COBrA DAPP Source code

Aldo D'Aquino

12/01/2018

1 DAPP

1.1 DAPP/author-server

Main.java

```
1  package com.aldodaquino.cobra.authorserver;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.math.BigInteger;
6  import java.net.InetSocketAddress;
7  import java.nio.channels.ServerSocketChannel;
8  import java.util.HashMap;
9  import java.util.Map;
10
11 import com.aldodaquino.cobra.connections.Status;
12 import com.aldodaquino.javautils.FileExchange;
13 import com.aldodaquino.cobra.connections.CobraHttpHelper;
14 import com.aldodaquino.cobra.main.CatalogManager;
15 import com.aldodaquino.cobra.main.ContentManager;
16 import com.aldodaquino.javautils.CliHelper;
17 import com.sun.net.httpserver.HttpExchange;
18 import com.sun.net.httpserver.HttpServer;
19
20 import org.web3j.crypto.Credentials;
21
22 /**
23  * The Main class of the author server. The author server remains always online to listen
   ↪  access request. The author can
24  * use its server to deploy his content on the blockchain and publish it on the catalog. This
   ↪  server serves content also
25  * when the author has closed his client.
26  * Includes two functions that handle the two urls /deploy (to deploy a content) and /access to
   ↪  access a content.
27  * @author Aldo D'Aquino.
28  * @version 1.0.
29  */
30 public class Main {
31
32     private static final int DEFAULT_PORT = 8080;
33     private static final String CONTENT_FILE_PATH = "author_content_files/";
34
35     /**
36      * Main method.
37      * @param args a String[] passed through command line.
38      * @throws IOException if it is not possible to create the server.
39      */
40     public static void main(String[] args) throws IOException {
41
42         // Parse cmd options
43         CliHelper cliHelper = new CliHelper();
44         cliHelper.addOption("h", "help", false, "Print this help message.");
45         cliHelper.addOption("k", "private-key", true,
46                 "Private key of your account (required).");
47         cliHelper.addOption("c", "catalog", true, "Catalog address (required).");
48         cliHelper.addOption("n", "hostname", true,
49                 "Name of this host, i.e. the IP address of this server, used to deploy content
                   ↪  (required).");
50         cliHelper.addOption("p", "port", true,
```

```
51                 "Port on which run the server. Default: " + DEFAULT_PORT + ".");
52         cliHelper.parse(args);
53 
54         if (cliHelper.isPresent("h")) {
55             System.out.println(cliHelper.getHelpMessage());
56             System.exit(0);
57         }
58 
59         Status status = new Status();
60 
61         status.privateKey = cliHelper.getValue("private-key");
62         if (status.privateKey == null || status.privateKey.length() == 0) {
63             System.err.println(cliHelper.getMissingOptionMessage("private-key"));
64             System.err.flush();
65             System.out.println(cliHelper.getHelpMessage());
66             System.out.flush();
67             System.exit(1);
68         }
69 
70         String catalogAddress = cliHelper.getValue("catalog");
71         if (catalogAddress == null || catalogAddress.length() == 0) {
72             System.err.println(cliHelper.getMissingOptionMessage("catalog"));
73             System.err.flush();
74             System.out.println(cliHelper.getHelpMessage());
75             System.out.flush();
76             System.exit(1);
77         }
78 
79         status.hostname = cliHelper.getValue("hostname");
80         if (status.hostname == null || status.hostname.length() == 0) {
81             System.err.println(cliHelper.getMissingOptionMessage("hostname"));
82             System.err.flush();
83             System.out.println(cliHelper.getHelpMessage());
84             System.out.flush();
85             System.exit(1);
86         }
87 
88         String portS = cliHelper.getValue("port");
89         status.port = portS != null && portS.length() != 0 ? Integer.parseInt(portS) :
           ↪  DEFAULT_PORT;
90 
91         // Init status
92         status.credentials = Credentials.create(status.privateKey);
93         status.catalogManager = new CatalogManager(status.credentials, catalogAddress);
94 
95         // Create server
96         HttpServer server = HttpServer.create(new InetSocketAddress(status.port), 0);
97         System.out.println("Server running on port " + status.port + ".\n");
98 
99         // set handlers
100        server.createContext("/deploy", CobraHttpHelper.newHandler(Main::deploy, status));
101        server.createContext("/access", CobraHttpHelper.newHandler(Main::access, status));
102 
103        // start server
104        server.setExecutor(null); // creates a default executor
105        server.start();
106    }
107 
108 
```

```
109     /**
110      * Handler for the /deploy url.
111      * @param request a POST request with JSON encoded data containing:
112      *                privateKey of the author;
113      *                name of the content;
114      *                genre of the content (can be null);
115      *                price of the content (if null is set to 0);
116      *                port on which is running the server socket that uploads the file.
117      */
118     private static void deploy(HttpExchange request, Status status) {
119         // get parameters
120         Map<String, String> parameters = CobraHttpHelper.parsePOST(request);
121 
122         if (!status.privateKey.equals(parameters.get("privateKey"))) {
123             CobraHttpHelper.sendResponse(request, "Only the author server owner can perform
                ↪  this action." +
124                     "You must login with the same private key of the server.", 403);
125             return;
126         }
127 
128         String name = parameters.get("name");
129         if (name == null) {
130             CobraHttpHelper.sendResponse(request, "ERROR: name not specified.", 400);
131             return;
132         }
133 
134         String genre = parameters.get("genre");
135 
136         String priceS = parameters.get("price");
137         BigInteger price;
138         try {
139             price = new BigInteger(priceS.length() != 0 ? priceS : "0");
140         } catch (NumberFormatException e) {
141             e.printStackTrace();
142             CobraHttpHelper.sendResponse(request, "ERROR: Invalid price.\n" + e.getMessage(),
                ↪  400);
143             return;
144         }
145 
146         String hostname = request.getRemoteAddress().getHostName();
147 
148         String portS = parameters.get("port");
149         int port;
150         try {
151             port = Integer.parseInt(portS);
152         } catch (NumberFormatException e) {
153             e.printStackTrace();
154             CobraHttpHelper.sendResponse(request, "ERROR: Invalid port number.\n" +
                ↪  e.getMessage(), 400);
155             return;
156         }
157 
158         String filename = parameters.get("filename");
159         if (filename == null) {
160             CobraHttpHelper.sendResponse(request, "ERROR: filename not specified.", 400);
161             return;
162         }
163 
164         // deploy the content
```

```
165         String address;
166         try {
167             ContentManager contentManager = new ContentManager(status.credentials,
168                     status.catalogManager.getAddress(), name, genre, price, status.hostname,
                        ↪  status.port);
169             address = contentManager.getAddress();
170         } catch (Exception e) {
171             e.printStackTrace();
172             CobraHttpHelper.sendResponse(request, e.getMessage(), 400);
173             return;
174         }
175 
176         // download the file
177         File file = new File(CONTENT_FILE_PATH + address + filename);
178         //noinspection ResultOfMethodCallIgnored
179         file.getParentFile().mkdirs();
180         FileExchange.receiveFile(file, hostname, port);
181 
182         // send the response
183         CobraHttpHelper.sendResponse(request, address);
184     }
185 
186     /**
187      * Handler for the /access url.
188      * @param request a POST request with JSON encoded data containing:
189      *                privateKey of the author
190      *                name of the content
191      *                genre of the content (can be null)
192      *                price of the content (if null is set to 0)
193      */
194     private static void access(HttpExchange request, Status status) {
195         // get parameters
196         Map<String, String> parameters = CobraHttpHelper.parsePOST(request);
197 
198         String address = parameters.get("address");
199         if (address == null) {
200             CobraHttpHelper.sendResponse(request, "ERROR: content address not specified.",
                    ↪  400);
201             return;
202         }
203 
204         String userPrivateKey = parameters.get("privateKey");
205         if (userPrivateKey == null) {
206             CobraHttpHelper.sendResponse(request, "ERROR: user private key not specified.",
                    ↪  400);
207             return;
208         }
209         Credentials credentials = Credentials.create(userPrivateKey);
210         String user = credentials.getAddress();
211 
212         if (!status.catalogManager.hasAccess(address, user)) {
213             CobraHttpHelper.sendResponse(request, "ERROR: you don't have access to this
                    ↪  content.", 400);
214             return;
215         }
216 
217         // open the socket for the file
218         ServerSocketChannel serverSocketChannel = FileExchange.openFileSocket();
219         if (serverSocketChannel == null) {
```

```
220             CobraHttpHelper.sendResponse(request, "ERROR: cannot open the server socket.",
                ↪  500);
221             return;
222         }
223
224         // pick the file
225         File[] files = new File(CONTENT_FILE_PATH).listFiles();
226         if (files == null) {
227             CobraHttpHelper.sendResponse(request, "ERROR: there is no file for this content.",
                ↪  500);
228             return;
229         }
230         File file = null;
231         for (File f : files)
232             if (f.isFile() && f.getName().contains(address)) {
233                 file = f;
234                 break;
235             }
236         if (file == null) {
237             CobraHttpHelper.sendResponse(request, "ERROR: there is no file for this content.",
                ↪  500);
238             return;
239         }
240         String filename = file.getName().replace(address, "");
241
242         int port = serverSocketChannel.socket().getLocalPort();
243         FileExchange.startFileSender(serverSocketChannel, file,
244                 () -> System.out.println("User " + user + " has received all the content " +
                    ↪  address + "."));
245
246         // consume the content
247         ContentManager contentManager = new ContentManager(credentials, address);
248         if (!contentManager.consumeContent()) {
249             CobraHttpHelper.sendResponse(request, "ERROR: cannot consume content.", 500);
250             return;
251         }
252
253         // communicate the port number and the filename
254         Map<String, String> response = new HashMap<>();
255         response.put("port", Integer.toString(port));
256         response.put("filename", filename);
257         CobraHttpHelper.sendResponse(request, CobraHttpHelper.jsonifyParameters(response));
258     }
259
260 }
```

1.2 DAPP/connections

API.java

```java
package com.aldodaquino.cobra.connections;

/**
 * Interface for the author server API. Specify the urls of the handlers.
 * Check Main class in {@link com.aldodaquino.cobra.authorserver}
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public interface API {

    String DEPLOY_API_PATH = "/deploy";
    String ACCESS_API_PATH = "/access";

}
```

CobraHttpHelper.java

```java
package com.aldodaquino.cobra.connections;

import com.aldodaquino.javautils.HttpHelper;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;

import java.util.function.BiConsumer;

/**
 * Contains method that help to make http request.
 * Works with JSON body for POST request and query-style GET parameters.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class CobraHttpHelper extends HttpHelper {

    private static class HttpRequestHandler implements HttpHandler {
        final BiConsumer<HttpExchange, Status> consumer;
        final Status status;
        HttpRequestHandler(BiConsumer<HttpExchange, Status> consumer, Status status) {
            this.consumer = consumer;
            this.status = status;
        }
        @Override
        public void handle(HttpExchange request) {
            consumer.accept(request, status);
        }
    }

    /**
     * Return new HttpHandler
     * @param consumer a function to be called when a new request arrive.
     *                 The consumer has to accept the HttpExchange request and the Status.
     * @param status to be passed to the consumer.
     * @return HttpHandler, the handler.
     */
    public static HttpHandler newHandler(BiConsumer<HttpExchange, Status> consumer, Status
    ↪  status) {
```

```
38          return new HttpRequestHandler(consumer, status);
39      }
40
41  }
```

Status.java

```
1   package com.aldodaquino.cobra.connections;
2
3   import com.aldodaquino.cobra.main.CatalogManager;
4   import org.web3j.crypto.Credentials;
5
6   /**
7    * Defines the status for the author-server.
8    * @author Aldo D'Aquino.
9    * @version 1.0.
10   */
11  public class Status {
12      public String privateKey;
13      public Credentials credentials;
14      public CatalogManager catalogManager;
15      public String hostname;
16      public int port;
17  }
```

