify messages between users. Let's see most important thinks in the ChatWindow.java

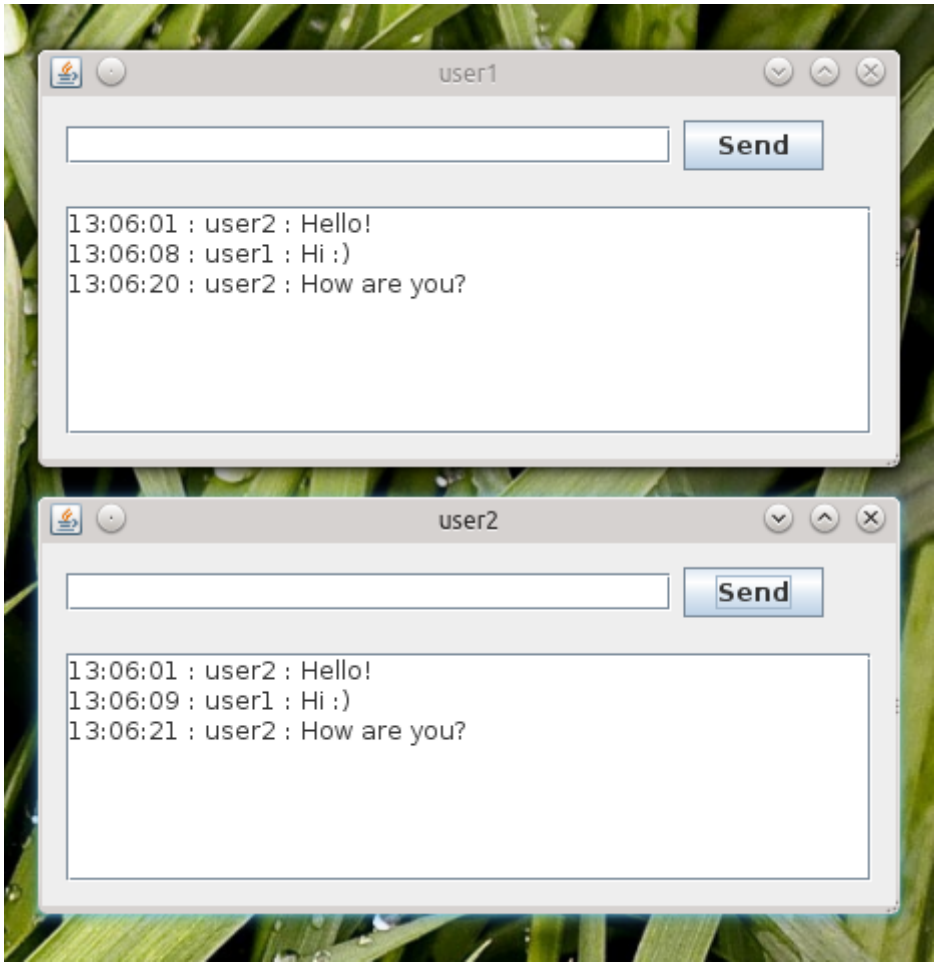

Figure 1: Chat window

source code. Its implementing Observer interface (method **update(...)**). By pressing jB_send button on chat window are invoked method on given *Observable* object to inform about new chat message. Now it could work only in one JVM.

```
package pl.edu.pjwstk.mteam.jcsync.samples.simpleChat;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Observable;
import java.util.Observer;

/**
 *
 * @author Piotr Bucior
 */
public class ChatWindow extends javax.swing.JFrame implements Observer {
    private Observable observable;
    private String publisher;
    // button to send message
    private javax.swing.JButton jB_send;
    private javax.swing.JScrollPane jScrollPane1;
    // chat history
    private javax.swing.JTextArea jTA_messages;
```

```
    // message to send
    private javax.swing.JTextField jt_message;
    /** Creates new form ChatWindow */
    public ChatWindow(Observable obs,String publisher) {
        initComponents();
        this.observable = obs;
        this.publisher = publisher;
        this.observable.addObserver(this);
        setTitle(publisher);
    }
    \\...
    @Override
    public void update(Observable o, Object arg) {
        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
        Date date = new Date();
        String message = (String)arg;
        StringBuilder sb = new StringBuilder();
        sb.append(dateFormat.format(date));
        sb.append(" : ");
        sb.append(message);
        sb.append("\n");
        this.jTA_messages.setText(this.jTA_messages.getText()+sb.toString());
    }
    private void jB_sendActionPerformed(java.awt.event.ActionEvent evt) {
        String message = this.publisher +" : "+this.jt_message.getText();
        this.jt_message.setText("");
        this.observable.notifyObservers(message);
    }
```

Listing 17: ChatWindow class

Lets try now to use *JCSyncObservable* to use this chat over the network layer. We will use some of code block described in previous sections.
The most important thinks is how to get a JCSync Observable extension, the code below shows how to do this. The method body is try to create new JCSyncObservable object with given id (*"observable"*) by using given JCSyncCore instance given as arguments. If object with the same identifier is already created then given JCSyncCore instance should try to subscribe with this object.

```
public JCSyncObservable getObservable(JCSyncCore coreAlg){
        String obs_id = "observable";
        SharedObservableObject soo = null;
        try {
            soo= new SharedObservableObject(obs_id, new JCSyncObservable(), ↩
                coreAlg);
        } catch (ObjectExistsException ex) {
            try {
                soo = (SharedObservableObject) SharedObservableObject.↩
                    getFromOverlay(obs_id, coreAlg);
            } catch (ObjectNotExistsException ex1) {
                Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, ↩
                    null, ex1);
            } catch (OperationForbiddenException ex1) {
                Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, ↩
                    null, ex1);
            } catch (Exception ex1) {
                Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, ↩
                    null, ex1);
            }
        } catch (Exception ex) {
            Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ↩
                ex);
        }
        return (JCSyncObservable) soo.getNucleusObject();
    }
```

Listing 18: Creating / subscribing JCSYncObservable instance

Next pages shows full source code if this small application.

```java
package pl.edu.pjwstk.mteam.jcsync.samples.simpleChat;

import java.util.logging.Level;
import java.util.logging.Logger;
import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
import pl.edu.pjwstk.mteam.jcsync.core.implementation.util.JCSyncObservable;
import pl.edu.pjwstk.mteam.jcsync.core.implementation.util.SharedObservableObject;
import pl.edu.pjwstk.mteam.jcsync.exception.ObjectExistsException;
import pl.edu.pjwstk.mteam.jcsync.exception.ObjectNotExistsException;
import pl.edu.pjwstk.mteam.jcsync.exception.OperationForbiddenException;
import pl.edu.pjwstk.mteam.jcsync.samples.utils.BootstrapServerRunner;
import pl.edu.pjwstk.mteam.p2p.P2PNode;

public class SimpleChat {
    private BootstrapServerRunner bs;
    private JCSyncCore core1;
    private JCSyncCore core2;
    private JCSyncObservable obs_core1;
    private JCSyncObservable obs_core2;

    public SimpleChat(){
        initBootstrapServer(6060);
        initNode1(5050, 6060);
        initNode2(5055, 6060);
        snooze(1000);
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ChatWindow(getObservable(core1),"user1").setVisible(true);
            }
        });
        snooze(1000);
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ChatWindow(getObservable(core2),"user2").setVisible(true);
            }
        });
    }
    public static void main(String []args){
        new SimpleChat();
    }

    public JCSyncObservable getObservable(JCSyncCore coreAlg){
        String obs_id = "observable";
        SharedObservableObject soo = null;
        try {
            soo= new SharedObservableObject(obs_id, new JCSyncObservable(), coreAlg);
        } catch (ObjectExistsException ex) {
            try {
                soo = (SharedObservableObject) SharedObservableObject.getFromOverlay(obs_id, coreAlg);
            } catch (ObjectNotExistsException ex1) {
                Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ex1);
            } catch (OperationForbiddenException ex1) {
                Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ex1);
            } catch (Exception ex1) {
                Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ex1);
            }
        } catch (Exception ex) {
            Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ex);
        }
        return (JCSyncObservable) soo.getNucleusObject();
    }

    private void initBootstrapServer(int i) {
        //creates simple bootstrap server
        this.bs = new BootstrapServerRunner(i);
        // run bs
        this.bs.start();
    }
    private void initNode1(int i, int bootPort) {
```

```
70          P2PNode node1;
71          node1 = new P2PNode(null, P2PNode.RoutingAlgorithm.SUPERPEER);
72          node1.setServerReflexiveAddress("127.0.0.1");
73          node1.setServerReflexivePort(bootPort);
74          node1.setBootIP("127.0.0.1");
75          node1.setBootPort(bootPort);
76          node1.setUserName("user1");
77          node1.setUdpPort(i);
78          node1.networkJoin();
79          //wait for connection
80          while(!node1.isConnected()){
81              try {
82                  Thread.sleep(100);
83              } catch (InterruptedException ex) {
84                  Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, ↩
                        null, ex);
85              }
86          }
87          //creates new jcsync core instance
88          core1 = new JCSyncCore(node1, i+2);
89          try {
90              core1.init();
91          } catch (Exception ex) {
92              Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ↩
                    ex);
93          }
94      }
95      /**
96       * init first node and jcsync instance
97       * @param i port
98       */
99      private void initNode2(int i, int bootPort) {
100         P2PNode node2;
101         node2 = new P2PNode(null, P2PNode.RoutingAlgorithm.SUPERPEER);
102         node2.setServerReflexiveAddress("127.0.0.1");
103         node2.setServerReflexivePort(bootPort);
104         node2.setBootIP("127.0.0.1");
105         node2.setBootPort(bootPort);
106         node2.setUserName("user2");
107         node2.setUdpPort(i);
108         node2.networkJoin();
109         //wait for connection
110         while(!node2.isConnected()){
111             try {
112                 Thread.sleep(100);
113             } catch (InterruptedException ex) {
114                 Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, ↩
                        null, ex);
115             }
116         }
117         //creates new jcsync core instance
118         core2 = new JCSyncCore(node2, i+2);
119 
120         try {
121             core2.init();
122         } catch (Exception ex) {
123             Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ↩
                    ex);
124         }
125     }
126     private void snooze(long time){
127         try {
128             Thread.sleep(time);
129         } catch (InterruptedException ex) {
130             Logger.getLogger(SimpleChat.class.getName()).log(Level.SEVERE, null, ↩
                    ex);
131         }
132     }
133 }
```

Listing 19: SimpleChat.java source code

```
1 
2 package pl.edu.pjwstk.mteam.jcsync.samples.simpleChat;
3 
```

```
4  import java.text.DateFormat;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import java.util.Observable;
8  import java.util.Observer;
9
10 /**
11  * @author Piotr Bucior
12  */
13 public class ChatWindow extends javax.swing.JFrame implements Observer {
14     private Observable observable;
15     private String publisher;
16     /** Creates new form ChatWindow */
17     public ChatWindow(Observable obs,String publisher) {
18         initComponents();
19         this.observable = obs;
20         this.publisher = publisher;
21         this.observable.addObserver(this);
22         setTitle(publisher);
23     }
24
25     /** This method is called from within the constructor to
26      * initialize the form.
27      * WARNING: Do NOT modify this code. The content of this method is
28      * always regenerated by the Form Editor.
29      */
30     @SuppressWarnings("unchecked")
31     // <editor-fold defaultstate="collapsed" desc="Generated Code">
32     private void initComponents() {
33
34         jt_message = new javax.swing.JTextField();
35         jB_send = new javax.swing.JButton();
36         jScrollPane1 = new javax.swing.JScrollPane();
37         jTA_messages = new javax.swing.JTextArea();
38
39         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
40
41         jt_message.setText("insert message ...");
42
43         jB_send.setLabel("Send");
44         jB_send.addActionListener(new java.awt.event.ActionListener() {
45             public void actionPerformed(java.awt.event.ActionEvent evt) {
46                 jB_sendActionPerformed(evt);
47             }
48         });
49
50         jTA_messages.setColumns(20);
51         jTA_messages.setEditable(false);
52         jTA_messages.setRows(5);
53         jTA_messages.setCursor(new java.awt.Cursor(java.awt.Cursor.TEXT_CURSOR));
54         jTA_messages.setFocusable(false);
55         jScrollPane1.setViewportView(jTA_messages);
56
57         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(↩
               getContentPane());
58         getContentPane().setLayout(layout);
59         layout.setHorizontalGroup(
60             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
61             .addGroup(layout.createSequentialGroup()
62                 .addContainerGap()
63                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.↩
                       Alignment.LEADING)
64                     .addComponent(jScrollPane1, javax.swing.GroupLayout.↩
                           DEFAULT_SIZE, 402, Short.MAX_VALUE)
65                     .addGroup(layout.createSequentialGroup()
66                         .addComponent(jt_message, javax.swing.GroupLayout.↩
                               PREFERRED_SIZE, 302, javax.swing.GroupLayout.↩
                               PREFERRED_SIZE)
67                         .addPreferredGap(javax.swing.LayoutStyle.↩
                               ComponentPlacement.RELATED)
68                         .addComponent(jB_send)))
69                 .addContainerGap())
70         );
71         layout.setVerticalGroup(
72             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
73             .addGroup(layout.createSequentialGroup()
```

```
74                  .addContainerGap()
75                  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.↩
                        Alignment.BASELINE)
76                      .addComponent(jt_message, javax.swing.GroupLayout.↩
                            PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, ↩
                            javax.swing.GroupLayout.PREFERRED_SIZE)
77                      .addComponent(jB_send))
78                  .addGap(18, 18, 18)
79                  .addComponent(jScrollPane1, javax.swing.GroupLayout.↩
                        PREFERRED_SIZE, 114, javax.swing.GroupLayout.PREFERRED_SIZE)
80                  .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.↩
                        MAX_VALUE))
81          );
82
83          pack();
84      }// </editor-fold>
85
86      private void jB_sendActionPerformed(java.awt.event.ActionEvent evt) {
87          // TODO add your handling code here:
88          String message = this.publisher +" : "+this.jt_message.getText();
89          this.jt_message.setText("");
90          this.observable.notifyObservers(message);
91      }
92
93      // Variables declaration - do not modify
94      private javax.swing.JButton jB_send;
95      private javax.swing.JScrollPane jScrollPane1;
96      private javax.swing.JTextArea jTA_messages;
97      private javax.swing.JTextField jt_message;
98      // End of variables declaration
99
100     @Override
101     public void update(Observable o, Object arg) {
102         DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
103         Date date = new Date();
104         String message = (String)arg;
105         StringBuilder sb = new StringBuilder();
106         sb.append(dateFormat.format(date));
107         sb.append(" : ");
108         sb.append(message);
109         sb.append("\n");
110         this.jTA_messages.setText(this.jTA_messages.getText()+sb.toString());
111     }
112 }
```