1.3 DAPP/contracts

CatalogManager.java

```java
package com.aldodaquino.cobra.main;

import com.aldodaquino.cobra.contracts.CatalogContract;
import org.web3j.crypto.Credentials;
import org.web3j.protocol.core.DefaultBlockParameterName;
import org.web3j.tuples.generated.*;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.BiConsumer;

/**
 * An higher level Catalog Manager.
 * Contains methods that call the methods in the {@link CatalogContract} generated by Web3j and parse and aggregate the
 * result in a more comfortable representation.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class CatalogManager extends ContractManager {

    private final CatalogContract catalog;

    // event callbacks
    private final List<BiConsumer<String, String>> newContentAvailableBiConsumers = new ArrayList<>();
    private final List<Runnable> newContentAvailableRunnables = new ArrayList<>();
    private final Map<String, List<BiConsumer<String, String>>> accessGrantedMap = new HashMap<>();
    private final Map<String, List<Runnable>> becomesPremiumMap = new HashMap<>();
    private final List<BiConsumer<String, String>> feedbackAvailableBiConsumer = new ArrayList<>();
    private final Map<String, List<BiConsumer<String, String>>> paymentAvailableMap = new HashMap<>();
    private final List<Runnable> catalogClosedRunnables = new ArrayList<>();

    /*
     * CONSTRUCTORS
     */

    /**
     * Deploy and manage a new catalog contract.
     * @param credentials your account credentials.
     */
    public CatalogManager(Credentials credentials) {
        super(credentials);
        catalog = (CatalogContract) deploy(CatalogContract.class);
        listenForEvents();
    }

    /**
     * Load and manage an existent catalog contract.
     * @param credentials your account credentials.
```

```
52          * @param contractAddress the existent contract address on blockchain.
53          */
54         public CatalogManager(Credentials credentials, String contractAddress) {
55             super(credentials);
56             catalog = (CatalogContract) load(CatalogContract.class, contractAddress);
57             listenForEvents();
58         }
59
60         // auxiliary function
61         private void listenForEvents() {
62             catalog.newContentAvailableEventObservable(DefaultBlockParameterName.EARLIEST,
               ↪ DefaultBlockParameterName.LATEST)
63                     .subscribe(e -> {
64                         for (BiConsumer<String, String> biConsumer :
                           ↪ newContentAvailableBiConsumers)
65                             biConsumer.accept(Utils.bytes32ToString(e.name), e.addr);
66                         for (Runnable runnable : newContentAvailableRunnables)
67                             runnable.run();
68                     });
69
70             catalog.grantedAccessEventObservable(DefaultBlockParameterName.EARLIEST,
               ↪ DefaultBlockParameterName.LATEST)
71                     .subscribe(e -> {
72                         List<BiConsumer<String, String>> biConsumers =
                           ↪ accessGrantedMap.get(e.user);
73                         if (biConsumers != null) {
74                             for (BiConsumer<String, String> biConsumer : biConsumers)
75                                 biConsumer.accept(e.content, getTheName(e.content));
76                         }
77                     });
78
79             catalog.becomesPremiumEventObservable(DefaultBlockParameterName.EARLIEST,
               ↪ DefaultBlockParameterName.LATEST)
80                     .subscribe(e -> {
81                         List<Runnable> runnables = becomesPremiumMap.get(e.user);
82                         if (runnables != null)
83                             for (Runnable runnable : runnables)
84                                 runnable.run();
85                     });
86
87             catalog.feedbackAvailableEventObservable(DefaultBlockParameterName.EARLIEST,
               ↪ DefaultBlockParameterName.LATEST)
88                     .subscribe(e -> {
89                         if (e.user.equals(credentials.getAddress())) {
90                             String name = "";
91                             try {
92                                 name =
                                   ↪ Utils.bytes32ToString(catalog.getContentInfo(e.content).send().getValue1())
93                             } catch (Exception exception) {
94                                 exception.printStackTrace();
95                             }
96                             for (BiConsumer<String, String> biConsumer :
                               ↪ feedbackAvailableBiConsumer)
97                                 biConsumer.accept(e.content, name);
98                         }
99                     });
100
101            catalog.paymentAvailableEventObservable(DefaultBlockParameterName.EARLIEST,
               ↪ DefaultBlockParameterName.LATEST)
```

```
                .subscribe(e -> {
                    List<BiConsumer<String, String>> biConsumers =
                    ↪  paymentAvailableMap.get(e.content);
                    if (biConsumers != null) {
                        for (BiConsumer<String, String> biConsumer : biConsumers)
                            biConsumer.accept(e.content, getTheName(e.content));
                    }
                });

        catalog.catalogClosedEventObservable(DefaultBlockParameterName.EARLIEST,
        ↪  DefaultBlockParameterName.LATEST)
                .subscribe(e -> {
                    for (Runnable runnable : catalogClosedRunnables)
                        runnable.run();
                });
    }

    // auxiliary function
    private String getTheName(String contentAddress) {
        try {
            return
            ↪  Utils.bytes32ToString(catalog.getContentInfo(contentAddress).send().getValue1());
        } catch (Exception exception) {
            exception.printStackTrace();
            return "";
        }
    }

    /*
     * CATALOG CONTRACT SPECIFIC METHODS
     */

    /* Events */

    /**
     * Subscribe a callback for new content available events.
     * @param callback a BiConsumer of content name and address.
     */
    public void listenNewContentAvailable(BiConsumer<String, String> callback) {
        newContentAvailableBiConsumers.add(callback);
    }

    /**
     * Subscribe a callback for new content available events.
     * @param callback a Runnable.
     */
    public void listenNewContentAvailable(Runnable callback) {
        newContentAvailableRunnables.add(callback);
    }

    /**
     * Subscribe a callback for access granted events for this user.
     * @param callback a BiConsumer of content address and content name.
     */
    public void listenAccessGranted(BiConsumer<String, String> callback) {
        listenAccessGranted(credentials.getAddress(), callback);
    }

    /**
```

```
158         * Subscribe a callback for access granted events for the specified user.
159         * @param user the user for which be registered.
160         * @param callback a BiConsumer of content address and content name.
161         */
162        public void listenAccessGranted(String user, BiConsumer<String, String> callback) {
163            accessGrantedMap.putIfAbsent(user, new ArrayList<>());
164            accessGrantedMap.get(user).add(callback);
165        }
166
167        /**
168         * Subscribe a callback for becomes premium events for this user.
169         * @param callback a Runnable.
170         */
171        public void listenBecomesPremium(Runnable callback) {
172            listenBecomesPremium(credentials.getAddress(), callback);
173        }
174
175        /**
176         * Subscribe a callback for becomes premium events for this user.
177         * @param user the user for which be registered.
178         * @param callback a Runnable.
179         */
180        public void listenBecomesPremium(String user, Runnable callback) {
181            becomesPremiumMap.putIfAbsent(user, new ArrayList<>());
182            becomesPremiumMap.get(user).add(callback);
183        }
184
185        /**
186         * Subscribe a callback for feedback available events for this user.
187         * @param callback a BiConsumer of content address and content name.
188         */
189        public void listenFeedbackAvailable(BiConsumer<String, String> callback) {
190            feedbackAvailableBiConsumer.add(callback);
191        }
192
193        /**
194         * Subscribe a callback for payment available events for this user.
195         * @param content the content of which listen to.
196         * @param callback a BiConsumer of content address and content name.
197         */
198        public void listenPaymentAvailable(String content, BiConsumer<String, String> callback) {
199            paymentAvailableMap.putIfAbsent(content, new ArrayList<>());
200            paymentAvailableMap.get(content).add(callback);
201        }
202
203        /**
204         * Subscribe a callback for payment available events for this user.
205         * @param callback a Runnable.
206         */
207        public void listenCatalogClosed(Runnable callback) {
208            catalogClosedRunnables.add(callback);
209        }
210
211        /* Catalog interaction methods */
212
213        /**
214         * Check if the user has access to a content.
215         * @param address the content address.
216         * @return boolean if has access, false otherwise.
```

```
217         */
218        public boolean hasAccess(String address) {
219            return hasAccess(address, credentials.getAddress());
220        }
221
222        /**
223         * Check if the specified user has access to a content.
224         * @param address the content address.
225         * @param user the user address.
226         * @return boolean if has access, false otherwise.
227         */
228        public boolean hasAccess(String address, String user) {
229            try {
230                return catalog.hasAccess(user, address).send();
231            } catch (Exception e) {
232                e.printStackTrace();
233                return false;
234            }
235        }
236
237        /**
238         * Buy a content.
239         * @param address the content address.
240         * @param price the content price.
241         * @return a boolean representing the operation outcome.
242         */
243        public boolean buyContent(String address, BigInteger price) {
244            try {
245                return catalog.getContent(address, price).send().isStatusOK();
246            } catch (Exception e) {
247                e.printStackTrace();
248                return false;
249            }
250        }
251
252        /**
253         * Gift a content to another user.
254         * @param address the content address.
255         * @param user the user address.
256         * @param price the content price.
257         * @return a boolean representing the operation outcome.
258         */
259        public boolean giftContent(String address, String user, BigInteger price) {
260            try {
261                return catalog.giftContent(address, user, price).send().isStatusOK();
262            } catch (Exception e) {
263                e.printStackTrace();
264                return false;
265            }
266        }
267
268        /**
269         * Buy a premium subscription.
270         * @return a boolean representing the operation outcome.
271         */
272        public boolean buyPremium() {
273            try {
274                BigInteger premiumCost = catalog.premiumCost().send();
275                return catalog.buyPremium(premiumCost).send().isStatusOK();
```

```
276             } catch (Exception e) {
277                 e.printStackTrace();
278                 return false;
279             }
280         }
281 
282         /**
283          * Gift a premium subscription to another user.
284          * @param user the user address.
285          * @return a boolean representing the operation outcome.
286          */
287         public boolean giftPremium(String user) {
288             try {
289                 BigInteger contentCost = catalog.premiumCost().send();
290                 return catalog.giftPremium(user, contentCost).send().isStatusOK();
291             } catch (Exception e) {
292                 e.printStackTrace();
293                 return false;
294             }
295         }
296 
297         /**
298          * Return true if the user has an active premium subscription.
299          * @return a boolean representing the operation outcome.
300          */
301         public boolean isPremium() {
302             try {
303                 return catalog.isPremium(credentials.getAddress()).send();
304             } catch (Exception e) {
305                 e.printStackTrace();
306                 return false;
307             }
308         }
309 
310         /**
311          * Set rating for a content.
312          * @param content the content address.
313          * @param enjoy the rating for the enjoy category.
314          * @param valueForMoney the rating for the value for money category.
315          * @param contentMeaning the rating for the content meaning category.
316          */
317         public void vote(String content, int enjoy, int valueForMoney, int contentMeaning) {
318             try {
319                 byte[] enjoyS = catalog.ratingCategories(new BigInteger("0")).send();
320                 byte[] valueForMoneyS = catalog.ratingCategories(new BigInteger("1")).send();
321                 byte[] contentMeaningS = catalog.ratingCategories(new BigInteger("2")).send();
322                 catalog.leaveFeedback(content, enjoyS, new BigInteger(Integer.toString(enjoy)));
323                 catalog.leaveFeedback(content, valueForMoneyS, new
                    ↪  BigInteger(Integer.toString(valueForMoney)));
324                 catalog.leaveFeedback(content, contentMeaningS, new
                    ↪  BigInteger(Integer.toString(contentMeaning)));
325             } catch (Exception e) {
326                 e.printStackTrace();
327             }
328         }
329 
330         /* Getters for lists, statistics and charts */
331 
332         /**
```

```
333         * Returns all the info and ratings of a content.
334         * @param address the content address.
335         * @return a list of Content objects.
336         */
337        public Content getContentInfo(String address) {
338            try {
339                Tuple5<byte[], String, byte[], BigInteger, BigInteger> info =
                    ↪ catalog.getContentInfo(address).send();
340                Tuple4<BigInteger, BigInteger, BigInteger, BigInteger> ratings =
                    ↪ catalog.getContentRatings(address).send();
341
342                return new Content(address, info.getValue1(), info.getValue2(), info.getValue3(),
                    ↪ info.getValue4(),
343                        info.getValue5(), ratings.getValue1(), ratings.getValue2(),
                            ↪ ratings.getValue3(), ratings.getValue4());
344            } catch (Exception e) {
345                e.printStackTrace();
346                return null;
347            }
348        }
349
350        /**
351         * Returns a list of all contents in the Catalog.
352         * @return a list of Content objects.
353         */
354        public List<Content> getContentList() {
355            try {
356                Tuple2<List<byte[]>, List<String>> statistics = catalog.getContentList().send();
357                List<byte[]> names = statistics.getValue1();
358                List<String> addresses = statistics.getValue2();
359
360                List<Content> contents = new ArrayList<>();
361                for (int i = 0; i < names.size(); i++)
362                    contents.add(new Content(addresses.get(i), names.get(i)));
363
364                return contents;
365            } catch (Exception e) {
366                e.printStackTrace();
367                return null;
368            }
369        }
370
371        /**
372         * Returns a list of all contents in the Catalog and its views.
373         * @return a list of Content objects.
374         */
375        public List<Content> getContentListWithViews() {
376            try {
377                Tuple3<List<byte[]>, List<String>, List<BigInteger>> statistics =
                    ↪ catalog.getStatistics().send();
378                List<byte[]> names = statistics.getValue1();
379                List<String> addresses = statistics.getValue2();
380                List<BigInteger> views = statistics.getValue3();
381
382                List<Content> contents = new ArrayList<>();
383                for (int i = 0; i < names.size(); i++)
384                    contents.add(new Content(addresses.get(i), names.get(i), views.get(i)));
385
386                return contents;
```

```
387             } catch (Exception e) {
388                 e.printStackTrace();
389                 return null;
390             }
391         }
392 
393         /**
394          * Return the list of all the content of a given author.
395          * @param author the authors address.
396          * @return a list of Content objects.
397          */
398         public List<Content> getAuthorContents(String author) {
399             ContentList contentList = new ContentList();
400             return contentList.getFilteredContentList(contentList.authors, author);
401         }
402 
403         /**
404          * Return the n latest releases.
405          * @param n the number of item that you want in the list.
406          * @return List of Content objects with the latest n contents.
407          */
408         public List<Content> getNewContentList(int n) {
409             try {
410                 // get the list
411                 Tuple2<List<byte[]>, List<String>> res =
412                         catalog.getNewContentList(new BigInteger(Integer.toString(n))).send();
413                 List<byte[]> names = res.getValue1();
414                 List<String> addresses = res.getValue2();
415 
416                 // parse the list in a String matrix
417                 List<Content> contents = new ArrayList<>();
418                 for (int i = 0; i < names.size(); i++)
419                     contents.add(new Content(addresses.get(i), names.get(i)));
420                 return contents;
421             } catch (Exception e) {
422                 e.printStackTrace();
423                 return null;
424             }
425         }
426 
427         /**
428          * Return the latest release.
429          * @return String[] where the first element is the name of the content and the second is
    ↪  the address.
430          */
431         public Content getLatest() {
432             List<Content> contents = getNewContentList(1);
433             if (contents == null || contents.size() == 0) return null;
434             return contents.get(0);
435         }
436 
437         /**
438          * Return the latest release for a genre.
439          * @param genre the chosen genre.
440          * @return String[] where the first element is the name of the content and the second is
    ↪  the address.
441          */
442         public Content getLatestByGenre(String genre) {
443             try {
```

```
444             Tuple2<byte[], String> res =
                ↪  catalog.getLatestByGenre(Utils.stringToBytes32(genre)).send();
445             return new Content(res.getValue2(), res.getValue1());
446         } catch (Exception e) {
447             e.printStackTrace();
448             return null;
449         }
450     }
451
452     /**
453      * Return the latest release of an author.
454      * @param author the author address.
455      * @return String[] where the first element is the name of the content and the second is
    ↪  the address.
456      */
457     public Content getLatestByAuthor(String author) {
458         try {
459             Tuple2<byte[], String> res = catalog.getLatestByAuthor(author).send();
460             return new Content(res.getValue2(), res.getValue1());
461         } catch (Exception e) {
462             e.printStackTrace();
463             return null;
464         }
465     }
466
467     /**
468      * Return the most popular content.
469      * @return Content.
470      */
471     public Content getMostPopular() {
472         try {
473             Tuple2<byte[], String> res = catalog.getMostPopular().send();
474             return new Content(res.getValue2(), res.getValue1());
475         } catch (Exception e) {
476             e.printStackTrace();
477             return null;
478         }
479     }
480
481     /**
482      * Return the most popular content for a genre.
483      * @param genre the chosen genre.
484      * @return String[] where the first element is the name of the content and the second is
    ↪  the address.
485      */
486     public Content getMostPopularByGenre(String genre) {
487         try {
488             Tuple2<byte[], String> res =
                ↪  catalog.getMostPopularByGenre(Utils.stringToBytes32(genre)).send();
489             return new Content(res.getValue2(), res.getValue1());
490         } catch (Exception e) {
491             e.printStackTrace();
492             return null;
493         }
494     }
495
496     /**
497      * Return the most popular content of an author.
498      * @param author the author address.
```

```
499         * @return String[] where the first element is the name of the content and the second is
    ↪   the address.
500         */
501         public Content getMostPopularByAuthor(String author) {
502             try {
503                 Tuple2<byte[], String> res = catalog.getMostPopularByAuthor(author).send();
504                 return new Content(res.getValue2(), res.getValue1());
505             } catch (Exception e) {
506                 e.printStackTrace();
507                 return null;
508             }
509         }
510 
511         /**
512          * Return the highest rated content.
513          * @param category the category name for which you want to know the rating.
514          * @return Content.
515          */
516         public Content getMostRated(String category) {
517             try {
518                 Tuple2<byte[], String> res =
                    ↪  catalog.getMostRated(Utils.stringToBytes32(category)).send();
519                 return new Content(res.getValue2(), res.getValue1());
520             } catch (Exception e) {
521                 e.printStackTrace();
522                 return null;
523             }
524         }
525 
526         /**
527          * Return the highest rated content for a genre.
528          * @param genre the chosen genre.
529          * @param category the category name for which you want to know the rating.
530          * @return Content.
531          */
532         public Content getMostRatedByGenre(String genre, String category) {
533             try {
534                 Tuple2<byte[], String> res =
                    ↪  catalog.getMostRatedByGenre(Utils.stringToBytes32(genre),
535                         Utils.stringToBytes32(category)).send();
536                 return new Content(res.getValue2(), res.getValue1());
537             } catch (Exception e) {
538                 e.printStackTrace();
539                 return null;
540             }
541         }
542 
543         /**
544          * Return the highest rated content of an author.
545          * @param author the author address.
546          * @param category the category name for which you want to know the rating.
547          * @return Content.
548          */
549         public Content getMostRatedByAuthor(String author, String category) {
550             try {
551                 Tuple2<byte[], String> res = catalog.getMostRatedByAuthor(author,
                    ↪  Utils.stringToBytes32(category)).send();
552                 return new Content(res.getValue2(), res.getValue1());
553             } catch (Exception e) {
```

```
            e.printStackTrace();
            return null;
        }
    }

    /* Authors method */

    /**
     * Collect the payout for a content. Can be called only on content of the current user.
     * @param address the address of the content.
     * @return a BigInteger with the withdrawn amount, 0 otherwise (for example if the
    ↪  specified content is not owned by
     * the user that have done the login).
     */
    public BigInteger withdraw(String address) {
        try {
            BigInteger amount = catalog.payoutAvailable(address).send();
            if (!amount.equals(BigInteger.ZERO))
                catalog.collectPayout(address).send();
            return amount;
        } catch (Exception e) {
            e.printStackTrace();
            return BigInteger.ZERO;
        }
    }


    /* Auxiliary class */
    private class ContentList {

        List<String> addresses;
        List<byte[]> names;
        List<String> authors;
        List<byte[]> genres;
        List<BigInteger> prices;
        List<BigInteger> views;
        List<BigInteger> averageRatings;
        List<BigInteger> enjoyRatings;
        List<BigInteger> priceFairnessRatings;
        List<BigInteger> contentMeaningRatings;

        ContentList() {
            try {
                // Query the CatalogContract for the list
                Tuple6<List<String>, List<byte[]>, List<String>, List<byte[]>,
                ↪  List<BigInteger>, List<BigInteger>>
                        fullContentList = catalog.getFullContentList().send();
                Tuple5<List<String>, List<BigInteger>, List<BigInteger>, List<BigInteger>,
                ↪  List<BigInteger>>
                        ratingsList = catalog.getRatingsList().send();

                // Parse parameters
                addresses = fullContentList.getValue1();
                names = fullContentList.getValue2();
                authors = fullContentList.getValue3();
                genres = fullContentList.getValue4();
                prices = fullContentList.getValue5();
                views = fullContentList.getValue6();
                averageRatings = ratingsList.getValue2();
```

```
                enjoyRatings = ratingsList.getValue3();
                priceFairnessRatings = ratingsList.getValue4();
                contentMeaningRatings = ratingsList.getValue5();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        /**
         * Returns a list of all contents that has the parameter where equals to value.
         * The list is not filtered if where is null.
         * @param filterBy a list of this class that can be addresses, names, authors, genres,
    ↪  prices or views.
         * @param filterValue the value that filterBy must have.
         * @return a list of Content objects.
         */
        <T> List<Content> getFilteredContentList(List<T> filterBy, T filterValue) {
            // Build an usable list
            List<Content> contentList = new ArrayList<>();
            for (int i = 0; i < addresses.size(); i++)
                // if the where list is null do not filter
                if (filterBy == null || filterBy.get(i).equals(filterValue))
                    contentList.add(new Content(
                            addresses.get(i),
                            names.get(i),
                            authors.get(i),
                            genres.get(i),
                            prices.get(i),
                            views.get(i),
                            averageRatings.get(i),
                            enjoyRatings.get(i),
                            priceFairnessRatings.get(i),
                            contentMeaningRatings.get(i)
                            ));
            return contentList;
        }
    }

}
```