Listing 20: ChatWindow.java source code

2 Developers cookbook

2.1 Understanding the implementation

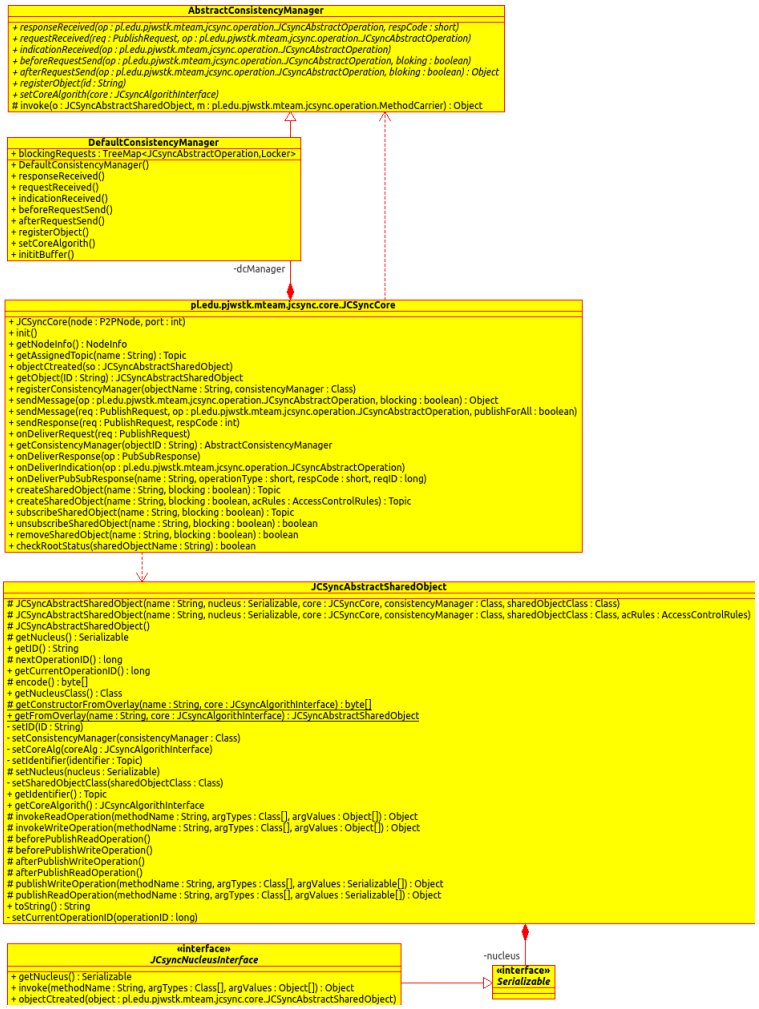

Figure 2: JCSyncArrayList implementation

2.1.1 JCSyncCore

The main component of the JCSync.Closely working with *Publish-Subscribe* layer to send messages and holds received messages. All incoming messages is forwarded to *(*AbstractConsistencyManager) subclass, where they are further processed.

2.1.2 AbstractConsistencyManager

Class represents an abstract mechanism for data integrity management.
Every action Performed on collections (after initialisation) is passed through this mechanism.
Through Appropriate implementations have the ability programmer to set priorities for specific action (for example, When We decide to set priority to invoke events for the selected node), to filter out certain requests or queuing of events to their calling in the order given.
Class methods are divided into 3 categories:
- first category provide a set of communication method between CoreAlgorith and ConsistencyManager (they are used primarily to transmit received messages), they include:
- *indicationReceived*

- *requestReceived*
- *responseReceived*
- *registerObject*
- *setCoreAlgorith*

- second one provide a mechanism for communication between JCSyncAbstractSharedObject (and its derivatives) and ConsistencyManager. That can be used for example to suspend the worker threads until a response and/or indication is received. These are:
- *beforeRequestSend*
- *afterRequestSend*

- the third category is a set of internal methods for better transparency of the implementation:
- *invoke*

2.1.3 DefaultConsistencyManager

Basic implementation of AbstractConsistencyManager. Its represents FIFO (first-in-first-out) policy for managing operation.

2.1.4 JCSyncAbstractSharedObject

Class that represents the shared object in the layer. It stores all the necessary information, such as object ID, the ConsistensyManager, nucleus object on which we make the operations that are published in the layer. There is two way to get instance of this class:
1. by calling the constructor in the subclass

```
1  class SimpleSharedObject extends JCSyncAbstractSharedObject implements List {
2
3    public SimpleSharedObject(String name, JCSyncCore core, Class ↩
         consistencyManager, AccessControlRules acRules) throws ↩
         ObjectExistsException, Exception {
4        super(name, new ArrayList(), core, consistencyManager, SimpleSharedObject.↩
             class, acRules);
5    }
6
7    public SimpleSharedObject(String name, JCSyncCore core, Class ↩
         consistencyManager) throws ObjectExistsException, Exception {
8        super(name, new ArrayList(), core, consistencyManager, SimpleSharedObject.↩
             class);
9    }
10 [...]
11 }
```

Above code allows to create new object in the overlay, if the object with given name already exists then ObjectExistsException is thrown. 2. or by calling static getFromOverlay method

```
1 P2PNode node;
2 JCSyncCore core;
3 [...]
4 String name = "existent\_shared\_object";
5 JCSyncAbstractSharedObject s2 = null;
6 s2 = JCSyncAbstractSharedObject.getFromOverlay(name, core);
7 [...]
```

Used only if the shared object is already known in the overlay.

JCSyncAbstractSharedObject also provides skeleton mechanism to invoke methods on the JCSyncNucleusInterface object.

2.1.5 JCSyncNucleusInterface

The JCsyncNucleusInterface provides skeleton functionality of JCSync mechanism for implemented collections and other extensions.

It describes 3 methods:
java.io.Serializable getNucleus()

Returns a nucleus object associated with current shared object. Typically for implemented collections classes it will returns a super class of current implementation,for example in the JCSyncArrayList method looks like below:

```
public Serializable getNucleus() {
            return (ArrayList) this;
      }
```

The second one:

java.lang.Object invoke(java.lang.String methodName,
java.lang.Class[] argTypes,
java.lang.Object[] argValues,
boolean local)

Allows to invoke method delivered from the overlay.

And the last is:

objectCreated(JCSyncAbstractSharedObject object)

Which informs *JCSyncNucleusInterface* about the associated shared object is already created.

2.1.6 Understanding collection implementation with JCSyncArrayList as an example

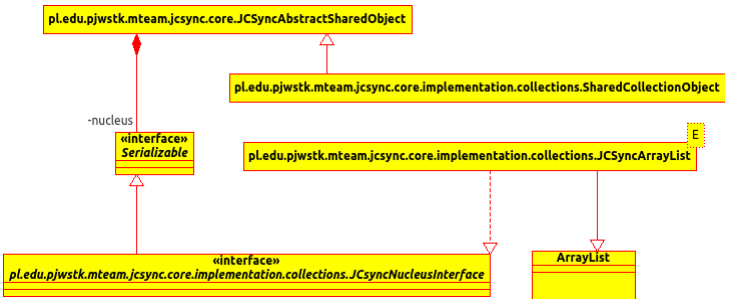

Figure 3: JCSyncArrayList implementation

The methods of implementation will be presented on the already implemented ArrayList collection with detailing of the basic aspects of the implementation.

At the outset we need to define which methods will be overloaded and used in the final application.Also we need to make sure that our selection method did not produce a mutually - In this case, not all should benefit from the mechanism JCSync.
In view of the reproducibility cited example is limited to two methods, one for write operations and one for read operations.
Full source code will be listed below containing the constructors defined in the ArrayList class provided by the JDK and the implementations of methods defined in the interface JCsyncNucleusInterface – these methods will be used to manage the collection of the mechanism JCSync.

```
public class JCSyncArrayList<E> extends ArrayList<E> implements ↩
    JCsyncNucleusInterface{

private Object shared_object = null;

public JCSyncArrayList(Collection<? extends E> c) {
        super(c);
}
public JCSyncArrayList() {
        super();
}
public JCSyncArrayList(int initialCapacity) {
        super(initialCapacity);
}

@Override
public Serializable getNucleus() {
        return (ArrayList) this;
}

@Override
public Object invoke(String methodName, Class[] argTypes, Object[] argValues) {
        methodName = methodName + '_';
        Object retVal = null;
        Method m = null;
        try {
            if (argTypes != null && argTypes.length > 0) {
                m = getClass().getDeclaredMethod(methodName, argTypes);
                m.setAccessible(true);
                retVal = m.invoke(this, argValues);
            } else {
                m = getClass().getDeclaredMethod(methodName);
                m.setAccessible(true);
                retVal = m.invoke(this);
            }

        } catch (Exception e) {
```

```
            retVal = e;
        }
        return retVal;
    }

@Override
public void objectCtreated(JCSyncAbstractSharedObject object) {
        this.shared_object = (SharedCollectionObject) object;
    }

private void writeObject(ObjectOutputStream ostr) throws IOException {
        ArrayList m = (ArrayList) this;
        ostr.writeObject(m);
        //else  ostr.write(0);
        }
@Override
    public void add(int index, E element) {
        try {
            Class[] aT = {int.class,Object.class};
            Serializable[] aV = {index,(Serializable) element};
            ((SharedCollectionObject) shared_object).publishWriteOperation("add",↩
                 aT, aV);
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new IllegalArgumentException(ex.getMessage());
        }
    }

    private void add_(int index, E element) {
        super.add(index, element);
    }
@Override
    public boolean contains(Object o) {
        return super.contains(o);
    }
}
```

Listing 21: JCSyncArrayList.java source code

The figure of read-type method is very simple, simply invokes the parent class and the action is over.
Write-type method used in this example is a **add(int index, E element)** method. In the method body there are grouping given arguments and pass them to further processing to an object collection management:
((SharedCollectionObject) shared_object)
.publishWriteOperation("add", aT, aV);
At this moment the worker thread is locked. When the executive message is received it is passed to the method:
public Object invoke(String methodName,
Class[] argTypes, Object[] argValues),
where to the method name is added suffix defined as a character „_" which allows to maintain a methods naming similarity. The result of this action is invoking method called:
add_(int index, E element) and then the worker thread is resumed.

Full source code of JCSyncArrayList are listed below.

```
1  package pl.edu.pjwstk.mteam.jcsync.core.implementation.collections;
2
3  import java.io.IOException;
4  import java.io.ObjectOutputStream;
5  import java.io.Serializable;
6  import java.lang.reflect.Method;
7  import java.util.ArrayList;
8  import java.util.Collection;
9  import pl.edu.pjwstk.mteam.jcsync.core.JCSyncAbstractSharedObject;
10
11 /**
12  * A subclass of <tt>ArrayList<E></tt> which provides synchronisation mechanism
13  * over the network layer.
```

```
14  *
15  * @author Piotr Bucior
16  * @serial
17  */
18 public class JCSyncArrayList<E> extends ArrayList<E> implements ↩
        JCsyncNucleusInterface{
19
20 private Object shared_object = null;
21
22
23     public JCSyncArrayList(Collection<? extends E> c) {
24         super(c);
25     }
26
27     public JCSyncArrayList() {
28         super();
29     }
30
31     public JCSyncArrayList(int initialCapacity) {
32         super(initialCapacity);
33     }
34
35     @Override
36     public boolean add(E e) {
37         Boolean retVal = false;
38         try {
39             Class[] aT = {Object.class};
40             Serializable[] aV = {(Serializable) e};
41             retVal = (Boolean)((SharedCollectionObject) shared_object).↩
                   publishWriteOperation("add", aT, aV);
42         } catch (Exception ex) {
43             ex.printStackTrace();
44             throw new IllegalArgumentException(ex.getMessage());
45         }
46         return retVal.booleanValue();
47     }
48
49     private boolean add_(E e) {
50         return super.add(e);
51     }
52
53     @Override
54     public void add(int index, E element) {
55         try {
56             Class[] aT = {int.class,Object.class};
57             Serializable[] aV = {index,(Serializable) element};
58             ((SharedCollectionObject) shared_object).publishWriteOperation("add",↩
                    aT, aV);
59         } catch (Exception ex) {
60             ex.printStackTrace();
61             throw new IllegalArgumentException(ex.getMessage());
62         }
63     }
64
65     private void add_(int index, E element) {
66         super.add(index, element);
67     }
68
69     @Override
70     public boolean addAll(Collection<? extends E> c) {
71         Boolean retVal = false;
72         try {
73             Class[] aT = {Collection.class};
74             Serializable[] aV = {(Serializable) c};
75             retVal = (Boolean)((SharedCollectionObject) shared_object).↩
                   publishWriteOperation("addAll", aT, aV);
76         } catch (Exception ex) {
77             ex.printStackTrace();
78             throw new IllegalArgumentException(ex.getMessage());
79         }
80         return retVal.booleanValue();
81     }
82     private boolean addAll_(Collection<? extends E> c) {
83         return super.addAll(c);
84     }
85
```

```
86      @Override
87      public boolean addAll(int index, Collection<? extends E> c) {
88          Boolean retVal = false;
89          try {
90              Class[] aT = {int.class,Collection.class};
91              Serializable[] aV = {index,(Serializable) c};
92              retVal = (Boolean)((SharedCollectionObject) shared_object).↩
                    publishWriteOperation("addAll", aT, aV);
93          } catch (Exception ex) {
94              ex.printStackTrace();
95              throw new IllegalArgumentException(ex.getMessage());
96          }
97          return retVal.booleanValue();
98      }
99
100     private boolean addAll_(int index, Collection<? extends E> c) {
101         return super.addAll(index, c);
102     }
103
104     @Override
105     public void clear() {
106         try {
107             ((SharedCollectionObject) shared_object).publishWriteOperation("clear↩
                    ", null, null);
108         } catch (Exception ex) {
109             ex.printStackTrace();
110             throw new IllegalArgumentException(ex.getMessage());
111         }
112     }
113
114     private void clear_() {
115         super.clear();
116     }
117
118     @Override
119     public Object clone() {
120         ArrayList retVal = new ArrayList(this.size());
121         retVal.addAll(this);
122         return retVal;
123     }
124
125     @Override
126     public boolean contains(Object o) {
127         return super.contains(o);
128     }
129
130     @Override
131     public void ensureCapacity(int minCapacity) {
132         super.ensureCapacity(minCapacity);
133     }
134
135     private void ensureCapacity_(int minCapacity) {
136         super.ensureCapacity(minCapacity);
137     }
138
139     @Override
140     public E get(int index) {
141         return super.get(index);
142     }
143
144     @Override
145     public int indexOf(Object o) {
146         return super.indexOf(o);
147     }
148
149     @Override
150     public boolean isEmpty() {
151         return super.isEmpty();
152     }
153
154     @Override
155     public int lastIndexOf(Object o) {
156         return super.lastIndexOf(o);
157     }
158
159     @Override
```

```
160     public E remove(int index) {
161         E retVal = null;
162         try {
163             Class[] aT = {int.class};
164             Serializable[] aV = {index};
165             retVal = (E) ((SharedCollectionObject) shared_object).↩
                    publishWriteOperation("remove", aT, aV);
166         } catch (Exception ex) {
167             throw new IllegalArgumentException(ex.getMessage());
168         }
169         return retVal;
170     }
171 
172     private E remove_(int index) {
173         return super.remove(index);
174     }
175 
176     @Override
177     public boolean remove(Object o) {
178         Boolean retVal = false;
179         try {
180             Class[] aT = {Object.class};
181             Serializable[] aV = {(Serializable)o};
182             retVal = (Boolean)((SharedCollectionObject) shared_object).↩
                    publishWriteOperation("remove", aT, aV);
183         } catch (Exception ex) {
184             ex.printStackTrace();
185             throw new IllegalArgumentException(ex.getMessage());
186         }
187         return retVal.booleanValue();
188     }
189     private boolean remove_(Object o) {
190         return super.remove(o);
191     }
192 
193     @Override
194     protected void removeRange(int fromIndex, int toIndex) {
195         try {
196             Class[] aT = {int.class,int.class};
197             Serializable[] aV = {fromIndex,toIndex};
198             ((SharedCollectionObject) shared_object).publishWriteOperation("↩
                    removeRange", aT, aV);
199         } catch (Exception ex) {
200             throw new IllegalArgumentException(ex.getMessage());
201         }
202     }
203 
204     private void removeRange_(int fromIndex, int toIndex) {
205         super.removeRange(fromIndex, toIndex);
206     }
207 
208     @Override
209     public E set(int index, E element) {
210         E retVal = null;
211         try {
212             Class[] aT = {int.class,Object.class};
213             Serializable[] aV = {index,(Serializable)element};
214             retVal = (E) ((SharedCollectionObject) shared_object).↩
                    publishWriteOperation("set", aT, aV);
215         } catch (Exception ex) {
216             throw new IllegalArgumentException(ex.getMessage());
217         }
218         return retVal;
219     }
220     private E set_(int index, E element) {
221         return super.set(index, element);
222     }
223 
224     @Override
225     public int size() {
226         return super.size();
227     }
228 
229     @Override
230     public Object[] toArray() {
231         return super.toArray();
```

```
232     }
233
234     @Override
235     public <T> T[] toArray(T[] a) {
236         return super.toArray(a);
237     }
238
239     @Override
240     public void trimToSize() {
241         super.trimToSize();
242     }
243
244     //--------------------------------------------------------
245     //JCSync code
246     //--------------------------------------------------------
247
248     @Override
249     public Serializable getNucleus() {
250         return (ArrayList) this;
251     }
252
253     @Override
254     public Object invoke(String methodName, Class[] argTypes, Object[] argValues,↩
          boolean local) {
255         methodName = methodName + '_';
256         Object retVal = null;
257         Method m = null;
258         try {
259             if (argTypes != null && argTypes.length > 0) {
260                 m = getClass().getDeclaredMethod(methodName, argTypes);
261                 m.setAccessible(true);
262                 retVal = m.invoke(this, argValues);
263             } else {
264                 m = getClass().getDeclaredMethod(methodName);
265                 m.setAccessible(true);
266                 retVal = m.invoke(this);
267             }
268
269         } catch (Exception e) {
270             retVal = e;
271         }
272         return retVal;
273     }
274
275     @Override
276     public void objectCtreated(JCSyncAbstractSharedObject object) {
277         this.shared_object = (SharedCollectionObject) object;
278     }
279
280     @Override
281     public boolean equals(Object obj) {
282         if (obj == null) {
283             return false;
284         }
285         if (getClass() != obj.getClass()) {
286             return false;
287         }
288         final JCSyncArrayList<E> other = (JCSyncArrayList<E>) obj;
289         return true;
290     }
291
292     @Override
293     public int hashCode() {
294         return super.hashCode();
295     }
296     /**
297      * Writes this object to the stream as a super class.<br>
298      * <pre>
299      * ArrayList m = (ArrayList) this;
300        ostr.writeObject(m);
301      * </pre>
302      */
303     private void writeObject(ObjectOutputStream ostr) throws IOException {
304         ArrayList m = (ArrayList) this;
305         ostr.writeObject(m);
306         //else  ostr.write(0);
```

```
307         }
308 }
```