Content.java

```
package com.aldodaquino.cobra.main;

import java.math.BigInteger;

/**
 * Defines a content object.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class Content {

    public final String address;        // required
    public final String name;           // required
    public final String author;         // can be null
    public final String genre;          // can be null
    public final BigInteger price;      // can be null
```

```
    public final BigInteger views;      // can be null
    public final int averageRating;     // can be -1
    public final int enjoy;             // can be -1
    public final int priceFairness;     // can be -1
    public final int contentMeaning;    // can be -1

    Content(String address, byte[] name) {
        this.address = address;
        this.name = Utils.bytes32ToString(name);

        this.views = null;
        this.author = null;
        this.genre = null;
        this.price = null;
        this.averageRating = -1;
        this.enjoy = -1;
        this.priceFairness = -1;
        this.contentMeaning = -1;
    }

    Content(String address, byte[] name, BigInteger views) {
        this.address = address;
        this.name = Utils.bytes32ToString(name);
        this.views = views;

        this.author = null;
        this.genre = null;
        this.price = null;
        this.averageRating = -1;
        this.enjoy = -1;
        this.priceFairness = -1;
        this.contentMeaning = -1;
    }

    Content(String address, byte[] name, String author, byte[] genre, BigInteger price,
    ↪  BigInteger views,
            BigInteger averageRating, BigInteger enjoy, BigInteger priceFairness, BigInteger
            ↪  contentMeaning) {
        this.address = address;
        this.name = Utils.bytes32ToString(name);
        this.author = author;
        this.genre = Utils.bytes32ToString(genre);
        this.price = price;
        this.views = views;
        this.averageRating = averageRating.intValue();
        this.enjoy = enjoy.intValue();
        this.priceFairness = priceFairness.intValue();
        this.contentMeaning = contentMeaning.intValue();
    }

}
```

ContentManager.java

```
package com.aldodaquino.cobra.main;

import com.aldodaquino.cobra.contracts.DAPPContentManagementContract;
import org.web3j.crypto.Credentials;
```

```java
import org.web3j.protocol.core.DefaultBlockParameterName;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

/**
 * An higher level Content Manager.
 * Contains methods that call the methods in the {@link DAPPContentManagementContract} generated by Web3j and parse and
 * aggregate the result in a more comfortable representation.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class ContentManager extends ContractManager {

    private final DAPPContentManagementContract content;

    private final List<Runnable> contentPublishedRunnables = new ArrayList<>();

    /**
     * Deploy and manage a new content manager contract.
     * @param credentials your account credentials.
     * @param catalogAddress the address of the catalog.
     * @param name the name you want to assign to this content.
     * @param genre the genre you want to assign to this content.
     * @param price the price you want to assign to this content.
     * @param hostname the hostname of the author server.
     * @param port the port on which is running the author server.
     * @throws Exception if there is errors when deploying the Content contract.
     */
    public ContentManager(Credentials credentials, String catalogAddress, String name, String genre, BigInteger price,
                          String hostname, int port)
            throws Exception {
        super(credentials);
        content = (DAPPContentManagementContract) deploy(DAPPContentManagementContract.class);
        content.setName(Utils.stringToBytes32(name)).send();
        content.setGenre(Utils.stringToBytes32(genre)).send();
        content.setPrice(price).send();
        content.setHostname(Utils.stringToBytes32(hostname)).send();
        content.setPort(new BigInteger(Integer.toString(port))).send();
        content.publish(catalogAddress).send();

        content.contentPublishedEventObservable(DefaultBlockParameterName.EARLIEST, DefaultBlockParameterName.LATEST)
                .subscribe(e -> {
                    for (Runnable runnable : contentPublishedRunnables)
                        runnable.run();
                });
    }

    /**
     * Load and manage an existent content manager contract.
     * @param credentials your account credentials.
     * @param contractAddress the existent contract address on blockchain.
     */
    public ContentManager(Credentials credentials, String contractAddress) {
        super(credentials);
```

```
61          content = (DAPPContentManagementContract) load(DAPPContentManagementContract.class,
                ↪ contractAddress);
62      }
63
64      /**
65       * Subscribe a callback for content published events.
66       * @param callback a Runnable.
67       */
68      public void listenContentPublished(Runnable callback) {
69          contentPublishedRunnables.add(callback);
70      }
71
72      /**
73       * Consume a bought content.
74       * Note that the content is consumed by the user that owns the credentials passed to the
    ↪ constructor.
75       * @return a boolean representing the operation outcome.
76       */
77      public boolean consumeContent() {
78          try {
79              return content.consumeContent().send().isStatusOK();
80          } catch (Exception e) {
81              e.printStackTrace();
82              return false;
83          }
84      }
85
86      /**
87       * Returns the hostname of the author-server.
88       * @return String the hostname.
89       */
90      public String getHostname() {
91          try {
92              return Utils.bytes32ToString(content.hostname().send());
93          } catch (Exception e) {
94              e.printStackTrace();
95              return null;
96          }
97      }
98
99      /**
100      * Returns the port number of the author-server.
101      * @return the int number of the port.
102      */
103     public int getPort() {
104         try {
105             return content.port().send().intValue();
106         } catch (Exception e) {
107             e.printStackTrace();
108             return 0;
109         }
110     }
111
112 }
```

ContractManager.java

```
1  package com.aldodaquino.cobra.main;
2
```

```
3  import org.web3j.crypto.Credentials;
4  import org.web3j.protocol.Web3j;
5  import org.web3j.protocol.core.RemoteCall;
6  import org.web3j.protocol.http.HttpService;
7  import org.web3j.tx.Contract;
8
9  import java.lang.reflect.InvocationTargetException;
10 import java.lang.reflect.Method;
11 import java.math.BigInteger;
12
13 /**
14  * An higher level Contract Manager.
15  * Contains basic methods of all the contracts, such as the deploy, load and suicide function,
   ↪  but also the get owner
16  * and get address functions.
17  * @author Aldo D'Aquino.
18  * @version 1.0.
19  */
20 class ContractManager {
21
22     private Web3j web3;
23     private BigInteger gasPrice;
24     private BigInteger gasLimit;
25     Credentials credentials;
26
27     private Contract contract;
28     private Class<? extends Contract> contractClass;
29     private String owner;
30
31     /*
32      * CONSTRUCTORS
33      */
34
35     /**
36      * Save credentials, connect to web3 and save the gas information.
37      * @param credentials your account credentials.
38      */
39     ContractManager(Credentials credentials) {
40         if (credentials == null) throw new IllegalArgumentException("Credentials cannot be
           ↪  null.");
41         // save credentials
42         this.credentials = credentials;
43         // connect to web3
44         web3 = Web3j.build(new HttpService());    // defaults to http://localhost:8545/
45         // web3 = Web3j.build(new
           ↪  HttpService("https://ropsten.infura.io/v3/12a335f54b784b988c4d9ba9d983cd65"));
46         // get gas information
47         gasPrice = Utils.getGasPrice(web3);
48         gasLimit = Utils.getGasLimit(web3);
49         Utils.getBalance(web3, credentials.getAddress());
50     }
51
52     /**
53      * Deploy a new contract of class contractClass and return the contractClass instance.
54      * @param contractClass the class of the contract that you want deploy.
55      * @return the deployed contract as contractClass instance.
56      */
57     Contract deploy(Class<? extends Contract> contractClass) {
58         this.contractClass = contractClass;
```

```
59          try {
60              Object[] params = {web3, credentials, gasPrice, gasLimit};
61              Class[] paramsTypes = {Web3j.class, Credentials.class, BigInteger.class,
                ↪  BigInteger.class};
62              Method deploy = contractClass.getMethod("deploy", paramsTypes);
63              contract = (Contract) ((RemoteCall) deploy.invoke(null, params)).send();
64              owner = credentials.getAddress();   // who deploy the contract is the owner
65          } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e)
            ↪  {
66              System.err.println("ERROR while deploying " + contractClass + ".");
67              e.printStackTrace();
68          } catch (Exception e) {
69              System.err.println("Got Web3j error while deploying " + contractClass + ".");
70              // I want to end the program if the exception occur,
71              // but I don't want to have to manage this exception that should not be thrown
72              throw new RuntimeException(e);
73          }
74          return contract;
75      }
76
77      /**
78       * Load an existent contract of class contractClass and return the contractClass instance.
79       * @param contractClass the class of the contract that you want to load.
80       * @param contractAddress the address of the contract.
81       * @return the loaded contract as contractClass instance.
82       */
83      Contract load(Class<? extends Contract> contractClass, String contractAddress) {
84          this.contractClass = contractClass;
85          Object[] params = {contractAddress, web3, credentials, gasPrice, gasLimit};
86          Class[] paramsTypes = {String.class, Web3j.class, Credentials.class, BigInteger.class,
            ↪  BigInteger.class};
87          try {
88              // Load contract
89              Method load = contractClass.getMethod("load", paramsTypes);
90              contract = (Contract) load.invoke(null, params);
91              // Get the contract owner
92              Method owner = contractClass.getMethod("owner");
93              try {
94                  this.owner = (String) ((RemoteCall) owner.invoke(contract)).send();
95              } catch (NullPointerException e) {
96                  System.err.println("ERROR while loading " + contractClass +
97                          ". Contract " + contractAddress + " may not exists.");
98                  e.printStackTrace();
99              }
100         } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e)
            ↪  {
101             System.err.println("ERROR while loading " + contractClass + ".");
102             e.printStackTrace();
103         } catch (Exception e) {
104             System.err.println("Got Web3j error while trying to get the contract owner.");
105             e.printStackTrace();
106         }
107         return contract;
108     }
109
110     /**
111      * Returns the contract address.
112      * @return a string containing the contract address.
113      */
```

```
114     public String getAddress() {
115         return contract.getContractAddress();
116     }
117 
118     /**
119      * Returns the contract owner.
120      * @return a string containing the contract owner.
121      */
122     public String getOwner() {
123         return owner;
124     }
125 
126     /**
127      * Call the suicide function on the contract.
128      * @return true if the contract suicide has been committed, false in case of errors.
129      */
130     public boolean suicide() {
131         try {
132             Method suicide = contractClass.getMethod("_suicide");
133             contract = (Contract) ((RemoteCall) suicide.invoke(null)).send();
134             return true;
135         } catch (Exception e) {
136             System.err.println("Got Web3j error while try to get the contract owner.");
137             e.printStackTrace();
138             return false;
139         }
140     }
141 
142 }
```

Utils.java

```
1  package com.aldodaquino.cobra.main;
2 
3  import org.web3j.protocol.Web3j;
4  import org.web3j.protocol.core.DefaultBlockParameter;
5  import org.web3j.protocol.core.DefaultBlockParameterName;
6  import org.web3j.protocol.core.Ethereum;
7  import org.web3j.protocol.core.methods.response.EthBlock;
8 
9  import java.io.IOException;
10 import java.math.BigInteger;
11 import java.nio.charset.StandardCharsets;
12 import java.util.Arrays;
13 
14 /**
15  * Some utilities for the blockchain.
16  * Contains method to get gas information and to convert bytes32 to Strings and vice-versa.
17  * @author Aldo D'Aquino.
18  * @version 1.0.
19  */
20 class Utils {
21 
22     private static final String BLOCK_GAS_LIMIT = "5000000";
23 
24     /**
25      * Return the average gas price.
26      * @param web3 a web3j instance.
```

```
27          * @return a BigInteger of the gas price, 0 in case of error.
28          */
29         static BigInteger getGasPrice(Web3j web3) {
30             BigInteger gasPrice;
31             try {
32                 gasPrice = web3.ethGasPrice().send().getGasPrice();
33             } catch (IOException e) {
34                 System.err.println("Cannot get the gas limit.");
35                 e.printStackTrace();
36                 gasPrice = BigInteger.ZERO;
37             }
38             System.out.println("Gas price: " + gasPrice);
39             return gasPrice;
40         }
41 
42         /**
43          * Return the maximum gas limit that we can use in a transaction.
44          * @param web3 a web3j instance.
45          * @return a BigInteger of the gas limit, 0 in case of error.
46          */
47         static BigInteger getGasLimit(Web3j web3) {
48             BigInteger gasLimit;
49             try {
50                 EthBlock.Block block =
51                         web3.ethGetBlockByNumber(DefaultBlockParameterName.LATEST,
                            ↪  true).send().getBlock();
52                 if (block != null) {
53                     System.out.println("Latest block number: " + block.getNumber());
54                     gasLimit = block.getGasLimit();
55                 }
56                 else gasLimit = new BigInteger(BLOCK_GAS_LIMIT);
57             } catch (IOException e) {
58                 System.err.println("Cannot get the gas limit.");
59                 e.printStackTrace();
60                 gasLimit = BigInteger.ZERO;
61             }
62             System.out.println("Block gas limit: " + gasLimit);
63             return gasLimit;
64         }
65 
66         static BigInteger getBalance(Web3j web3, String address) {
67             BigInteger balance;
68             try {
69                  balance = web3.ethGetBalance(address, DefaultBlockParameterName.LATEST).send()
70                          .getBalance();
71             } catch (IOException e) {
72                 System.err.println("Cannot get the account balance.");
73                 e.printStackTrace();
74                 balance = BigInteger.ZERO;
75             }
76             System.out.println("Account balance: " + balance);
77             return balance;
78         }
79 
80         /**
81          * Convert a bytes32 in a String.
82          * @param bytes32 the byte[].
83          * @return the String.
84          */
```

```
85      static String bytes32ToString(byte[] bytes32) {
86          int i = bytes32.length - 1;
87          while (i >= 0 && bytes32[i] == 0) i--;
88          bytes32 = Arrays.copyOf(bytes32, i + 1);
89          return new String(bytes32, StandardCharsets.UTF_8);
90      }
91
92      /**
93       * Convert a String in a bytes32.
94       * @param string the String.
95       * @return the byte[].
96       */
97      static byte[] stringToBytes32(String string) {
98          byte[] byte32 = new byte[32];
99          if (string != null) {
100             byte[] bytes = string.getBytes();
101             System.arraycopy(bytes, 0, byte32, 0, bytes.length);
102         }
103         return byte32;
104     }
105
106 }
```

1.4 DAPP/gui

Main.java

```
1  package com.aldodaquino.cobra.gui;
2
3  import com.aldodaquino.cobra.gui.constants.Strings;
4  import com.aldodaquino.cobra.gui.panels.AuthorPanel;
5  import com.aldodaquino.cobra.gui.panels.CustomerPanel;
6  import com.aldodaquino.cobra.gui.panels.StarterPanel;
7
8  import javax.swing.*;
9
10 /**
11  * The GUI Main. Starts the GUI with the {@link StarterPanel}.
12  * @author Aldo D'Aquino.
13  * @version 1.0.
14  */
15 public class Main {
16
17     private static JFrame window;
18
19     /**
20      * Main method.
21      * @param args an empty array, no parameters are required.
22      */
23     public static void main(String[] args) {
24
25         // Create the starter panel
26         JPanel starterPanel = new StarterPanel(Main::showMainPanel);
27
28         // Create the window
29         window = Utils.newWindow(Strings.appName, starterPanel, true);
30         window.setMinimumSize(starterPanel.getMinimumSize());
31
32     }
33
34     private static void setContent(JPanel replacement) {
35         window.setContentPane(replacement);
36         window.revalidate();
37         window.repaint();
38         window.pack();
39         window.setLocationRelativeTo(null);
40         window.setMinimumSize(replacement.getMinimumSize());
41     }
42
43     private static void showMainPanel(Status status) {
44         JPanel newPanel;
45
46         switch (status.getRole()) {
47             case (Status.ROLE_CUSTOMER):
48                 newPanel = new CustomerPanel(status);
49                 break;
50             case (Status.ROLE_AUTHOR):
51                 newPanel = new AuthorPanel(status);
52                 break;
53             default:
54                 throw new IllegalArgumentException("Status role property has an invalid value.
                    ↪  " +
55                         "Should be one of the Status.ROLE_X constants.");
```

```
        }
        setContent(newPanel);
    }
}
```

Status.java

```
package com.aldodaquino.cobra.gui;

import com.aldodaquino.cobra.main.CatalogManager;
import org.web3j.crypto.Credentials;

import javax.naming.OperationNotSupportedException;

/**
 * The Status class. A Status object is generated in the {@link com.aldodaquino.cobra.gui.panels.StarterPanel}.
 * Contains all the information about the user and the catalog and is passed through the classes.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class Status {

    public static final int ROLE_CUSTOMER = 0;
    public static final int ROLE_AUTHOR = 1;

    private String privateKey;
    public Credentials credentials;
    private CatalogManager catalogManager;
    private int role;

    /**
     * Given a private key generates and stores the credentials for this user.
     * @param privateKey the user's private key.
     * @throws OperationNotSupportedException if the user is already logged in.
     */
    public void login (String privateKey) throws OperationNotSupportedException {
        this.privateKey = privateKey;
        if (privateKey == null || privateKey.length() == 0) throw new IllegalArgumentException("Empty private key");
        if (credentials != null)
            throw new OperationNotSupportedException("Already logged in as " + credentials.getAddress() + ".");
        credentials = Credentials.create(privateKey);
    }

    /**
     * Connects to an existent catalog and stores it.
     * @param catalogAddress the catalog address.
     * @throws OperationNotSupportedException if the user is not logged in.
     */
    public void connectCatalog(String catalogAddress) throws OperationNotSupportedException {
        if (catalogAddress == null || catalogAddress.length() == 0)
            throw new IllegalArgumentException("Empty catalog address");
        if (credentials == null)
            throw new OperationNotSupportedException("You must be logged in to connect to a catalog.");
```

```
47          catalogManager = new CatalogManager(credentials, catalogAddress);
48      }
49
50      /**
51       * Disconnects from the catalog.
52       */
53      public void disconnectCatalog() {
54          catalogManager = null;
55      }
56
57      /**
58       * Deploys a new catalog with the user credentials.
59       * @throws OperationNotSupportedException if the user is not logged in.
60       */
61      public void deployCatalog() throws OperationNotSupportedException {
62          if (credentials == null)
63              throw new OperationNotSupportedException("You must be logged in to deploy a new
                 ↪  catalog.");
64          catalogManager = new CatalogManager(credentials);
65      }
66
67      /**
68       * Returns the user address.
69       * @return a String with the user address.
70       * @throws OperationNotSupportedException if the user is not logged in.
71       */
72      public String getUserAddress() throws OperationNotSupportedException {
73          if (credentials == null)
74              throw new OperationNotSupportedException("You must be logged in.");
75          return credentials.getAddress();
76      }
77
78      /**
79       * Returns true if the user that deployed the catalog is the current user, false otherwise.
80       * @return true if the current user is the catalog owner, false otherwise.
81       * @throws OperationNotSupportedException if the user is not connected to a catalog.
82       */
83      public boolean isCatalogOwner() throws OperationNotSupportedException {
84          if (catalogManager == null)
85              throw new OperationNotSupportedException("Not connected to a catalog.");
86          return catalogManager.getOwner().equals(getUserAddress());
87      }
88
89      /**
90       * Returns the catalog manager.
91       * @return the CatalogManager.
92       */
93      public CatalogManager getCatalogManager() {
94          return catalogManager;
95      }
96
97      /**
98       * Returns the user's private key.
99       * @return a String with the user's private key.
100      */
101     public String getPrivateKey() {
102         return privateKey;
103     }
104
```

```
105     /**
106      * Set the role for the current user.
107      * @param role an int specifying the role.
108      */
109     public void setRole(int role) {
110         this.role = role;
111     }
112 
113     /**
114      * Returns the role for the current user.
115      * @return an int specifying the role.
116      */
117     int getRole() {
118         return role;
119     }
120 }
```