Listing 22: JCSyncArrayList.java full source code

2.1.7 Understanding JCSyncAbstractSharedObject object

This object provides a set of methods used to publish request about invoked method and allows to invoke received methods on its owned nucleus object (e.g. on the JCSyncArrayList). Every actions which will be made on our nucleus object is passed by this object to or from ConsistencyManager instance.

beforePublishReadOperation()
- always invoked when the lock is required, after this method and until afterPublishReadOperation is called all incoming requests is passed to the buffer and waits until release.

afterPublishReadOperation()
- invoked to release the locker on object root node.

beforePublishWriteOperation()
- always invoked when the lock is required, after this method and until afterPublishWriteOperation is called all incoming requests is passed to the buffer and waits until release.

afterPublishWriteOperation()
- Invoked to release the locker.

getConstructorFromOverlay(java.lang.String name,JCsyncAlgorithInterface core)
- returns object representation as byte array, which is received from the layer.

getFromOverlay(java.lang.String name, JCsyncAlgorithInterface core)
- used to subscribe with existing shared object.

invokeReadOperation(java.lang.String methodName, java.lang.Class[] argTypes, java.lang.Object[] argValues)
- called when the 'read-type' operation (method) is received from the layer and must be invoked on the nucleus object instance.

invokeWriteOperation(java.lang.String methodName, java.lang.Class[] argTypes, java.lang.Object[] argValues, boolean local)
- called when the 'write-type' operation (method) is received from the layer and must be invoked on the nucleus object instance.

publishReadOperation(java.lang.String methodName, java.lang.Class[] argTypes, java.io.Serializable[] argValues)
- invoked to publish method to invoke with given arguments.

publishWriteOperation(java.lang.String methodName, java.lang.Class[] argTypes, java.io.Serializable[] argValues)
- invoked to publish method to invoke with given arguments.

2.2 Flow diagrams

2.2.1 Creating new object scenario

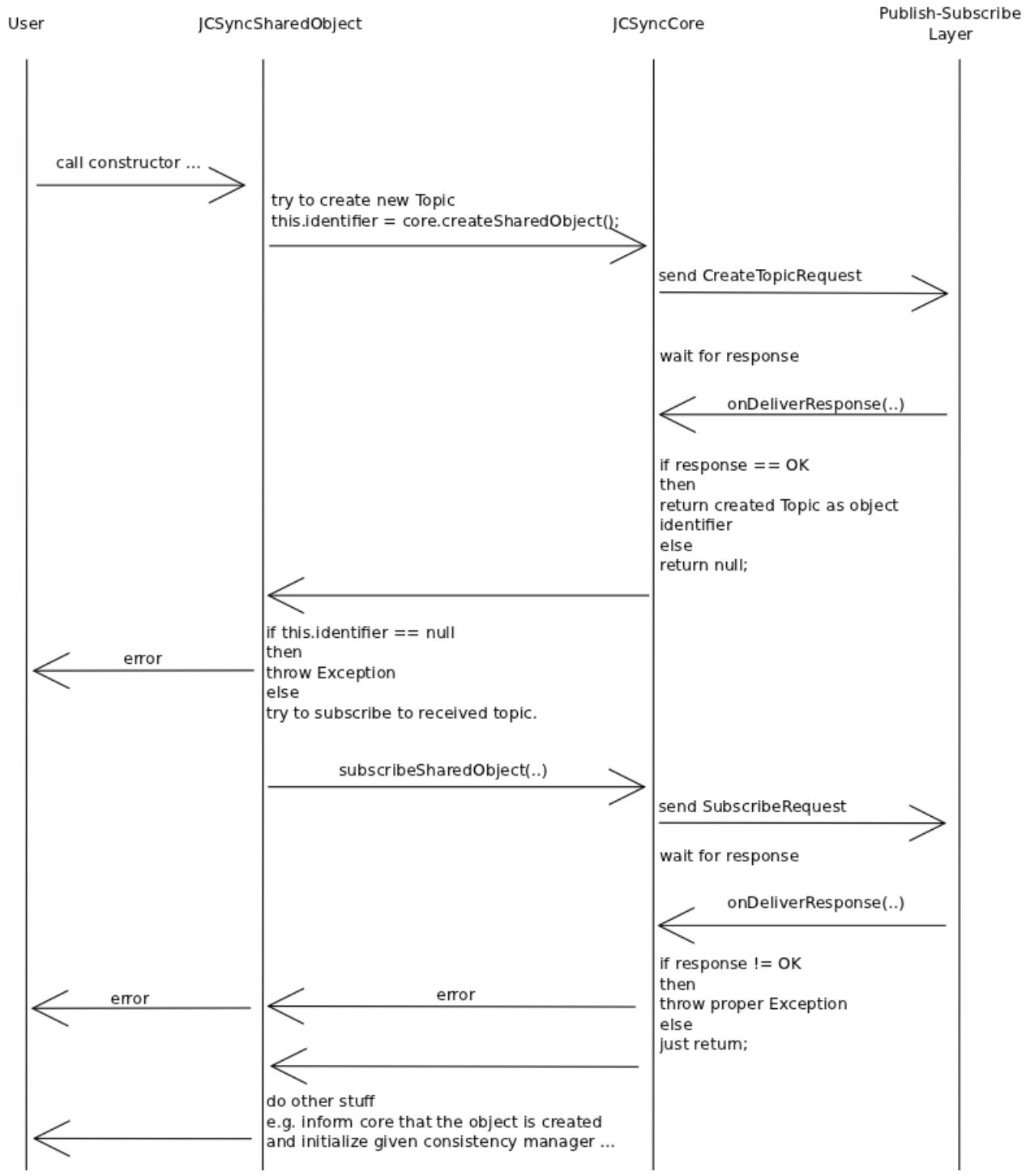

Figure 4: Creating new shared object scenario.

2.2.2 Subscribing to the object scenario

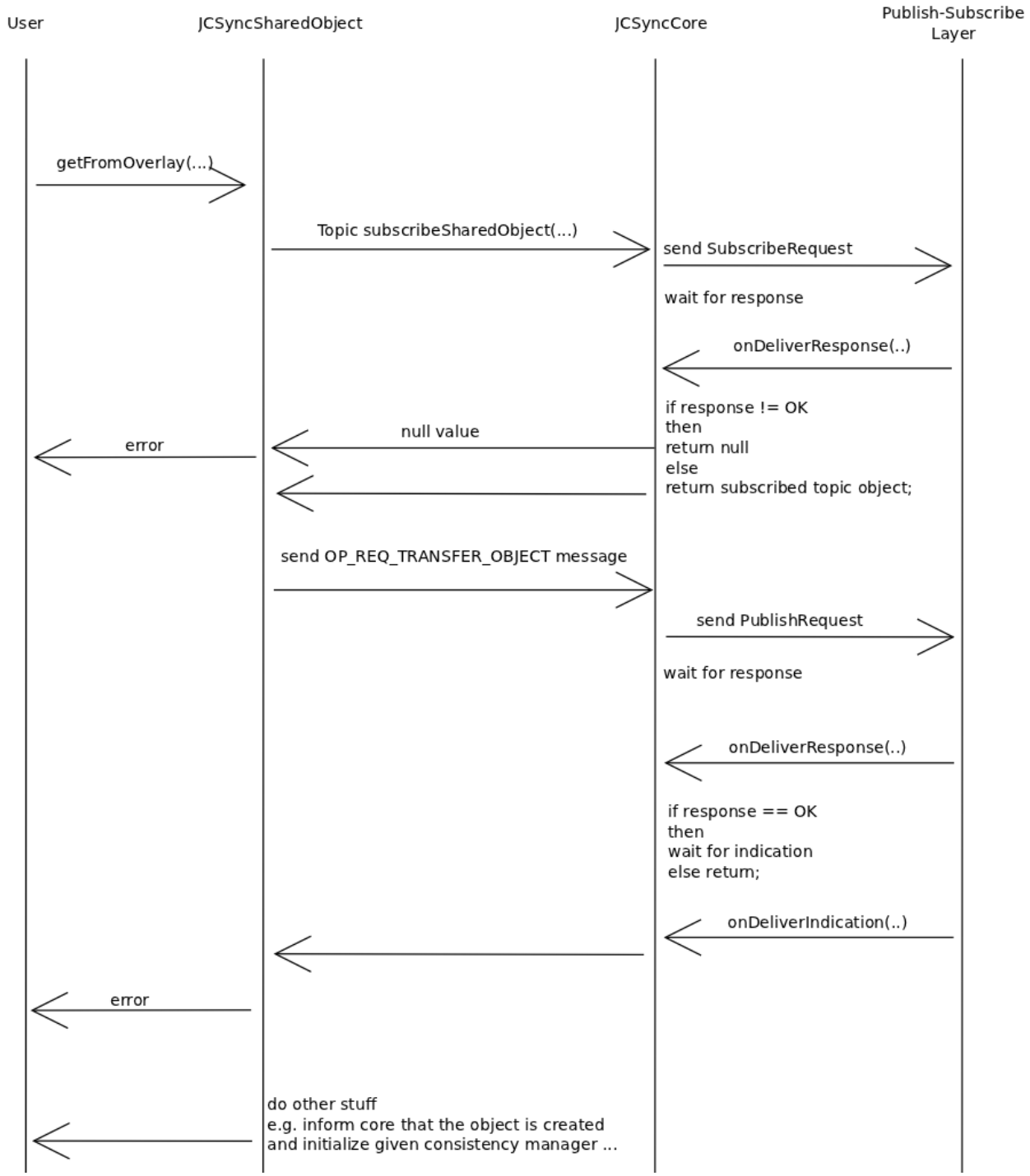

Figure 5: Subscribing to shared object scenario.

2.2.3 Method invocation scenario

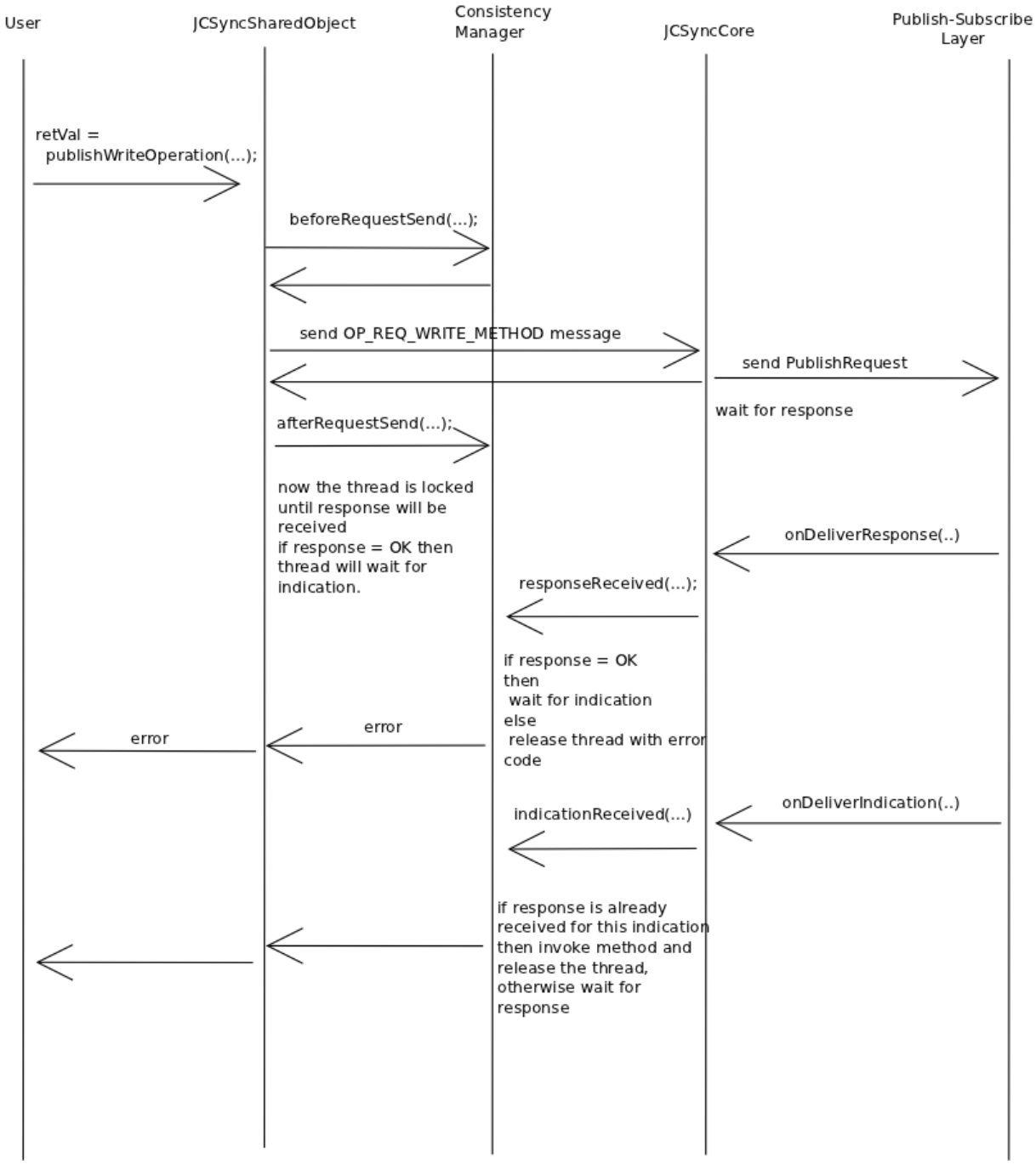

Figure 6: Method publishing and invoking scenario.

2.3 Extending JCSync

2.3.1 How to implement own synchronized object

Lets look at the java.util.concurrent.Exchanger class[3]. Its allow to exchange data between two threads.
Lets imagine now that we want to implement this feature to exchange data between two nodes in the layer. In this simple case one of them will be filling some data and the second will be erasing this data. This sample usage is shows below:

```
1  class FillAndEmpty {
2     Exchanger<DataBuffer> exchanger = new Exchanger<DataBuffer>();
3     DataBuffer initialEmptyBuffer = ... a made-up type
4     DataBuffer initialFullBuffer = ...
5
6     class FillingLoop implements Runnable {
7       public void run() {
8         DataBuffer currentBuffer = initialEmptyBuffer;
9         try {
10          while (currentBuffer != null) {
11            addToBuffer(currentBuffer);
12            if (currentBuffer.isFull())
13              currentBuffer = exchanger.exchange(currentBuffer);
14          }
15        } catch (InterruptedException ex) { ... handle ... }
16      }
17    }
18
19    class EmptyingLoop implements Runnable {
20      public void run() {
21        DataBuffer currentBuffer = initialFullBuffer;
22        try {
23          while (currentBuffer != null) {
24            takeFromBuffer(currentBuffer);
25            if (currentBuffer.isEmpty())
26              currentBuffer = exchanger.exchange(currentBuffer);
27          }
28        } catch (InterruptedException ex) { ... handle ...}
29      }
30    }
31
32    void start() {
33      new Thread(new FillingLoop()).start();
34      new Thread(new EmptyingLoop()).start();
35    }
36  }
```