Utils.java

```
1  package com.aldodaquino.cobra.gui;
2  
3  import com.aldodaquino.cobra.gui.constants.Images;
4  
5  import javax.swing.*;
6  import java.awt.*;
7  import java.io.File;
8  
9  /**
10  * Utilities for the GUI.
11  * @author Aldo D'Aquino.
12  * @version 1.0.
13  */
14 public class Utils {
15 
16     /**
17      * Show a new centered Window with fixed dimensions and not resizable.
18      * @param title of the Window.
19      * @param panel to show in the Window body.
20      * @param exitOnClose if true exit the Client when the Window is close.
21      * @return the generated JFrame.
22      */
23     public static JFrame newWindow(String title, JComponent panel, boolean exitOnClose) {
24         JFrame window = new JFrame(title);                                    // create a window
25         window.setIconImage(Images.logo.getImage());                          // set logo as
           ↪  application icon
26         window.setContentPane(panel);                                        // put a panel
           ↪  inside the window
27         window.pack();                                  // resize the window based on content size
28         window.setLocationRelativeTo(null);                                  // center the
           ↪  window
29         if (exitOnClose)
30             window.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); // exit program
               ↪  when window gets closed
31         window.setVisible(true);                                             // show it
32         return window;
33     }
34 
35     /**
```

```
36          * Shows a dialog, running on another thread.
37          * @param msg the dialog message.
38          */
39         public static void newMessageDialog(String msg) {
40             newDialog("Info", msg, JOptionPane.INFORMATION_MESSAGE);
41         }
42 
43         /**
44          * Shows an error dialog, running on another thread.
45          * @param msg the error message.
46          */
47         public static void newErrorDialog(String msg) {
48             newDialog("Error", msg, JOptionPane.WARNING_MESSAGE);
49         }
50 
51         // auxiliary function
52         private static void newDialog(String title, String msg, int type) {
53             Thread t = new Thread(() -> JOptionPane.showMessageDialog(null, msg, title, type));
54             t.start();
55         }
56 
57         /**
58          * Shows an error dialog that advice that the program will close, then exit the program.
59          * @param errorMessage the error message.
60          */
61         public static void newExitDialog(String errorMessage) {
62             JOptionPane.showMessageDialog(null, errorMessage, "ERROR! Exiting...",
63                     JOptionPane.ERROR_MESSAGE);
64             System.err.println("Exiting from the program. Reason: " + errorMessage);
65             System.exit(1);
66         }
67 
68         /**
69          * Shows a confirmation dialog (yes/no).
70          * @param msg the question.
71          * @return true for yes, false for no.
72          */
73         @SuppressWarnings("BooleanMethodIsAlwaysInverted")
74         public static boolean newConfirmDialog(String msg) {
75             return JOptionPane.showConfirmDialog(null, msg,"Warning", JOptionPane.YES_NO_OPTION) ==
76                     JOptionPane.YES_OPTION;
77         }
78 
79         /**
80          * Set the font size of a label.
81          * @param label of which set the font size.
82          * @param fontSize the size to set.
83          */
84         public static void setFontSize(JLabel label, int fontSize) {
85             label.setFont(new Font(label.getFont().getName(), Font.PLAIN, fontSize));
86         }
87 
88         /**
89          * Show the file selection dialog for file choosing and saving.
90          * @return a File.
91          */
92         public static File openFileDialog() {
93             return fileDialog(true, null);
94         }
```

```
95
96      /**
97       * Show the file selection dialog for file choosing and saving.
98       * @param defaultName specify the original filename of the incoming file.
99       * @return a File.
100      */
101     public static File saveFileDialog(String defaultName) {
102         return fileDialog(false, defaultName);
103     }
104
105     // auxiliary function
106     private static File fileDialog(boolean isOpenDialog, String filename) {
107         File selected = null;
108         boolean aFileIsSelected = false;
109
110         JFileChooser chooser = null;
111         LookAndFeel previousLF = UIManager.getLookAndFeel();
112         try {
113             UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
114             chooser = new JFileChooser();
115             UIManager.setLookAndFeel(previousLF);
116         } catch (IllegalAccessException | UnsupportedLookAndFeelException |
            ↪  InstantiationException |
117                 ClassNotFoundException e) {
118             e.printStackTrace();
119         }
120
121         if (chooser == null) chooser = new JFileChooser();
122         if (filename != null) chooser.setSelectedFile(new File(filename));
123
124         do {
125             chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
126             int returnStatus = isOpenDialog ? chooser.showOpenDialog(null) :
                ↪  chooser.showSaveDialog(null);
127
128             if (returnStatus == JFileChooser.APPROVE_OPTION)
129                 selected = chooser.getSelectedFile();
130             else continue;
131
132             if (!isOpenDialog && selected.exists()) {
133                 aFileIsSelected = newConfirmDialog("The file will be overwritten. Are you
                    ↪  sure?");
134             } else if (!isOpenDialog && selected.exists() && !selected.canWrite()) {
135                 newErrorDialog("Can't write in the specified path. Please try again.");
136             } else if (isOpenDialog && !selected.canRead()) {
137                 newErrorDialog("Can't read the selected file. Please try again.");
138             } else {
139                 aFileIsSelected = true;
140             }
141
142         } while (!aFileIsSelected);
143
144         return selected;
145     }
146
147 }
```

1.4.1 DAPP/gui/components

AsyncPanel.java

```
1  package com.aldodaquino.cobra.gui.components;
2
3  import javax.swing.*;
4  import javax.swing.event.AncestorEvent;
5  import javax.swing.event.AncestorListener;
6  import java.awt.*;
7
8  /**
9   * A JPanel that listen for ancestor changes. If it is added to an ancestor it saves it a
   ↪  protected variable window.
10  * Include a doAsync method that runs the Runnable in another Thread and set the glass panel of
   ↪  the window in a loading
11  * state to prevent actions from the user.
12  * @author Aldo D'Aquino.
13  * @version 1.0.
14  */
15 public class AsyncPanel extends JPanel {
16
17     protected JFrame window;
18
19     /**
20      * Constructor.
21      */
22     protected AsyncPanel() {
23
24         addAncestorListener(new AncestorListener() {
25
26             @Override
27             public void ancestorAdded(AncestorEvent event) {
28                 Component ancestor = event.getAncestor();
29                 if (ancestor.getClass() == JFrame.class)
30                     window = (JFrame) ancestor;
31             }
32
33             @Override
34             public void ancestorRemoved(AncestorEvent event) {
35                 window = null;
36             }
37
38             @Override
39             public void ancestorMoved(AncestorEvent event) {
40                 ancestorAdded(event);
41             }
42
43         });
44     }
45
46     /**
47      * Run a runnable asynchronously. Shows a loading panel during the loading.
48      * @param runnable to be run.
49      */
50     protected void doAsync(Runnable runnable) {
51         startLoading(window);
52         new Thread(() -> {
53             runnable.run();
54             stopLoading(window);
```

```
        }).start();
    }

    private static void startLoading(JFrame window) {
        if (window == null) return;
        window.setGlassPane(ComponentFactory.newSpinner());
        window.getGlassPane().setVisible(true);
    }

    private static void stopLoading(JFrame window) {
        if (window == null) return;
        window.getGlassPane().setVisible(false);
    }
}
```

AuthorContentTable.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.panels.AuthorInfoPanel;
import com.aldodaquino.cobra.gui.panels.GenreInfoPanel;
import com.aldodaquino.cobra.main.Content;

import javax.swing.*;
import javax.swing.table.TableCellRenderer;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.List;

/**
 * A JTable for the {@link com.aldodaquino.cobra.gui.panels.AuthorPanel}.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class AuthorContentTable extends JTable {

    private static final String[] colNames = {"Address", "Name", "Author", "Genre", "Views",
    ↪  "Enjoy", "Price fairness",
            "Content meaning", "Price"};

    private static Object[][] prepareRows(List<Content> contents) {
        Object[][] rows = new Object[contents.size()][colNames.length];
        for (int i = 0; i < contents.size(); i++) {
            String address = contents.get(i).address;
            rows[i][0] = address;
            rows[i][1] = contents.get(i).name;
            rows[i][2] = contents.get(i).author;
            rows[i][3] = contents.get(i).genre;
            rows[i][4] = contents.get(i).views;
            rows[i][5] = contents.get(i).enjoy;
            rows[i][6] = contents.get(i).priceFairness;
            rows[i][7] = contents.get(i).contentMeaning;
            rows[i][8] = contents.get(i).price;
        }
        return rows;
    }
```

```
41
42      /**
43       * Constructor.
44       * @param status the Status object.
45       * @param contents a List of Content objects.
46       */
47      public AuthorContentTable(Status status, List<Content> contents) {
48          super(prepareRows(contents), colNames);
49
50          // render author and genre as link style
51          TableCellRenderer linkRenderer = (table, value, arg2, arg3, arg4, arg5) ->
52                  new JLabel("<html><a href=\"about:" + value + "\">" + value + "</a>");
53          getColumnModel().getColumn(2).setCellRenderer(linkRenderer);
54          getColumnModel().getColumn(3).setCellRenderer(linkRenderer);
55
56          // mouse listener for author and genre click and hover
57          addMouseListener(new MouseAdapter() {
58              @Override
59              public void mouseClicked(MouseEvent e) {
60                  int row = rowAtPoint(new Point(e.getX(), e.getY()));
61                  int col = columnAtPoint(new Point(e.getX(), e.getY()));
62                  String cellContent = (String) getModel().getValueAt(row, col);
63                  if (col == 2) AuthorInfoPanel.newWindow(status, cellContent);
64                  if (col == 3) GenreInfoPanel.newWindow(status, cellContent);
65              }
66
67              @Override
68              public void mouseEntered(MouseEvent e) {
69                  int col = columnAtPoint(new Point(e.getX(), e.getY()));
70                  if (col == 2 || col == 3) {
71                      setCursor(new Cursor(Cursor.HAND_CURSOR));
72                  }
73              }
74
75              @Override
76              public void mouseExited(MouseEvent e) {
77                  int col = columnAtPoint(new Point(e.getX(), e.getY()));
78                  if (col != 2 && col != 3) {
79                      setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
80                  }
81              }
82          });
83      }
84
85      // Make cells not editable
86      @Override
87      public boolean isCellEditable(int row, int column) {
88          return false;
89      }
90
91  }
```

CatalogForm.java

```
1   package com.aldodaquino.cobra.gui.components;
2
3   import com.aldodaquino.cobra.gui.Utils;
4   import com.aldodaquino.cobra.gui.constants.Dimensions;
```

```java

import javax.swing.*;
import java.util.function.Consumer;

/**
 * A JPanel used in the {@link StarPanel}. Ask the user which catalog to connect to.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class CatalogForm extends JPanel {

    private final JTextField catalogAddressField;

    private final Consumer<String> connectCallback;

    /**
     * Constructor.
     * @param connectCallback a callback to be invoked when the deploy button is clicked or the form is submitted and
     *                        the catalog address is correct.
     * @param deployCallback a callback invoked when the deploy button is clicked.
     */
    public CatalogForm(Consumer<String> connectCallback, Runnable deployCallback) {
        this.connectCallback = connectCallback;

        // set layout (vertical)
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        // Labels
        JLabel catalogAddressLabel = new JLabel("Catalog address:");

        // catalogManager address field: on enter connect
        catalogAddressField = ComponentFactory.newTextField(e -> connect());

        // Buttons
        JButton connectButton = ComponentFactory.newButton("Connect", e -> connect());
        JButton deployButton = ComponentFactory.newButton("Deploy", e -> deployCallback.run());

        // titled border panel for catalogManager connection
        JPanel connectPanel = ComponentFactory.newTitledBorderPanel("Existent catalogManager");
        connectPanel.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
        connectPanel.add(catalogAddressLabel);
        connectPanel.add(catalogAddressField);
        connectPanel.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
        connectPanel.add(connectButton);

        // titled border panel for catalogManager connection
        JPanel deployPanel = ComponentFactory.newTitledBorderPanel("New catalogManager");
        deployPanel.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
        deployPanel.add(deployButton);

        // add all to the panel
        add(connectPanel);
        add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M));
        add(deployPanel);
    }

    private void connect() {
        // get input data
```

```
        String address = catalogAddressField.getText().trim();

        // add "0x" to the address if not present
        if (address.length() == 40)
            address = "0x" + address;

        // check the length  of the inputs and validate the form
        if (address.length() == 42)
            connectCallback.accept(address);
        else Utils.newErrorDialog("Check the values entered in the fields.");
    }

}
```

ChartWidget.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.main.CatalogManager;

/**
 * An {@link InfoPanel} used in the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel}.
 * Shows the chart of the catalog, that includes: the latest content, the most popular content,
 ↪  and the highest rated
 * content in absolute and for each category.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class ChartWidget extends InfoPanel {

    /**
     * Constructor.
     * @param status the Status object.
     */
    public ChartWidget(Status status) {
        super(status, "Charts");
        CatalogManager catalogManager = status.getCatalogManager();

        latestLabel.update(catalogManager.getLatest());
        mostPopularLabel.update(catalogManager.getMostPopular());
        highestRatedLabel.update(catalogManager.getMostRated(null));
        mostEnjoyedLabel.update(catalogManager.getMostRated("enjoy"));
        biggestPriceFairnessLabel.update(catalogManager.getMostRated("value for money"));
        highestContentMeaningLabel.update(catalogManager.getMostRated("content"));
    }

}
```

ComponentFactory.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.constants.Images;

import javax.swing.*;
import javax.swing.border.Border;
```

```java
import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.event.ActionListener;

/**
 * Components factory: makes components creation faster and easier.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class ComponentFactory {

    /**
     * Returns a new JButton.
     * @param text the text of the button.
     * @param e the ActionListener of the button.
     * @return a JButton.
     */
    public static JButton newButton(String text, ActionListener e) {
        JButton button = new JButton(text);
        button.addActionListener(e);
        return button;
    }

    /**
     * Returns a new JTextField.
     * @param e the ActionListener called when enter is pressed.
     * @return a JTextField.
     */
    public static JTextField newTextField(ActionListener e) {
        return newField(e, false);
    }

    /**
     * Returns a new JTextField for password. Characters are replaced with dots.
     * @param e the ActionListener called when enter is pressed.
     * @return a JTextField.
     */
    static JTextField newPasswordField(ActionListener e) {
        return newField(e, true);
    }

    // inner class
    private static JTextField newField(ActionListener e, boolean isPassword) {
        JTextField field = isPassword ? new JPasswordField() : new JTextField();
        field.addActionListener(e);
        return field;
    }

    /**
     * Returns a new border with the specified dimensions.
     * @param width the border width.
     * @param height the border height.
     * @return a Border.
     */
    public static Border newBorder(int width, int height) {
        return BorderFactory.createEmptyBorder(height, width,  height, width);
    }

    /**
```

```
66          * Returns a vertical spacer of the specified dimensions.
67          * @param dimension the dimensions.
68          * @return a Component.
69          */
70         public static Component newVSpacer(Dimension dimension) {
71             return Box.createRigidArea(dimension);
72         }
73 
74         /**
75          * Returns a panel with a border with the specified title centered and a vertical layout.
76          * @param title the title string.
77          * @return a JPanel.
78          */
79         static JPanel newTitledBorderPanel(String title) {
80             JPanel panel = new JPanel();
81             TitledBorder titledBorder = BorderFactory.createTitledBorder(title);
82             titledBorder.setTitleJustification(TitledBorder.CENTER);
83             panel.setBorder(titledBorder);
84             panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
85             return panel;
86         }
87 
88         /**
89          * Return a new panel with a centered JLabel containing a loading message with a spinner.
90          * Used in the {@link AsyncPanel}.
91          * @return a JPanel.
92          */
93         static JPanel newSpinner() {
94             JPanel panel = new JPanel();
95             panel.setLayout(new BorderLayout());
96             panel.add(new JLabel("loading... ", Images.loading, JLabel.CENTER));
97             return panel;
98         }
99     }
```

ContentList.java