Listing 23: java.util.concurrent.Exchanger sample usage

From the beginning we need to implement own instance of
pl.edu.pjwstk.mteam.jcsync.core.JCSyncAbstractSharedObject
which will manage our Exchanger implementation.
We call it *SharedExchangerObject.java*.
Source code is copy-based from the *SharedCollectionObject.java* with the exception of one method:
Object publishWriteOperation
(String methodName, Class[] argTypes, Serializable[] argValues)
Lets see how its body looks in the super class:

```
1 protected Object publishWriteOperation(String methodName, Class[] argTypes, ↩
    Serializable[] argValues) throws Exception{
2       MethodCarrier mc = new MethodCarrier(methodName);
3       mc.setArgTypes(argTypes);
4       mc.setArgValues(argValues);
5       JCsyncAbstractOperation op = JCsyncAbstractOperation.getByType(↩
            RegisteredOperations.OP_REQ_WRITE_METHOD, this.ID, mc,this.coreAlg.↩
            getNodeInfo().getName());
```

[3] JDK Exchanger specification

```
6           this.coreAlg.getConsistencyManager(this.ID).beforeRequestSend(op, true);
7           this.coreAlg.sendMessage(op, false);
8           Object e = this.coreAlg.getConsistencyManager(this.ID).afterRequestSend(↩
                op, true);
9           if(e!= null && e instanceof Exception) throw (Exception)e;
10          else return e;
11      }
```

Listing 24: publishWriteOperation(...) method body in the JCSyncAbstractSharedObject class

There is two interesting code lines, first of them:
this.coreAlg.getConsistencyManager(this.ID).beforeRequestSend(op, true);
means that *ConsistencyManager* will be informed to create thread lock for current thread - this lock will be released when the operation will be received from the network. Second line:
Object e = this.coreAlg.getConsistencyManager(this.ID).afterRequestSend(op, true);
is responsible for acquiring thread locker. After release the result of invoked method is returned (e).
In our case we don't need to use lockers for working threads - they are already locked by Exchanger implementation. Full code is shows below:

```
1       package pl.edu.pjwstk.mteam.jcsync.samples.Exchanger;
2
3   import java.io.Serializable;
4   import java.lang.reflect.InvocationTargetException;
5   import pl.edu.pjwstk.mteam.jcsync.core.AccessControlLists;
6   import pl.edu.pjwstk.mteam.jcsync.core.JCSyncAbstractSharedObject;
7   import pl.edu.pjwstk.mteam.jcsync.core.JCSyncCore;
8   import pl.edu.pjwstk.mteam.jcsync.core.consistencyManager.↩
        DefaultConsistencyManager;
9   import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.↩
        JCsyncNucleusInterface;
10  import pl.edu.pjwstk.mteam.jcsync.exception.ObjectExistsException;
11  import pl.edu.pjwstk.mteam.jcsync.operation.JCsyncAbstractOperation;
12  import pl.edu.pjwstk.mteam.jcsync.operation.MethodCarrier;
13  import pl.edu.pjwstk.mteam.jcsync.operation.RegisteredOperations;
14
15  /**
16   * @author Piotr Bucior
17   */
18  public class SharedExchangerObject extends JCSyncAbstractSharedObject{
19
20      /**
21       * Creates new instance with blank constructor.
22       */
23      public SharedExchangerObject(){
24
25      }
26      /**
27       * Creates new instance with given arguments.
28       * @param name the name of new shared object
29       * @param nucleus the object on which we the method are called
30       * @param core the core algorithm instance
31       * @throws ObjectExistsException if the object with this name is already ↩
             created in the network
32       * @throws Exception any occurred exception
33       */
34      public SharedExchangerObject(String name, JCsyncNucleusInterface nucleus, ↩
            JCSyncCore core) throws ObjectExistsException, Exception{
35          super(name, nucleus, core, DefaultConsistencyManager.class, ↩
                SharedExchangerObject.class);
36          nucleus.objectCtreated(this);
37      }
38      /**
39       * Creates new instance with given arguments.
40       * @param name the name of new shared object
41       * @param nucleus the object on which we the method are called
42       * @param core the core algorithm instance
43       * @param acRules customised access control rules
```

```
44      * @throws ObjectExistsException if the object with this name is already ↩
            created in the network
45      * @throws Exception any occurred exception
46      */
47     public SharedExchangerObject(String name, JCsyncNucleusInterface nucleus, ↩
            JCSyncCore core, AccessControlLists acRules) throws ObjectExistsException↩
            , Exception{
48         super(name, nucleus, core, DefaultConsistencyManager.class, ↩
               SharedExchangerObject.class,acRules);
49         nucleus.objectCtreated(this);
50     }
51     /**
52      * Creates new instance with given arguments.
53      * @param name the name of new shared object
54      * @param nucleus the object on which we the method are called
55      * @param core the core algorithm instance
56      * @param consistencyManager class of consistency manager which will hold ↩
            this object
57      * @throws ObjectExistsException if the object with this name is already ↩
            created in the network
58      * @throws Exception any occurred exception
59      */
60     public SharedExchangerObject(String name, JCsyncNucleusInterface nucleus, ↩
            JCSyncCore core, Class consistencyManager) throws ObjectExistsException, ↩
            Exception{
61         super(name, nucleus, core, consistencyManager, SharedExchangerObject.↩
               class);
62         nucleus.objectCtreated(this);
63     }
64     /**
65      * Creates new instance with given arguments.
66      * @param name the name of new shared object
67      * @param nucleus the object on which we the method are called
68      * @param core the core algorithm instance
69      * @param consistencyManager class of consistency manager which will hold ↩
            this object
70      * @param acRules customised access control rules
71      * @throws ObjectExistsException if the object with this name is already ↩
            created in the network
72      * @throws Exception any occurred exception
73      */
74     public SharedExchangerObject(String name, JCsyncNucleusInterface nucleus, ↩
            JCSyncCore core, Class consistencyManager,AccessControlLists acRules) ↩
            throws ObjectExistsException, Exception{
75         super(name, nucleus, core, consistencyManager, SharedExchangerObject.↩
               class,acRules);
76         nucleus.objectCtreated(this);
77     }
78     @Override
79     protected Object publishReadOperation(String methodName, Class[] argTypes, ↩
            Serializable[] argValues) throws Exception {
80         return super.publishReadOperation(methodName, argTypes, argValues);
81     }
82
83     @Override
84     protected Object publishWriteOperation(String methodName, Class[] argTypes, ↩
            Serializable[] argValues) throws Exception {
85         //return super.publishWriteOperation(methodName, argTypes, argValues);
86         MethodCarrier mc = new MethodCarrier(methodName);
87         mc.setArgTypes(argTypes);
88         mc.setArgValues(argValues);
89         JCsyncAbstractOperation op = JCsyncAbstractOperation.getByType(↩
               RegisteredOperations.OP_REQ_WRITE_METHOD, super.getID(), mc,this.↩
               coreAlg.getNodeInfo().getName());
90         //this.coreAlg.getConsistencyManager(this.ID).beforeRequestSend(op, true)↩
               ;
91         this.coreAlg.sendMessage(op, false);
92         //Object e = this.coreAlg.getConsistencyManager(this.ID).afterRequestSend↩
               (op, true);
93         //if(e!= null && e instanceof Exception) throw (Exception)e;
94         //else return e;
95         return null;
96     }
97
98     @Override
```

```
99      protected Object invokeReadOperation(String methodName, Class[] argTypes, ↩
            Object[] argValues) throws SecurityException, NoSuchMethodException, ↩
            IllegalAccessException, IllegalArgumentException, ↩
            InvocationTargetException {
100         return ((JCsyncNucleusInterface)getNucleusObject()).invoke(methodName, ↩
                argTypes, argValues, false);
101     }
102
103     @Override
104     protected Object invokeWriteOperation(String methodName, Class[] argTypes, ↩
            Object[] argValues, boolean local) throws SecurityException, ↩
            NoSuchMethodException, IllegalAccessException, IllegalArgumentException, ↩
            InvocationTargetException {
105         super.nextOperationID();
106         return ((JCsyncNucleusInterface)getNucleusObject()).invoke(methodName, ↩
                argTypes, argValues, local);
107     }
108
109     /**
110      * Returns an <tt>JCSyncNucleusInterface</tt> extended object.
111      * @return object associated with current shared object.
112      */
113     public Object getNucleusObject(){
114         return super.getNucleus();
115     }
116
117     /**
118      * Sets nucleus object for current shared object.
119      * @param nucleus an <tt>JCSyncNucleusInterface</tt> extension.
120      */
121     @Override
122     protected void setNucleus(Serializable nucleus) {
123         super.setNucleus(nucleus);
124         ((JCsyncNucleusInterface)nucleus).objectCtreated(this);
125     }
126
127 }
```