```
1   package com.aldodaquino.cobra.gui.components;
2 
3   import com.aldodaquino.cobra.gui.Status;
4   import com.aldodaquino.cobra.gui.panels.ContentInfoPanel;
5   import com.aldodaquino.cobra.main.Content;
6 
7   import javax.swing.*;
8   import java.awt.event.MouseAdapter;
9   import java.awt.event.MouseEvent;
10  import java.util.List;
11 
12  /**
13   * a JList of Content objects.
14   * @author Aldo D'Aquino.
15   * @version 1.0.
16   */
17  public class ContentList extends JList<String> {
18 
19      private static String[] prepareRows(List<Content> contents) {
20          String[] rows = new String[contents.size()];
21          for (int i = 0; i < contents.size(); i++)
```

```
            rows[i] = contents.get(i).name;
        return rows;
    }

    /**
     * Constructor.
     * @param status the Status object.
     * @param contents a List of Content objects.
     */
    public ContentList(Status status, List<Content> contents) {
        super(prepareRows(contents));

        // double-click listener
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() < 2) return;
                int index = locationToIndex(e.getPoint());
                ContentInfoPanel.newWindow(status, contents.get(index).address);
            }
        });
    }

}
```

InfoPanel.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.constants.Dimensions;
import com.aldodaquino.cobra.gui.Utils;

import javax.swing.*;
import java.awt.*;

import static com.aldodaquino.cobra.gui.constants.Dimensions.INFO_PANEL_PADDING;

/**
 * Info panel, superclass of {@link com.aldodaquino.cobra.gui.panels.AuthorInfoPanel},
 * {@link com.aldodaquino.cobra.gui.panels.GenreInfoPanel} and {@link ChartWidget}.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class InfoPanel extends JPanel {

    protected final LabelPanel latestLabel;
    protected final LabelPanel mostPopularLabel;
    protected final LabelPanel highestRatedLabel;
    protected final LabelPanel mostEnjoyedLabel;
    protected final LabelPanel biggestPriceFairnessLabel;
    protected final LabelPanel highestContentMeaningLabel;

    /**
     * Constructor. Can be invoked only by its children.
     * @param status the Status object.
     * @param mainLabelString the String to be putted in the top of the panel, with a bigger
↪   font.
     */
```

```
32      protected InfoPanel(Status status, String mainLabelString) {
33
34          // set layout and border
35          setLayout(new GridBagLayout());
36          setBorder(ComponentFactory.newBorder(INFO_PANEL_PADDING.width,
            ↪  INFO_PANEL_PADDING.height));
37
38          // prepare content
39          JLabel mainLabel = new JLabel(mainLabelString);
40          Utils.setFontSize(mainLabel, mainLabel.getFont().getSize() * 2);
41          latestLabel = new LabelPanel(status, "Latest release: ");
42          mostPopularLabel = new LabelPanel(status, "Most popular content: ");
43          highestRatedLabel = new LabelPanel(status, "Highest rated content: ");
44          mostEnjoyedLabel = new LabelPanel(status, "Most enjoyed content: ");
45          biggestPriceFairnessLabel = new LabelPanel(status, "Biggest value for money content:
            ↪  ");
46          highestContentMeaningLabel = new LabelPanel(status, "Highest rated for content meaning:
            ↪  ");
47
48          // add all to the panel
49          add(mainLabel, UpgradablePanel.newGBC(1, 1));
50          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), UpgradablePanel.newGBC(1, 2));
51          add(latestLabel, UpgradablePanel.newGBC(1, 3));
52          add(mostPopularLabel, UpgradablePanel.newGBC(1, 4));
53          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), UpgradablePanel.newGBC(1, 5));
54          add(highestRatedLabel, UpgradablePanel.newGBC(1, 6));
55          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S), UpgradablePanel.newGBC(1, 7));
56          add(mostEnjoyedLabel, UpgradablePanel.newGBC(1, 8));
57          add(biggestPriceFairnessLabel, UpgradablePanel.newGBC(1, 9));
58          add(highestContentMeaningLabel, UpgradablePanel.newGBC(1, 10));
59
60      }
61
62  }
```

LabelPanel.java

```
1   package com.aldodaquino.cobra.gui.components;
2
3   import com.aldodaquino.cobra.gui.Status;
4   import com.aldodaquino.cobra.gui.constants.Images;
5   import com.aldodaquino.cobra.gui.panels.ContentInfoPanel;
6   import com.aldodaquino.cobra.main.Content;
7
8   import javax.swing.*;
9   import java.awt.*;
10  import java.awt.event.MouseAdapter;
11  import java.awt.event.MouseEvent;
12
13  /**
14   * Label JPanel, contains a fixed label and an updatable value. The initial value is a loader
    ↪  spinner.
15   * @author Aldo D'Aquino.
16   * @version 1.0.
17   */
18  public class LabelPanel extends UpgradablePanel {
19
20      private final JLabel loader = new JLabel(new ImageIcon(Images.loading.getImage()),
        ↪  JLabel.CENTER);
```

```
    private final GridBagConstraints replacingPosition = newGBC(2, 1);

    private final Status status;

    /**
     * Constructor.
     * @param status the Status object.
     * @param label the label for the value.
     */
    LabelPanel(Status status, String label) {
        this.status = status;
        add(new JLabel(label), newGBC(1, 1));
        add(loader, replacingPosition);
    }

    /**
     * Set the content name as value of the panel in a link style.
     * @param content the Content object.
     */
    public void update(Content content) {
        JLabel link = new JLabel(content == null ? ""
                : "<html><a href=\"about:" + content.address + "\">" + content.name + "</a>");
        // onClick show content panel
        if (content != null) link.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                ContentInfoPanel.newWindow(status, content.address);
            }
            @Override
            public void mouseEntered(MouseEvent e) {
                setCursor(new Cursor(Cursor.HAND_CURSOR));
            }

            @Override
            public void mouseExited(MouseEvent e) {
                setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
            }
        });
        replaceComponent(loader, link, replacingPosition);
    }

}
```

LoginForm.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.gui.constants.Dimensions;

import javax.swing.*;
import java.util.function.Consumer;

/**
 * Login Form under the Logo in the Starter Panel.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
```

```java
public class LoginForm extends JPanel {

    private final JTextField privateKeyInput;
    private final Consumer<String> loginCallback;

    /**
     * Constructor.
     * @param loginCallback a String Consumer called if the login button is clicked or enter is pressed and the private
     *                      key is valid.
     */
    public LoginForm(Consumer<String> loginCallback) {
        this.loginCallback = loginCallback;

        // set layout (vertical)
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        // label over the input fields
        JLabel privateKeyLabel = new JLabel("Private key:");

        // private key field: on enter login
        privateKeyInput = ComponentFactory.newPasswordField(e -> login());

        // send button
        JButton sendButton = ComponentFactory.newButton("Login", e -> login());

        // add all to the panel
        add(privateKeyLabel);
        add(privateKeyInput);
        add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
        add(sendButton);
    }

    /**
     * Submit form action.
     * Called when the login button is clicked or enter key is pressed from the private key field.
     */
    private void login() {
        // get input data
        String privateKey = privateKeyInput.getText().trim();

        // check the length  of the inputs and validate the form
        if (privateKey.length() == 64) loginCallback.accept(privateKey);
        else Utils.newErrorDialog("Check the values entered in the fields.");
    }
}
```

Logo.java

```java
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.gui.constants.Dimensions;
import com.aldodaquino.cobra.gui.constants.Images;
import com.aldodaquino.cobra.gui.constants.Strings;

import javax.swing.*;
```

```java
import java.awt.*;

/**
 * Application Logo for the {@link com.aldodaquino.cobra.gui.panels.StarterPanel}.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class Logo extends JPanel {

    /**
     * Constructor.
     */
    public Logo() {
        // icon
        Image icon = Images.logo.getImage()
                .getScaledInstance(Dimensions.LOGO_SIZE, Dimensions.LOGO_SIZE,
                ↪  Image.SCALE_SMOOTH);
        JLabel iconLabel = new JLabel(new ImageIcon(icon), JLabel.CENTER);

        // title
        JLabel title = new JLabel(Strings.appName);
        title.setForeground(Color.BLACK);
        title.setHorizontalAlignment(JLabel.CENTER);
        // find out how much the font can grow in width and calculate the corresponding font
        ↪  size
        Font labelFont = title.getFont();
        String labelText = title.getText();
        int stringWidth = title.getFontMetrics(labelFont).stringWidth(labelText);
        double widthRatio = (double) Dimensions.LOGO_SIZE / (double)stringWidth;
        int newFontSize = (int) (labelFont.getSize() * widthRatio);
        // set the new font size
        Utils.setFontSize(title, newFontSize);

        // put components in a container
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.add(iconLabel);
        container.add(title);

        // prepare this panel
        setAlignmentX(Component.CENTER_ALIGNMENT);
        add(container);
    }
}
```

NewContentsWidget.java

```java
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.main.CatalogManager;
import com.aldodaquino.cobra.main.Content;

import javax.swing.*;
import java.util.List;

/**
```

```
 * A widget inserted in the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel} under the
 ↪  {@link ChartWidget}.
 * Allows to select how many content the user want in the new content list and shows a window
 ↪  with this content.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class NewContentsWidget extends JPanel {

    private final Status status;
    private final CatalogManager catalogManager;
    private final JSpinner numberSpinner;

    /**
     * Constructor.
     * @param status the Status object.
     */
    public NewContentsWidget(Status status) {
        this.status = status;
        catalogManager = status.getCatalogManager();

        JLabel label1 = new JLabel("Get");
        SpinnerNumberModel spinnerModel = new SpinnerNumberModel(10, 0, 100, 1);
        numberSpinner = new JSpinner(spinnerModel);
        JLabel label2 = new JLabel("new contents");
        JButton goButton = ComponentFactory.newButton("Go", e -> getNewContentList());

        add(label1);
        add(numberSpinner);
        add(label2);
        add(goButton);
    }

    private void getNewContentList() {
        List<Content> contents = catalogManager.getNewContentList((int)
        ↪  numberSpinner.getValue());
        JList contentList = new ContentList(status, contents);
        Utils.newWindow("New content list", contentList, false);
    }

}
```

RoleForm.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.constants.Dimensions;

import javax.swing.*;

/**
 * The last of the Panels shows in the {@link com.aldodaquino.cobra.gui.panels.StarterPanel}.
 * Asks the user if want to see the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel} or
 ↪  the
 * {@link com.aldodaquino.cobra.gui.panels.AuthorPanel}.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
```

```java
public class RoleForm extends JPanel {

    /**
     * Constructor.
     * @param browseCallback callback for the "Browser contents" button.
     * @param manageCallback callback for the "Manage my contents" button.
     * @param disconnectCallback callback for the "Disconnect" button.
     * @param deleteCallback callback for the "Delete catalogManager" contents button. If null the button is not shoum.
     */
    public RoleForm(Runnable browseCallback, Runnable manageCallback, Runnable disconnectCallback,
                    Runnable deleteCallback) {

        // set layout (vertical)
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        // Labels
        JLabel title = new JLabel("What do you want to do?");

        // Buttons
        JButton browseButton = ComponentFactory.newButton("Browse contents", e -> browseCallback.run());
        JButton manageButton = ComponentFactory.newButton("Manage my contents", e -> manageCallback.run());
        JButton disconnectButton = ComponentFactory.newButton("Disconnect from catalogManager",
                e -> disconnectCallback.run());

        // add all to the panel
        add(title);
        add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
        add(browseButton);
        add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
        add(manageButton);
        add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M));
        add(disconnectButton);

        // only for catalogManager owner
        if (deleteCallback != null) {
            add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
            add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L));
            JButton deleteButton = ComponentFactory.newButton("Delete catalogManager", e -> deleteCallback.run());
            add(deleteButton);
        }
    }

}
```

StarPanel.java

```java
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.constants.Dimensions;
import com.aldodaquino.cobra.gui.constants.Images;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
```

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import javax.swing.*;

/**
 * Shows rating or ask the user to vote.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class StarPanel extends UpgradablePanel {

    private static final int STARS_NUMBER = 5;

    private int rating;
    private final boolean enabled;
    private final List<JLabel> stars = new ArrayList<>(Collections.nCopies(5, null));

    /**
    * Constructor. Initialize an empty StarPanel, enabled.
    */
    public StarPanel() {
        this(0, true);
    }

    /**
     * Constructor. Initialize a StarPanel with the specified rating, disabled.
     * @param rating in the interval [0, 5]. 0 means no rating.
     */
    public StarPanel(int rating) {
        this(rating, false);
    }

    /**
    * Constructor.
    * @param rating the rating in the interval [0, 5]. 0 means no rating.
    * @param enabled true if users can vote, false to show only the rating.
    */
    private StarPanel(int rating, boolean enabled) {
        stars.add(null);

        setRating(rating);
        this.enabled = enabled;
    }

    /**
    * Returns the rating.
    * @return int in the interval [0, 5]. 0 means no rating.
    */
    public int getRating() {
        return rating;
    }

    private void setRating(int rating) {
        if (rating < 0 || rating > STARS_NUMBER)
            throw new IllegalArgumentException("The rating must appertain at the interval [0, "
            ↪  + STARS_NUMBER + "].");
        this.rating = rating;
```

```
        for (int i = 0; i < STARS_NUMBER; i++) {
            JLabel newStar = getStar(i < rating);
            replaceComponent(stars.get(i), newStar, newGBC(i + 1, 0));
            stars.add(i, newStar);
        }
    }

    private JLabel getStar(boolean filled) {
        ImageIcon icon = filled ? Images.filledStar : Images.emptyStar;
        icon = Images.getScaled(icon, Dimensions.STAR_SIZE);
        JLabel label = new JLabel(icon, JLabel.CENTER);
        if (enabled) label.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                setRating(stars.indexOf(label) + 1);
            }
        });
        return label;
    }

}
```

UpgradablePanel.java

```
package com.aldodaquino.cobra.gui.components;

import java.awt.*;

/**
 * An {@link AsyncPanel} with {@link GridBagLayout}. Implements method to easy replace panels.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class UpgradablePanel extends AsyncPanel {

    /**
     * Constructor. Set the GridBagLayout. Can be called only by its children.
     */
    protected UpgradablePanel() {
        setLayout(new GridBagLayout());
    }

    /**
     * Replace a component with another one. Can be called only by its children.
     * @param toBeReplaced the old component.
     * @param replacement the new component.
     * @param position the position of the component to be replaced.
     */
    protected void replaceComponent(Component toBeReplaced, Component replacement,
    ↪  GridBagConstraints position) {
        if (replacement == null) return;
        if (toBeReplaced != null) remove(toBeReplaced);
        add(replacement, position);
        if (window != null) {
            window.revalidate();
            window.repaint();
            window.pack();
        }
```

```
    }

    /**
     * A public method that help to create GridBagConstraints in less time.
     * @param x the gridx property of the GridBagConstraints object.
     * @param y the gridy property of the GridBagConstraints object.
     * @return a GridBagConstraints object.
     */
    public static GridBagConstraints newGBC(int x, int y) {
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = x;
        gbc.gridy = y;
        return gbc;
    }
}
```

UserInfo.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.constants.Colors;
import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.constants.Dimensions;

import javax.naming.OperationNotSupportedException;
import javax.swing.*;
import java.awt.*;

/**
 * Shows the catalog to which the user is connected and the user address and premium status.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class UserInfo extends UpgradablePanel {

    private final Status status;

    private JLabel catalogAddressLabel;
    private final GridBagConstraints catalogAddressPosition;

    private JLabel accountAddressLabel;
    private final GridBagConstraints accountAddressPosition;

    private JLabel premiumLabel;
    private final GridBagConstraints premiumPosition;

    /**
     * Constructor.
     * @param status the Status object.
     */
    public UserInfo(Status status) {
        this.status = status;

        // catalog label
        JLabel catalog = new JLabel("Catalog:");
        catalogAddressLabel = newCatalogAddressLabel();
        catalogAddressPosition = UpgradablePanel.newGBC(1, 2);

```

```
41          // account label
42          JLabel accountLabel = new JLabel("Account:");
43          accountAddressLabel = newAccountAddressLabel();
44          accountAddressPosition = UpgradablePanel.newGBC(1, 5);
45
46          // premium label
47          premiumLabel = newPremiumLabel();
48          premiumPosition = UpgradablePanel.newGBC(1, 6);
49
50          // add to the panel
51          add(catalog, UpgradablePanel.newGBC(1, 1));
52          add(catalogAddressLabel, catalogAddressPosition);
53          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S), UpgradablePanel.newGBC(1, 3));
54          add(accountLabel, UpgradablePanel.newGBC(1, 4));
55          add(accountAddressLabel, accountAddressPosition);
56          add(premiumLabel, premiumPosition);
57      }
58
59      /**
60       * Update its fields according with the latest information available in status.
61       */
62      public void updateStatus() {
63          JLabel newCatalogAddressLabel = newCatalogAddressLabel();
64          replaceComponent(catalogAddressLabel, newCatalogAddressLabel, catalogAddressPosition);
65          catalogAddressLabel = newCatalogAddressLabel;
66
67          JLabel newAccountAddressLabel = newAccountAddressLabel();
68          replaceComponent(accountAddressLabel, newAccountAddressLabel, accountAddressPosition);
69          accountAddressLabel = newAccountAddressLabel;
70
71          JLabel newPremiumLabel = newPremiumLabel();
72          replaceComponent(premiumLabel, newPremiumLabel, premiumPosition);
73          premiumLabel = newPremiumLabel;
74      }
75
76      private JLabel newAccountAddressLabel() {
77          String account;
78          try {
79              account = status.getUserAddress();
80          } catch (OperationNotSupportedException e) {
81              // not logged in
82              account = "not logged in";
83          }
84          return new JLabel(account);
85      }
86
87      private JLabel newCatalogAddressLabel() {
88          return new JLabel(status.getCatalogManager() == null ? "not connected"
89                  : status.getCatalogManager().getAddress());
90      }
91
92      private JLabel newPremiumLabel() {
93          if (status.getCatalogManager() == null) return new JLabel();     // catalog not
            ↪  connected
94          Boolean isPremium = status.getCatalogManager().isPremium();
95          JLabel newPremiumLabel;
96          if (isPremium) {
97              newPremiumLabel = new JLabel("Premium user");
98              newPremiumLabel.setForeground(Colors.GREEN);
```

```
        } else {
            newPremiumLabel = new JLabel("Not Premium user");
            newPremiumLabel.setForeground(Colors.RED);
        }
        return newPremiumLabel;
    }

}
```

ViewsContentTable.java

```
package com.aldodaquino.cobra.gui.components;

import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.panels.ContentInfoPanel;
import com.aldodaquino.cobra.main.Content;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.List;

/**
 * A content list that includes also the number of views.
 * Accessible in the {@link com.aldodaquino.cobra.gui.panels.CustomerPanel}.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class ViewsContentTable extends JTable {

    private static final String[] colNames = {"Name", "Views"};

    private static Object[][] prepareRows(List<Content> contents) {
        Object[][] rows = new Object[contents.size()][colNames.length];
        for (int i = 0; i < contents.size(); i++) {
            rows[i][0] = contents.get(i).name;
            rows[i][1] = contents.get(i).views;
        }
        return rows;
    }

    /**
     * Constructor.
     * @param status the Status object.
     * @param contents a List of Content objects to be shown in this table.
     */
    public ViewsContentTable(Status status, List<Content> contents) {
        super(prepareRows(contents), colNames);

        // double-click listener
        addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() < 2) return;
                int row = rowAtPoint(new Point(e.getX(), e.getY()));
                ContentInfoPanel.newWindow(status, contents.get(row).address);
            }
```

```
48          });
49      }
50
51      // Make cells not editable
52      @Override
53      public boolean isCellEditable(int row, int column) {
54          return false;
55      }
56
57  }
```

1.4.2 DAPP/gui/constants

Colors.java

```
1  package com.aldodaquino.cobra.gui.constants;
2
3  import java.awt.*;
4
5  /**
6   * Colors constants for the Graphic Interface.
7   * @author Aldo D'Aquino.
8   * @version 1.0.
9   */
10 public class Colors {
11
12     public static final Color GREEN = new Color(0, 150, 0);
13     public static final Color RED = new Color(255, 40, 40);
14
15 }
```

Dimensions.java

```
1  package com.aldodaquino.cobra.gui.constants;
2
3  import java.awt.*;
4
5  /**
6   * Dimensions constant for the Graphic Interface.
7   * @author Aldo D'Aquino.
8   * @version 1.0.
9   */
10 public class Dimensions {
11
12     // icons size
13     public static final int LOGO_SIZE = 128;
14     public static final int STAR_SIZE = 10;
15
16     // borders padding
17     public static final Dimension STARTER_PANEL_PADDING = new Dimension(60, 30);
18     public static final Dimension INFO_PANEL_PADDING = new Dimension(30, 30);
19     public static final Dimension LATERAL_BAR_PADDING = new Dimension(15, 15);
20
21     // spacers and separators
22     public static final Dimension V_SPACER_S = new Dimension(0,5);
23     public static final Dimension V_SPACER_M = new Dimension(0,15);
24     public static final Dimension V_SPACER_L = new Dimension(0,35);
25
26 }
```

Images.java

```
1  package com.aldodaquino.cobra.gui.constants;
2
3
4  import javax.swing.*;
5  import java.awt.*;
6
```

```java
/**
 * Images used in the Graphic Interface.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class Images {

    public static final ImageIcon logo = new ImageIcon(Images.class.getResource("/logo.png"));
    public static final ImageIcon loading = new ImageIcon(Images.class.getResource("/loading.gif"));
    public static final ImageIcon emptyStar = new ImageIcon(Images.class.getResource("/empty-star.png"));
    public static final ImageIcon filledStar = new ImageIcon(Images.class.getResource("/filled-star.png"));

    /**
     * Returns the scaled version of an image.
     * @param icon the original image.
     * @param size the size that you want the final image to have.
     * @return another ImageIcon, scaled.
     */
    public static ImageIcon getScaled(ImageIcon icon, int size) {
        return new ImageIcon(icon.getImage().getScaledInstance(size, size, Image.SCALE_SMOOTH));
    }

}
```

Strings.java

```java
package com.aldodaquino.cobra.gui.constants;

/**
 * Strings constants for the app.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class Strings {

    public static final String appName = "COBrA ĐAPP";

}
```

1.4.3 DAPP/gui/panels

AuthorInfoPanel.java

```
1  package com.aldodaquino.cobra.gui.panels;
2
3  import com.aldodaquino.cobra.gui.Status;
4  import com.aldodaquino.cobra.gui.Utils;
5  import com.aldodaquino.cobra.gui.components.InfoPanel;
6  import com.aldodaquino.cobra.main.CatalogManager;
7
8  /**
9   * Shows the author charts in the catalog.
10  * @see InfoPanel the parent class.
11  * @author Aldo D'Aquino.
12  * @version 1.0.
13  */
14 public class AuthorInfoPanel extends InfoPanel {
15
16     static final String WINDOW_TITLE = "About the author";
17
18     /**
19      * Constructor.
20      * @param status the Status object.
21      * @param author the author address.
22      */
23     AuthorInfoPanel(Status status, String author) {
24         super(status, author);
25         CatalogManager catalogManager = status.getCatalogManager();
26
27         new Thread(() -> latestLabel.update(catalogManager.getLatestByAuthor(author))).start();
28         new Thread(() ->
           ↪ mostPopularLabel.update(catalogManager.getMostPopularByAuthor(author))).start();
29         new Thread(() -> highestRatedLabel.update(catalogManager.getMostRatedByAuthor(author,
           ↪ null))).start();
30         new Thread(() -> mostEnjoyedLabel.update(catalogManager.getMostRatedByAuthor(author,
           ↪ "enjoy"))).start();
31         new Thread(() ->
           ↪ biggestPriceFairnessLabel.update(catalogManager.getMostRatedByAuthor(author,
32                 "value for money"))).start();
33         new Thread(() ->
           ↪ highestContentMeaningLabel.update(catalogManager.getMostRatedByAuthor(author,
34                 "content"))).start();
35     }
36
37     /**
38      * Open a new window with this panel.
39      * @param status the Status.
40      * @param author of the content.
41      */
42     public static void newWindow(Status status, String author) {
43         Utils.newWindow(WINDOW_TITLE, new AuthorInfoPanel(status, author), false);
44     }
45
46 }
```

AuthorPanel.java

```
1  package com.aldodaquino.cobra.gui.panels;
2
```

```java
import com.aldodaquino.cobra.gui.components.AsyncPanel;
import com.aldodaquino.cobra.gui.components.AuthorContentTable;
import com.aldodaquino.cobra.gui.components.ComponentFactory;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.main.CatalogManager;
import com.aldodaquino.cobra.main.Content;
import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.main.ContentManager;

import java.math.BigInteger;
import java.util.List;
import javax.naming.OperationNotSupportedException;
import javax.swing.*;

/**
 * The author panel, a main panel showed after the starter panel if the user have chosen the
 ↪  author role.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class AuthorPanel extends AsyncPanel {

    private final Status status;
    private final CatalogManager catalogManager;

    private final JScrollPane tableContainer;
    private JTable table;

    /**
     * Constructor.
     * @param status the Status object.
     */
    public AuthorPanel(Status status) {
        this.status = status;
        catalogManager = status.getCatalogManager();

        // get the content list
        List<Content> contents;
        try {
            contents = catalogManager.getAuthorContents(status.getUserAddress());
        } catch (OperationNotSupportedException e) {
            throw new RuntimeException(e);
        }

        // listen events
        catalogManager.listenCatalogClosed(() -> Utils.newExitDialog("Catalog closed."));
        for (Content content : contents)
            catalogManager.listenPaymentAvailable(content.address,
                    (addr, name) -> Utils.newMessageDialog("Payment available for content " +
                    ↪  name + "."));

        // table container
        table = new AuthorContentTable(status, contents);
        tableContainer = new JScrollPane();
        tableContainer.setViewportView(table);

        // buttons
        JPanel buttonsPad = new JPanel();
        buttonsPad.setLayout(new BoxLayout(buttonsPad, BoxLayout.X_AXIS));
```

```
60          JButton deployButton = ComponentFactory.newButton("Deploy a new content", e ->
            ↪  deployContent());
61          JButton updateButton = ComponentFactory.newButton("Update table", e -> updateTable());
62          JButton withdrawButton = ComponentFactory.newButton("Withdraw selected", e ->
            ↪  withdrawSelected());
63          buttonsPad.add(deployButton);
64          buttonsPad.add(updateButton);
65          buttonsPad.add(withdrawButton);
66
67          // assemble the panel
68          setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
69          add(tableContainer);
70          add(buttonsPad);
71      }
72
73      private void deployContent() {
74          JPanel deployContentPanel = new DeployContentPanel(status, this::onDeployed);
75          Utils.newWindow("Deploy new content", deployContentPanel, false);
76      }
77
78      private void onDeployed(String address) {
79          // when deployed listen for payment available on this content
80          ContentManager contentManager = new ContentManager(status.credentials, address);
81          contentManager.listenContentPublished(() -> {
82              catalogManager.listenPaymentAvailable(address, (addr, name) ->
83                      Utils.newMessageDialog("Payment available for content " + name + "."));
84              updateTable();
85          });
86      }
87
88      private void updateTable() {
89          doAsync(() -> {
90              try {
91                  List<Content> contents =
                    ↪  catalogManager.getAuthorContents(status.getUserAddress());
92                  table = new AuthorContentTable(status, contents);
93                  tableContainer.setViewportView(table);
94              } catch (OperationNotSupportedException e) {
95                  e.printStackTrace();
96                  Utils.newErrorDialog(e.getMessage());
97                  System.exit(-1);
98              }
99          });
100     }
101
102     private void withdrawSelected() {
103         doAsync(() -> {
104             String address = table.getValueAt(table.getSelectedRow(), 0).toString();
105             BigInteger amount = catalogManager.withdraw(address);
106             if (amount.equals(BigInteger.ZERO))
107                 Utils.newErrorDialog("There is no payout available for this contract.");
108             else Utils.newMessageDialog(amount + " wei collected.");
109         });
110     }
111
112 }
```

ContentInfoPanel.java

```java
package com.aldodaquino.cobra.gui.panels;

import com.aldodaquino.cobra.connections.API;
import com.aldodaquino.cobra.connections.CobraHttpHelper;
import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.gui.components.AsyncPanel;
import com.aldodaquino.cobra.gui.components.ComponentFactory;
import com.aldodaquino.cobra.gui.components.StarPanel;
import com.aldodaquino.cobra.gui.components.UpgradablePanel;
import com.aldodaquino.cobra.gui.constants.Dimensions;
import com.aldodaquino.cobra.main.CatalogManager;
import com.aldodaquino.cobra.main.Content;
import com.aldodaquino.cobra.main.ContentManager;
import com.aldodaquino.javautils.FileExchange;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.util.HashMap;
import java.util.Map;

import static com.aldodaquino.cobra.gui.components.UpgradablePanel.newGBC;
import static com.aldodaquino.cobra.gui.constants.Dimensions.INFO_PANEL_PADDING;

/**
 * Shows info about a content. Allows also to buy and consume the content or gift it to another
 ↪  user.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class ContentInfoPanel extends AsyncPanel {

    private static final String WINDOW_TITLE = "About the author";

    private final Status status;
    private final CatalogManager catalogManager;
    private final Content content;
    private final ContentManager contentManager;

    /**
     * Constructor.
     * @param status the Status object.
     * @param address the content address.
     */
    private ContentInfoPanel(Status status, String address) {
        this.status = status;
        catalogManager = status.getCatalogManager();
        content = catalogManager.getContentInfo(address);
        contentManager = new ContentManager(status.credentials, address);

        // prepare content
        JLabel mainLabel = new JLabel(content.name);
        Utils.setFontSize(mainLabel, mainLabel.getFont().getSize() * 2);
        JLabel addressLabel = new JLabel("Address: " + content.address);
        JPanel authorLabel = prepareLink("Author: ", content.author,
                new AuthorInfoPanel(status, content.author), AuthorInfoPanel.WINDOW_TITLE);
```

```
59          JPanel genreLabel = prepareLink("Genre: ", content.genre,
60                  new GenreInfoPanel(status, content.genre), GenreInfoPanel.WINDOW_TITLE);
61          JLabel priceLabel = new JLabel("price: " + content.price);
62          JLabel viewsLabel = new JLabel("Views: " + content.views);
63          JPanel averageRatingLabel = prepareStar("Average rating: ", content.averageRating);
64          JPanel enjoyLabel = prepareStar("Enjoy: ", content.enjoy);
65          JPanel priceFairnessLabel = prepareStar("Value for money: ", content.priceFairness);
66          JPanel contentMeaningLabel = prepareStar("Content meaning: ", content.contentMeaning);
67          JButton viewButton = ComponentFactory.newButton("View", e -> view());
68          JButton giftButton = ComponentFactory.newButton("Gift this content", e -> gift());
69
70          // prepare the panel
71          setLayout(new GridBagLayout());
72          setBorder(ComponentFactory.newBorder(INFO_PANEL_PADDING.width,
            ↪  INFO_PANEL_PADDING.height));
73          add(mainLabel, newGBC(1, 1));
74          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 2));
75          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 4));
76          add(addressLabel, newGBC(1, 5));
77          add(authorLabel, newGBC(1, 6));
78          add(genreLabel, newGBC(1, 7));
79          add(priceLabel, newGBC(1, 8));
80          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 9));
81          add(viewsLabel, newGBC(1, 10));
82          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 11));
83          add(averageRatingLabel, newGBC(1, 12));
84          add(enjoyLabel, newGBC(1, 13));
85          add(priceFairnessLabel, newGBC(1, 14));
86          add(contentMeaningLabel, newGBC(1, 15));
87          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 16));
88          add(viewButton, newGBC(1, 17));
89          add(giftButton, newGBC(1, 18));
90
91      }
92
93      /**
94       * Open a new window with this panel.
95       * @param status the Status.
96       * @param address of the content.
97       */
98      public static void newWindow(Status status, String address) {
99          Utils.newWindow(WINDOW_TITLE, new ContentInfoPanel(status, address), false);
100     }
101
102     private void view() {
103         doAsync(() -> {
104             // Check if the user has access and ask for buy if he hasn't
105             if (!catalogManager.isPremium() && !catalogManager.hasAccess(content.address)) {
106                 if(!Utils.newConfirmDialog("You don't have access to this content. Do you want
                    ↪  to buy it for "
107                         + content.price + "?")) return; // doesn't have access and doesn't want
                        ↪  to buy the access
108                 if (catalogManager.buyContent(content.address, content.price))
109                     catalogManager.listenAccessGranted((addr, name) -> {
110                         Utils.newMessageDialog("Content bought.");
111                         retrieveContent();
112                     });
113                 else Utils.newErrorDialog("Cannot buy this content. You may have bought it
                    ↪  previously.");
```

```
114             } else retrieveContent();
115         });
116     }
117 
118     private void retrieveContent() {
119         // make the request
120         Map<String, String> parameters = new HashMap<>();
121         parameters.put("privateKey", status.getPrivateKey());
122         parameters.put("address", content.address);
123 
124         String hostname = contentManager.getHostname();
125         int port = contentManager.getPort();
126         if (port == 0) {
127             Utils.newErrorDialog("The content has an invalid port number. Cannot contact the
                ↪  author's server.");
128             return;
129         }
130         String url = "http://" + hostname + ":" + port + API.ACCESS_API_PATH;
131 
132         // get the response and retrieve the socket port number
133         CobraHttpHelper.Response response = CobraHttpHelper.makePost(url, parameters);
134         if (response.code != 200) Utils.newErrorDialog("HTTP ERROR " + response.code + ": " +
            ↪  response.data);
135         Map<String, String> map = CobraHttpHelper.parseJson(response.data);
136         int socketPort = Integer.parseInt(map.get("port"));
137         String filename = map.get("filename");
138 
139         // download the file
140         File file = Utils.saveFileDialog(filename);
141         FileExchange.receiveFile(file, hostname, socketPort);
142     }
143 
144     private void gift() {
145         JPanel pickUserPanel = new PickUserPanel((String user) ->
146                 doAsync(() -> {
147                     if (catalogManager.giftContent(content.address, user, content.price))
148                         catalogManager.listenAccessGranted(user, (address, name) ->
149                                 Utils.newMessageDialog("Content " + name + " gifted to " + user
                                    ↪  + "."));
150                     else Utils.newErrorDialog("Cannot gift this content. The user may have
                        ↪  already bought it.");
151                 }));
152         Utils.newWindow("Gift content", pickUserPanel, false);
153     }
154 
155     private JPanel prepareLink(String label, String value, JPanel onClickPanel, String
        ↪  windowTitle) {
156         JLabel link = new JLabel("<html><a href=\"about:" + value + "\">" + value + "</a>");
157         link.addMouseListener(new MouseAdapter() {
158             @Override
159             public void mouseClicked(MouseEvent e) {
160                 Utils.newWindow(windowTitle, onClickPanel, false);
161             }
162             @Override
163             public void mouseEntered(MouseEvent e) {
164                 setCursor(new Cursor(Cursor.HAND_CURSOR));
165             }
166 
167             @Override
```

```
            public void mouseExited(MouseEvent e) {
                setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
            }
        });
        JPanel panel = new JPanel(new GridBagLayout());
        panel.add(new JLabel(label), UpgradablePanel.newGBC(1, 1));
        panel.add(link, UpgradablePanel.newGBC(2, 1));
        return panel;
    }

    private JPanel prepareStar(String label, int rating) {
        JPanel panel = new JPanel(new GridBagLayout());
        panel.add(new JLabel(label), UpgradablePanel.newGBC(1, 1));
        panel.add(new StarPanel(rating), UpgradablePanel.newGBC(2, 1));
        return panel;
    }

}
```