Listing 25: SharedExchangerObject.java source code

Lets define now our Exchanger sub-class:

```
1       public class SharedExchanger<V> extends Exchanger<V> {
2       private Object shared_object = null;
3
4       public SharedExchanger(){
5           super();
6       }
7
8       @Override
9       public V exchange(V x) throws InterruptedException {
10          @SuppressWarnings("unchecked")
11          V retVal = null;
12          try {
13              Class[] aT = {Object.class};
14              Serializable[] aV = {(Serializable) x};
15
16              retVal = (V) ((SharedExchangerObject) shared_object).↩
                    publishWriteOperation("exchange", aT, aV);
17          } catch (Exception ex) {
18              ex.printStackTrace();
19              throw new IllegalArgumentException(ex.getMessage());
20          }
21          return super.exchange(x);
22      }
23      public V exchange_(V x) throws InterruptedException {
24          return super.exchange(x);
25      }
```

Listing 26: Exchanger sub-class

There is two interesting methods, the first one is overriden **exchange(...)** method. There are assembled information about given arguments and method is published over the layer. The second one method which is called **exchange_(...)** is called by ConsistencyManager

instance when the method with the same name but without suffix was received from the layer.

Now we must implement methods delivered by *(*JCSyncNucleusInterface interface), there is 3 methods (**plus one extra method which is also required - the writeObject(...) method**):

```
1   public class SharedExchanger<V> extends Exchanger<V> implements ↩
        JCsyncNucleusInterface{
2   private Object shared_object = null;
3
4   public SharedExchanger(){
5       super();
6   }
7
8   @Override
9   public Serializable getNucleus() {
10      return this;
11  }
12
13  @Override
14  public Object invoke(String methodName, Class[] argTypes, Object[] argValues,↩
        boolean local) {
15      Object retVal = null;
16      if(!local){
17          System.out.println("method will be invoked");
18      methodName = methodName + '_';
19      Method[] allMethods = getClass().getDeclaredMethods();
20      Method m = null;
21      try {
22          if (argTypes != null && argTypes.length > 0) {
23              m = getClass().getDeclaredMethod(methodName, argTypes);
24              m.setAccessible(true);
25              retVal = m.invoke(this, argValues);
26          } else {
27              m = getClass().getDeclaredMethod(methodName);
28              m.setAccessible(true);
29              retVal = m.invoke(this);
30          }
31
32      } catch (Exception e) {
33          retVal = e;
34      }
35      return retVal;
36      }
37      else{
38          System.out.println("method will not be invoked");
39      }
40      return retVal;
41  }
42
43  @Override
44  public void objectCtreated(JCSyncAbstractSharedObject object) {
45      this.shared_object = (SharedExchangerObject) object;
46  }
47  private void writeObject(ObjectOutputStream ostr) throws IOException {
48      Exchanger m = (Exchanger) this;
49      ostr.writeObject(m);
50      //else  ostr.write(0);
51  }
52  }
```

Listing 27: SharedExchanger.java source code

Take a look at the **invoke(...)** method.The *local* argument shows if the delivered method is from local machine or is received from the network layer. If its **true** then delivered method will not be invoked (the thread is already suspended by calling exchange(...) method) - only method not from the local machine will be invoked.
Now we have already written our implementation and we can use it:

```
    \\...
        SharedExchanger<String> exch1;
        SharedExchanger<String> exch2;
        String name = "test_exch";
        SharedExchangerObject so_1;
        SharedExchangerObject so_2;
        exch1 = new SharedExchanger<String>();
        // creating
        so_1 = new SharedExchangerObject(name, exch1, core);
        // subscribing on the second node
        so_2 = (SharedExchangerObject) SharedCollectionObject.getFromOverlay(name, core2);
        exch2 = (SharedExchanger<String>) so_2.getNucleusObject();
        FillingLoop fl = new FillingLoop((SharedExchanger) so_1.getNucleusObject());
        EmptyingLoop el = new EmptyingLoop((SharedExchanger) so_2.getNucleusObject());
        new Thread(fl).start();
        new Thread(el).start();
        // the producer and consumer threads
    class FillingLoop implements Runnable {

        private Exchanger<String> exchanger;

        public FillingLoop(Exchanger e) {
            this.exchanger = e;
        }

        public void run() {
            StringBuilder currentBuffer = new StringBuilder();
            try {
                while (currentBuffer != null) {
                    currentBuffer.append("test");
                    currentBuffer = new StringBuilder(exchanger.exchange(currentBuffer.toString()));
                    System.out.println("received data "+currentBuffer);
                    assertTrue("currentBuffer lenght must be ==0", currentBuffer.length() == 0);
                }
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }

    class EmptyingLoop implements Runnable {

        private Exchanger<String> exchanger;

        public EmptyingLoop(Exchanger e) {
            this.exchanger = e;
        }

        public void run() {
            StringBuilder currentBuffer = new StringBuilder();
            try {
                while (currentBuffer != null) {
                    currentBuffer = new StringBuilder(exchanger.exchange(""));
                    assertTrue("current buffer lenght must be > 0", currentBuffer.length() > 0);
                    Thread.sleep(3000);
                }
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
```

Listing 28: SharedExchanger usage example

The full source code of SharedExchanger.java:

```
package pl.edu.pjwstk.mteam.jcsync.samples.Exchanger;

import java.io.IOException;
```

```
4  import java.io.ObjectOutputStream;
5  import java.io.Serializable;
6  import java.lang.reflect.Method;
7  import java.util.HashMap;
8  import java.util.concurrent.Exchanger;
9  import pl.edu.pjwstk.mteam.jcsync.core.JCSyncAbstractSharedObject;
10 import pl.edu.pjwstk.mteam.jcsync.core.implementation.collections.↩
       JCsyncNucleusInterface;
11 public class SharedExchanger<V> extends Exchanger<V> implements ↩
       JCsyncNucleusInterface{
12     private Object shared_object = null;
13
14     public SharedExchanger(){
15         super();
16     }
17
18     @Override
19     public V exchange(V x) throws InterruptedException {
20         @SuppressWarnings("unchecked")
21         V retVal = null;
22         try {
23             Class[] aT = {Object.class};
24             Serializable[] aV = {(Serializable) x};
25
26             retVal = (V) ((SharedExchangerObject) shared_object).↩
                   publishWriteOperation("exchange", aT, aV);
27         } catch (Exception ex) {
28             ex.printStackTrace();
29             throw new IllegalArgumentException(ex.getMessage());
30         }
31         return super.exchange(x);
32     }
33     public V exchange_(V x) throws InterruptedException {
34         return super.exchange(x);
35     }
36
37     @Override
38     public Serializable getNucleus() {
39         return this;
40     }
41
42     @Override
43     public Object invoke(String methodName, Class[] argTypes, Object[] argValues,↩
           boolean local) {
44         Object retVal = null;
45         if(!local){
46             System.out.println("method will be invoked");
47         methodName = methodName + '_';
48         Method[] allMethods = getClass().getDeclaredMethods();
49         Method m = null;
50         try {
51             if (argTypes != null && argTypes.length > 0) {
52                 m = getClass().getDeclaredMethod(methodName, argTypes);
53                 m.setAccessible(true);
54                 retVal = m.invoke(this, argValues);
55             } else {
56                 m = getClass().getDeclaredMethod(methodName);
57                 m.setAccessible(true);
58                 retVal = m.invoke(this);
59             }
60         } catch (Exception e) {
61             retVal = e;
62         }
63         return retVal;
64         }
65         else{
66             System.out.println("method will not be invoked");
67         }
68         return retVal;
69     }
70
71     @Override
72     public void objectCtreated(JCSyncAbstractSharedObject object) {
73         this.shared_object = (SharedExchangerObject) object;
74     }
75     private void writeObject(ObjectOutputStream ostr) throws IOException {
```

```
76          Exchanger m = (Exchanger) this;
77          ostr.writeObject(m);
78      }
79  }
```

Listing 29: SharedExchanger.java full source code