CustomerPanel.java

```
package com.aldodaquino.cobra.gui.panels;

import com.aldodaquino.cobra.gui.components.*;
import com.aldodaquino.cobra.gui.constants.Dimensions;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.main.CatalogManager;
import com.aldodaquino.cobra.gui.Status;

import javax.swing.*;
import java.awt.*;

import static com.aldodaquino.cobra.gui.constants.Dimensions.LATERAL_BAR_PADDING;

/**
 * The customer panel, a main panel showed after the starter panel if the user have chosen the
 ↪  customer role.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class CustomerPanel extends UpgradablePanel {

    private final Status status;
    private final CatalogManager catalogManager;

    private final JScrollPane tableContainer;
    private Component table;
    private final JPanel lateralBar;
    private final UserInfo userInfo;
    private ChartWidget chartWidget;
    private final GridBagConstraints chartWidgetPosition;

    private boolean showViews = false;

    /**
     * Constructor.
     * @param status the Status object.
     */
```

```
37      public CustomerPanel(Status status) {
38          this.status = status;
39          catalogManager = status.getCatalogManager();
40
41          // listen for events
42          catalogManager.listenCatalogClosed(() -> Utils.newExitDialog("Catalog closed."));
43          catalogManager.listenNewContentAvailable((name, address) ->
44                  Utils.newMessageDialog("New content available: " + name + "."));
45          catalogManager.listenNewContentAvailable(this::update);
46          catalogManager.listenFeedbackAvailable((address, name) ->
47                  Utils.newWindow("Vote content", new VotingPanel(address, name, catalogManager),
                    ↪  false));
48
49          // table container
50          tableContainer = new JScrollPane();
51          table = getTable();
52          tableContainer.setViewportView(table);
53
54          // lateral bar
55          userInfo = new UserInfo(status);
56          JButton buyPremiumButton = ComponentFactory.newButton("Buy premium", e ->
            ↪  buyPremium());
57          JButton giftPremiumButton = ComponentFactory.newButton("Gift premium", e ->
            ↪  giftPremium());
58          JButton updateButton = ComponentFactory.newButton("Refresh", e -> update());
59          JButton showHideViewsButton = ComponentFactory.newButton("Show/hide views", e -> {
60              showViews = !showViews;
61              update();
62          });
63          chartWidget = new ChartWidget(status);
64          JPanel newContentWidget = new NewContentsWidget(status);
65
66          lateralBar = new JPanel(new GridBagLayout()) {
67              // prevent widely resize with window
68              @Override
69              public Dimension getMaximumSize() {
70                  Dimension dim = super.getMaximumSize();
71                  dim.width = getPreferredSize().width;
72                  return dim;
73              }
74              // minimum size to fit all component
75              @Override
76              public Dimension getMinimumSize() {
77                  return getPreferredSize();
78              }
79          };
80
81          lateralBar.add(userInfo, newGBC(1, 1));
82          lateralBar.setBorder(ComponentFactory.newBorder(LATERAL_BAR_PADDING.width,
            ↪  LATERAL_BAR_PADDING.height));
83          lateralBar.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M), newGBC(1, 2));
84          lateralBar.add(buyPremiumButton, newGBC(1, 3));
85          lateralBar.add(giftPremiumButton, newGBC(1, 4));
86          lateralBar.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 5));
87          lateralBar.add(updateButton, newGBC(1, 6));
88          lateralBar.add(showHideViewsButton, newGBC(1, 7));
89          lateralBar.add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 8));
90          chartWidgetPosition = newGBC(1, 9);
91          lateralBar.add(chartWidget, chartWidgetPosition);
```

```
92          lateralBar.add(newContentWidget, newGBC(1, 10));
93
94          // assemble the panel
95          setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
96          add(tableContainer);
97          add(lateralBar);
98      }
99
100     private void update() {
101         doAsync(() -> {
102             // update table
103             table = getTable();
104             tableContainer.setViewportView(table);
105
106             // update user info
107             userInfo.updateStatus();
108
109             // update charts
110             lateralBar.remove(chartWidget);
111             chartWidget = new ChartWidget(status);
112             lateralBar.add(chartWidget, chartWidgetPosition);
113         });
114     }
115
116     private Component getTable() {
117         return showViews ? new ViewsContentTable(status,
            ↪ catalogManager.getContentListWithViews())
118                 : new ContentList(status, catalogManager.getContentList());
119     }
120
121     private void buyPremium() {
122         doAsync(() -> {
123             if (catalogManager.buyPremium())
124                 // show pop-up only when the event is fired
125                 catalogManager.listenBecomesPremium(() -> Utils.newMessageDialog("Premium
                    ↪ bought."));
126             else Utils.newErrorDialog("UNKNOWN ERROR: cannot buy a premium subscription.");
127         });
128         userInfo.updateStatus();
129     }
130
131     private void giftPremium() {
132         JPanel pickUserPanel = new PickUserPanel((String user) ->
133                 doAsync(() -> {
134                     if (catalogManager.giftPremium(user))
135                         catalogManager.listenBecomesPremium(user, ()
                            ↪ ->Utils.newMessageDialog("Premium gifted."));
136                     else Utils.newErrorDialog("UNKNOWN ERROR: cannot gift a premium
                        ↪ subscription.");
137                 }));
138         Utils.newWindow("Gift premium", pickUserPanel, false);
139     }
140
141 }
```

DeployContentPanel.java

```
1 package com.aldodaquino.cobra.gui.panels;
2
```

```
3  import com.aldodaquino.cobra.connections.API;
4  import com.aldodaquino.cobra.gui.Status;
5  import com.aldodaquino.cobra.gui.components.AsyncPanel;
6  import com.aldodaquino.cobra.gui.components.ComponentFactory;
7  import com.aldodaquino.cobra.gui.constants.Dimensions;
8  import com.aldodaquino.cobra.gui.Utils;
9  import com.aldodaquino.cobra.connections.CobraHttpHelper;
10 import com.aldodaquino.javautils.FileExchange;
11
12 import javax.swing.*;
13 import java.io.File;
14 import java.math.BigInteger;
15 import java.nio.channels.ServerSocketChannel;
16 import java.util.HashMap;
17 import java.util.Map;
18 import java.util.function.Consumer;
19
20 /**
21  * Panel to deploy a new content.
22  * @author Aldo D'Aquino.
23  * @version 1.0.
24  */
25 class DeployContentPanel extends AsyncPanel {
26
27     // fields
28     private final JTextField addressField;
29     private final JTextField portField;
30     private final JTextField nameField;
31     private final JTextField genreField;
32     private final JTextField priceField;
33
34     private final Status status;
35     private final Consumer<String> deployCallback;
36
37     private File file;
38
39     /**
40      * Constructor.
41      * @param status the Status object.
42      * @param deployCallback a Consumer of content address, called after that the content has
    ↪  been deployed.
43      */
44     DeployContentPanel(Status status, Consumer<String> deployCallback) {
45         this.status = status;
46         this.deployCallback = deployCallback;
47
48         // set layout (vertical)
49         setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
50
51         // label over the input fields
52         JLabel introLabel = new JLabel("The deployed content will be placed in your author
           ↪  server. Specify the " +
53                 "address and the port (default 8080) of a server running an author server
                   ↪  instance. The author " +
54                 "server must always be online so that the content is accessible.");
55         JLabel addressLabel = new JLabel("Address (IP or domain:");
56         JLabel portLabel = new JLabel("Port:");
57         JLabel nameLabel = new JLabel("Name:");
58         JLabel genreLabel = new JLabel("Genre:");
```

```
59          JLabel priceLabel = new JLabel("Price:");
60          JLabel selectFileLabel = new JLabel("Pick the content file:");
61
62          // input field
63          priceField = ComponentFactory.newTextField(e -> {});
64          genreField = ComponentFactory.newTextField(e -> priceField.grabFocus());
65          nameField = ComponentFactory.newTextField(e -> genreField.grabFocus());
66          portField = ComponentFactory.newTextField(e -> nameField.grabFocus());
67          portField.setText("8080");
68          addressField = ComponentFactory.newTextField(e -> portField.grabFocus());
69          addressField.setText("localhost");
70
71          // buttons
72          JButton selectFileButton = ComponentFactory.newButton("Open", e -> file =
            ↪  Utils.openFileDialog());
73          JButton sendButton = ComponentFactory.newButton("Deploy", e -> deploy());
74
75          // add all to the panel
76          add(introLabel);
77          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L));
78          add(addressLabel);
79          add(addressField);
80          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
81          add(portLabel);
82          add(portField);
83          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
84          add(nameLabel);
85          add(nameField);
86          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
87          add(genreLabel);
88          add(genreField);
89          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
90          add(priceLabel);
91          add(priceField);
92          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
93          add(selectFileLabel);
94          add(selectFileButton);
95          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
96          add(sendButton);
97      }
98
99      /**
100      * Submit form action.
101      * Called when the deploy button is clicked.
102      */
103     private void deploy() {
104         // get input data
105         String url = addressField.getText().trim();
106         if (url.length() == 0) {
107             Utils.newErrorDialog("You must specify an url.");
108             return;
109         }
110         try {
111             String portS = portField.getText().trim();
112             int port = portS.equals("") ? 8080 : Integer.parseInt(portS);
113             if (port <= 0) throw new NumberFormatException();
114             url = "http://" + url + ":" + port + API.DEPLOY_API_PATH;
115         } catch (NumberFormatException e) {
116             Utils.newErrorDialog("Invalid port number.");
```

```
            return;
        }

        String name = nameField.getText().trim();
        if (name.length() == 0) {
            Utils.newErrorDialog("You must specify a name.");
            return;
        }
        String genre = genreField.getText().trim();
        String priceS = priceField.getText().trim();
        BigInteger price;
        try {
            price = new BigInteger(priceS.length() != 0 ? priceS : "0");
        } catch (NumberFormatException e) {
            Utils.newErrorDialog("Invalid port number.");
            return;
        }

        // open the socket for the file
        if (file == null) {
            Utils.newErrorDialog("You must choose a file to be uploaded.");
            return;
        }

        ServerSocketChannel serverSocketChannel = FileExchange.openFileSocket();
        if (serverSocketChannel == null) {
            Utils.newErrorDialog("Error while opening server socket.");
            return;
        }

        int port = serverSocketChannel.socket().getLocalPort();
        FileExchange.startFileSender(serverSocketChannel, file,
                () -> Utils.newMessageDialog("File uploaded successfully."));

        // make the request
        Map<String, String> parameters = new HashMap<>();
        parameters.put("privateKey", status.getPrivateKey());
        parameters.put("name", name);
        parameters.put("genre", genre);
        parameters.put("price", price.toString());
        parameters.put("port", Integer.toString(port));
        parameters.put("filename", file.getName());

        CobraHttpHelper.Response response = CobraHttpHelper.makePost(url, parameters);
        if (response.code != 200) {
            Utils.newErrorDialog("HTTP ERROR " + response.code + ": " + response.data);
            return;
        }

        // close the widow
        deployCallback.accept(response.data);
        window.dispose();
    }
}
```

GenreInfoPanel.java

```
package com.aldodaquino.cobra.gui.panels;

```

```java
import com.aldodaquino.cobra.gui.Status;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.gui.components.InfoPanel;
import com.aldodaquino.cobra.main.CatalogManager;

/**
 * Shows the genre charts in the catalog.
 * @see InfoPanel the parent class.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class GenreInfoPanel extends InfoPanel {

    static final String WINDOW_TITLE = "About the author";

    /**
     * Constructor.
     * @param status the Status object.
     * @param genre the genre.
     */
    GenreInfoPanel(Status status, String genre) {
        super(status, genre);
        CatalogManager catalogManager = status.getCatalogManager();

        new Thread(() -> latestLabel.update(catalogManager.getLatestByGenre(genre))).start();
        new Thread(() ->
        ↪  mostPopularLabel.update(catalogManager.getMostPopularByGenre(genre))).start();
        new Thread(() -> highestRatedLabel.update(catalogManager.getMostRatedByGenre(genre,
        ↪  null))).start();
        new Thread(() -> mostEnjoyedLabel.update(catalogManager.getMostRatedByGenre(genre,
        ↪  "enjoy"))).start();
        new Thread(() ->
        ↪  biggestPriceFairnessLabel.update(catalogManager.getMostRatedByGenre(genre,
                "value for money"))).start();
        new Thread(() ->
        ↪  highestContentMeaningLabel.update(catalogManager.getMostRatedByGenre(genre,
                "content"))).start();
    }

    /**
     * Open a new window with this panel.
     * @param status the Status.
     * @param genre of the content.
     */
    public static void newWindow(Status status, String genre) {
        Utils.newWindow(WINDOW_TITLE, new GenreInfoPanel(status, genre), false);
    }

}
```

PickUserPanel.java

```java
package com.aldodaquino.cobra.gui.panels;

import com.aldodaquino.cobra.gui.components.AsyncPanel;
import com.aldodaquino.cobra.gui.components.ComponentFactory;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.gui.constants.Dimensions;
```

```
7
8  import javax.swing.*;
9  import java.util.function.Consumer;
10
11 /**
12  * Asks the user for another user address.
13  * @author Aldo D'Aquino.
14  * @version 1.0.
15  */
16 class PickUserPanel extends AsyncPanel {
17     private final JTextField addressField;
18     // the callback to call if the input data are correct
19     private final Consumer<String> giftCallback;
20
21     /**
22      * Constructor.
23      * @param giftCallback a Consumer of user address, invoked when the button is clicked or
    ↪  enter is pressed if the
24      *                     address has a valid format.
25      */
26     PickUserPanel(Consumer<String> giftCallback) {
27         this.giftCallback = giftCallback;
28
29         // set layout (vertical)
30         setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
31
32         // label over the input fields
33         JLabel addressLabel = new JLabel("User address:");
34
35         // input field
36         addressField = ComponentFactory.newTextField(e -> gift());
37
38         // send button
39         JButton giftButton = ComponentFactory.newButton("Gift", e -> gift());
40
41         // add all to the panel
42         add(addressLabel);
43         add(addressField);
44         add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
45         add(giftButton);
46     }
47
48     /**
49      * Submit form action.
50      * Called when the gift button is clicked or enter key is pressed from the address field.
51      */
52     private void gift() {
53         // get input data
54         String address = addressField.getText().trim();
55
56         // add "0x" to the address if not present
57         if (address.length() == 40)
58             address = "0x" + address;
59
60         // check the length  of the inputs and validate the form
61         if (address.length() == 42)
62             giftCallback.accept(address);
63         else Utils.newErrorDialog("You must specify an address.");
64
```

```
65          // close the widow
66          window.dispose();
67      }
68  }
```

StarterPanel.java

```
1   package com.aldodaquino.cobra.gui.panels;
2
3   import com.aldodaquino.cobra.gui.components.*;
4   import com.aldodaquino.cobra.gui.constants.Dimensions;
5   import com.aldodaquino.cobra.gui.Utils;
6   import com.aldodaquino.cobra.gui.Status;
7
8   import javax.naming.OperationNotSupportedException;
9   import javax.swing.*;
10  import java.awt.*;
11  import java.util.function.Consumer;
12
13  import static com.aldodaquino.cobra.gui.constants.Dimensions.STARTER_PANEL_PADDING;
14
15  /**
16   * The starter panel. Main panel showed when the app starts.
17   * Manage the user login and require all the data to start the app.
18   * @author Aldo D'Aquino.
19   * @version 1.0.
20   */
21  public class StarterPanel extends UpgradablePanel {
22
23      private final Status status = new Status();
24      private final Consumer<Status> whenDone;
25
26      private final UserInfo userInfo;
27      private final JPanel loginForm;
28      private JPanel catalogForm;
29      private JPanel roleForm;
30
31      private final GridBagConstraints replacingPosition;
32
33      /**
34       * Constructor.
35       * @param whenDone a consumer of Status object called when all the requested fields in the
    ↪  status object are
36       *                 completed.
37       */
38      public StarterPanel(Consumer<Status> whenDone) {
39          this.whenDone = whenDone;
40
41          // init components
42          JPanel logo = new Logo();
43          userInfo = new UserInfo(status);
44          loginForm = new LoginForm(this::loginCallback);
45          replacingPosition = newGBC(1, 5);
46
47          // prepare the panel and add components
48          setBorder(ComponentFactory.newBorder(STARTER_PANEL_PADDING.width,
            ↪  STARTER_PANEL_PADDING.height));
49          add(logo, newGBC(1, 1));
```

```
50          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 2));
51          add(userInfo, newGBC(1, 3));
52          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L), newGBC(1, 4));
53          add(loginForm, replacingPosition);
54      }
55
56      // minimum size to fit all component
57      @Override
58      public Dimension getMinimumSize() {
59          return getPreferredSize();
60      }
61
62      /* CALLBACKS */
63      private void loginCallback(String privateKey) {
64          doAsync(() -> {
65              try {
66                  // set the status
67                  status.login(privateKey);
68                  // update the user info
69                  userInfo.updateStatus();
70                  // change form
71                  catalogForm = new CatalogForm(this::connectCallback, this::deployCallback);
72                  replaceComponent(loginForm, catalogForm, replacingPosition);
73              } catch (OperationNotSupportedException e) {
74                  e.printStackTrace();
75                  Utils.newErrorDialog(e.getMessage());
76              }
77          });
78      }
79
80      private void connectCallback(String catalogAddress) {
81          doAsync(() -> {
82              try {
83                  status.connectCatalog(catalogAddress);
84                  postConnect();
85              } catch (OperationNotSupportedException e) {
86                  e.printStackTrace();
87                  Utils.newErrorDialog(e.getMessage());
88              }
89          });
90      }
91
92      private void deployCallback() {
93          doAsync(() -> {
94              try {
95                  status.deployCatalog();
96                  postConnect();
97              } catch (OperationNotSupportedException e) {
98                  e.printStackTrace();
99                  Utils.newErrorDialog(e.getMessage());
100             }
101         });
102     }
103
104     private void postConnect() throws OperationNotSupportedException {
105         userInfo.updateStatus();
106         Runnable deleteCallback = status.isCatalogOwner() ? this::deleteCallback : null;
107         roleForm = new RoleForm(this::browseCallback, this::manageCallback,
            ↪  this::disconnectCallback, deleteCallback);
```

```
108             replaceComponent(catalogForm, roleForm, replacingPosition);
109         }
110 
111         private void disconnectCallback() {
112             doAsync(() -> {
113                 status.disconnectCatalog();
114                 userInfo.updateStatus();
115                 replaceComponent(roleForm, catalogForm, replacingPosition);
116             });
117         }
118 
119         private void deleteCallback() {
120             if (!Utils.newConfirmDialog("Do you really want to delete this catalog?")) return;
121             doAsync(() -> {
122                 if (status.getCatalogManager().suicide()) {
123                     Utils.newMessageDialog("Catalog deleted.");
124                     disconnectCallback();
125                 } else Utils.newErrorDialog("UNKNOWN ERROR: the catalog is not deleted.");
126             });
127         }
128 
129         private void browseCallback() {
130             status.setRole(Status.ROLE_CUSTOMER);
131             whenDone.accept(status);
132         }
133 
134         private void manageCallback() {
135             status.setRole(Status.ROLE_AUTHOR);
136             whenDone.accept(status);
137         }
138 
139     }
```

VotingPanel.java

```
1   package com.aldodaquino.cobra.gui.panels;
2   
3   import com.aldodaquino.cobra.gui.components.AsyncPanel;
4   import com.aldodaquino.cobra.gui.components.ComponentFactory;
5   import com.aldodaquino.cobra.gui.components.StarPanel;
6   import com.aldodaquino.cobra.gui.constants.Dimensions;
7   import com.aldodaquino.cobra.main.CatalogManager;
8   
9   import javax.swing.*;
10  
11  /**
12   * A panel to ask the user to vote a content consumed recently.
13   * @author Aldo D'Aquino.
14   * @version 1.0.
15   */
16  class VotingPanel extends AsyncPanel {
17  
18      /**
19       * Constructor.
20       * @param contentName the content name.
21       * @param contentAddress the content address.
22       * @param catalogManager the CatalogManager loaded or deployed by the user.
23       */
```

```
24      VotingPanel(String contentName, String contentAddress, CatalogManager catalogManager) {
25
26          // prepare components
27          JLabel infoLabel = new JLabel("You can now rate for the content " + contentName + ".");
28          JLabel enjoyLabel = new JLabel("Enjoy:");
29          StarPanel enjoyStars = new StarPanel();
30          JLabel valueForMoneyLabel = new JLabel("Value for money:");
31          StarPanel valueForMoneyStars = new StarPanel();
32          JLabel contentMeaningLabel = new JLabel("Content meaning:");
33          StarPanel contentMeaningStars = new StarPanel();
34          JButton voteButton = ComponentFactory.newButton("Vote", e -> {
35              catalogManager.vote(contentAddress, enjoyStars.getRating(),
                ↪  valueForMoneyStars.getRating(),
36                      contentMeaningStars.getRating());
37              window.dispose();
38          });
39
40          // prepare the panel
41          setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
42          add(infoLabel);
43          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_L));
44          add(enjoyLabel);
45          add(enjoyStars);
46          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
47          add(valueForMoneyLabel);
48          add(valueForMoneyStars);
49          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_S));
50          add(contentMeaningLabel);
51          add(contentMeaningStars);
52          add(ComponentFactory.newVSpacer(Dimensions.V_SPACER_M));
53          add(voteButton);
54      }
55  }
```

1.5 DAPP/test (not required)

Main.java

```
package com.aldodaquino.cobra.test;

import com.aldodaquino.javautils.CliHelper;
import com.aldodaquino.javautils.FileExchange;
import com.aldodaquino.cobra.connections.API;
import com.aldodaquino.cobra.connections.CobraHttpHelper;
import com.aldodaquino.cobra.gui.Utils;
import com.aldodaquino.cobra.main.CatalogManager;
import org.web3j.crypto.Credentials;

import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.nio.channels.ServerSocketChannel;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ThreadLocalRandom;

/**
 * Prepares an environment for tests:
 *  - deploys a CatalogContract;
 *  - starts an author-server on the default port 8080;
 *  - deploy some contents;
 *  - start two instances of the GUI, one for the Customer and one for the Author.
 * You have to login with the rights private key on the GUIs.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class Main {

    private static final boolean START_GUIS = true;

    private static final int NUMBER_OF_CONTENTS = 15;
    private static final String[] genres = {"Comedy", "Romance", "Thriller"};
    private static URI FILENAME;

    // static constructor
    static {
        try {
            FILENAME = Main.class.getResource("/test_file.png").toURI();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    // Change this value with your private keys.
    @SuppressWarnings("SpellCheckingInspection")
    private static final String CATALOG_OWNER_DEFAULT_KEY =
            "73bb2d6a2fb0776eaa90f299b18ced9a490cbfbf07bb1df88cb019e3ea2f75c8";
    @SuppressWarnings("SpellCheckingInspection")
    private static final String[] AUTHOR_DEFAULT_KEYS =
            {"48fb33ee64f893e159ad374206b2f17e60206606ebaf9f09ebc6ab7359e95055",
            "df50467e8c890cd6539be6a19212ecf2528a7e04f4cf1ba320d1f06079978630"};

    /**
```

```
57          * Starts the tests.
58          * @param args a String[], optionally containing 2 options:
59          *             -p --private-key    Catalog owner private key.
60          *             -a --author         Author's private key.
61          *             If args is null or an option is missing the default option will be used.
62          */
63         public static void main(String[] args) {
64
65             // Parse cmd options
66             CliHelper cliHelper = new CliHelper();
67             cliHelper.addOption("h", "help", false, "Print this help message.");
68             cliHelper.addOption("k", "private-key", true, "Catalog owner private key.");
69             cliHelper.addOption("a", "author", true, "Author's private key.");
70             cliHelper.parse(args);
71
72             String catalogOwnerKeyOpt = cliHelper.getValue("private-key");
73             String catalogOwnerKey = catalogOwnerKeyOpt.length() > 0 ? catalogOwnerKeyOpt :
               ↪  CATALOG_OWNER_DEFAULT_KEY;
74
75             String[] authorKeysOpt = cliHelper.getValues("author");
76             String[] authorKeys = authorKeysOpt.length > 0 ? authorKeysOpt : AUTHOR_DEFAULT_KEYS;
77
78
79             // Create credentials for catalog owner's private key
80             Credentials catalogOwnerCredentials = Credentials.create(catalogOwnerKey);
81             System.out.println("Created credentials for catalog owner. Account address: "
82                     + catalogOwnerCredentials.getAddress() + ".\n");
83
84
85             // Deploy a new Catalog and retrieve the address
86             CatalogManager catalogManager = new CatalogManager(catalogOwnerCredentials);
87             String catalogAddress = catalogManager.getAddress();
88             System.out.println("Deployed a new CatalogContract. Catalog address: " + catalogAddress
               ↪  + ".\n");
89
90
91             // Start n author servers on ports 8000-(8000+n-1),
92             // bound with the catalog and associated to the author's private key
93             System.out.println("Starting "+ authorKeys.length + " author servers, please wait...");
94             int[] port = {8000};  // "Clickety-click... Barba-trick!": in lambda expressions we can
               ↪  only use final
95                                   // (or effectively final) variables.
96             for (String authorKey : authorKeys) {
97                 new Thread(() -> {
98                     try {
99                         com.aldodaquino.cobra.authorserver.Main.main(
100                                new String[]{"-k", authorKey, "-c", catalogAddress, "-n",
                                   ↪  "localhost",
101                                        "-p", Integer.toString(port[0]++)});
102                    } catch (IOException e) {
103                        e.printStackTrace();
104                    }
105                }).start();
106            }
107
108            // Wait 3 seconds for the server to become online.
109            try {
110                Thread.sleep(3000);
111            } catch (InterruptedException e) {
```

```
112             System.err.println("Interrupted during while waiting the author server becomes
                ↪  online.");
113             e.printStackTrace();
114         }
115
116
117         // Deploying contents
118         for (int i = 0; i < NUMBER_OF_CONTENTS; i++) {
119             int authorIndex = rand(authorKeys.length);
120             deploy(8000 + authorIndex, authorKeys[authorIndex], "Content " + (i+1),
121                     genres[rand(genres.length)], Integer.toString(rand(5) * 5000));
122         }
123
124
125         // Start the GUI
126         if (START_GUIS) {
127             System.out.print("Starting two GUI windows...");
128             Process GUI1 = newGUIProcess();
129             Process GUI2 = newGUIProcess();
130
131             // Wait for the GUIs to end
132             try {
133                 assert GUI1 != null && GUI2 != null;
134                 GUI1.waitFor();
135                 GUI2.waitFor();
136                 System.out.println("    GUI 1 exit with value " + GUI1.exitValue()
137                         + ".\n    GUI 2 exit with value " + GUI2.exitValue() + ".\n\n");
138                 System.exit(GUI1.exitValue() + GUI2.exitValue()); // 0 if none fails, 1 if one
                    ↪  fails, 2 if both fail.
139             } catch (InterruptedException e) {
140                 e.printStackTrace();
141                 System.exit(1);
142             }
143         }
144
145     }
146
147     /* Auxiliary functions */
148
149     private static int rand(int lessThan) {
150         return ThreadLocalRandom.current().nextInt(0, lessThan);
151     }
152
153     private static void deploy(int serverPort, String authorKey, String name, String genre,
        ↪  String price) {
154         // assemble the url
155         String url = "http://localhost:" + serverPort + API.DEPLOY_API_PATH;
156
157         // prepare the file
158         ServerSocketChannel serverSocketChannel = FileExchange.openFileSocket();
159         if (serverSocketChannel == null) {
160             Utils.newErrorDialog("Error while opening server socket.");
161             return;
162         }
163
164         int port = serverSocketChannel.socket().getLocalPort();
165         FileExchange.startFileSender(serverSocketChannel, new File(FILENAME),
166                 () -> System.out.println("File uploaded successfully."));
167
```

```
168             // make the request
169             Map<String, String> parameters = new HashMap<>();
170             parameters.put("privateKey", authorKey);
171             parameters.put("name", name);
172             parameters.put("genre", genre);
173             parameters.put("price", price);
174             parameters.put("port", Integer.toString(port));
175 
176             System.out.println("Deploying a content..." +
177                     "\n    Url: " + url +
178                     "\n    Author key: " + authorKey +
179                     "\n    Name: " + name +
180                     "\n    Genre: " + genre +
181                     "\n    Price: " + price);
182 
183             CobraHttpHelper.Response response = CobraHttpHelper.makePost(url, parameters);
184             if (response.code != 200) System.err.println("Something went wrong. Response" +
                ↪  response.toString());
185             else System.out.println("Deployed successfully.\n");
186         }
187 
188         private static Process newGUIProcess() {
189             String javaHome = System.getProperty("java.home");
190             String javaBin = javaHome + File.separator + "bin" + File.separator + "java";
191             String classpath = System.getProperty("java.class.path");
192             String className = com.aldodaquino.cobra.gui.Main.class.getCanonicalName();
193             ProcessBuilder builder = new ProcessBuilder(javaBin, "-cp", classpath, className);
194             try {
195                 return builder.start();
196             } catch (IOException e) {
197                 e.printStackTrace();
198             }
199             return null;
200         }
201 
202     }
```

1.6 javautils

CliHelper.java

```java
package com.aldodaquino.javautils;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.stream.Stream;

/**
 * Help parsing the args[]. You can add the option you want retrieve and this class will parse it automatically.
 * @author Aldo D'Aquino.
 * @version 1.0.
 */
public class CliHelper {

    private final ArrayList<CliOption> cliOptions = new ArrayList<>();
    private final ArrayList<CliFlag> cliFlags = new ArrayList<>();
    private int maxLongOptLength = 0;

    /**
     * Add available option for the CLI.
     * @param shortOpt like "-v".
     * @param longOpt like "--verbose".
     * @param hasValue true if the option must have a value (i.e. -o value),
     *                 false if is an option without value (i.e. --help).
     * @param description the description of the option to be shown in the help message.
     * @throws IllegalArgumentException if the shortOpt or the longOpt already exists in another option.
     */
    public void addOption(String shortOpt, String longOpt, boolean hasValue, String description) {
        if (!hasValue) addFlag(shortOpt, longOpt, description);
        // search for an already existent option
        for (CliOption cliOption : cliOptions) {
            if (cliOption.isEqual(shortOpt) || cliOption.isEqual(longOpt))
                throw new IllegalArgumentException("Option already exist.");
        }
        // add to the options
        cliOptions.add(new CliOption(shortOpt, longOpt, description));
    }

    // Internal auxiliary method
    private void addFlag(String shortOpt, String longOpt, String description) {
        // search for an already existent option
        for (CliFlag cliFlag : cliFlags) {
            if (cliFlag.isEqual(shortOpt) || cliFlag.isEqual(longOpt))
                throw new IllegalArgumentException("Option already exist.");
        }
        // add to the options
        cliFlags.add(new CliFlag(shortOpt, longOpt, description));
    }

    /**
     * Parse the String[] args of the main and saves the option value.
     * @param args the main's args.
     */
    public void parse(String[] args) {
```

```
54          if (args == null) return;
55          for (int i = 0; i < args.length; i++) {
56              if (isNotAnOption(args[i])) System.err.println("Invalid option " + args[i] + ".");
57              if (i + 1 < args.length && isNotAnOption(args[i + 1])) { // args[i+1] is the value
                ↪  of args[i]
58                  for (CliOption cliOption : cliOptions)
59                      if (cliOption.parse(args[i], args[i+1])) break;
60                  i++;    // skip i + 1: is not an option
61              } else { // args[i] has no value, so is a flag
62                  for (CliFlag cliFlag : cliFlags)
63                      if (cliFlag.parse(args[i])) break;
64              }
65          }
66      }
67
68      /**
69       * Return the list of all values of an option. For example -o value1 -o value2 will return
    ↪  [value1, value2].
70       * @param opt the option in the short or the long format (will return the same list).
71       * @return a String[] containing all the values, empty array if the option has no value or
    ↪  null if this option
72       * doesn't exist.
73       */
74      public String[] getValues(String opt) {
75          for (CliOption cliOption : cliOptions)
76              if (cliOption.isEqual(opt)) return cliOption.getValues();
77          return null;
78      }
79
80      /**
81       * Return first value of an option. For example -o value1 -o value2 will return value1.
82       * Use it for functions that are intended usable only once.
83       * @param opt the option in the short or the long format (will return the same list).
84       * @return the first value, empty string if the option has no value or null if this option
    ↪  doesn't exist.
85       */
86      public String getValue(String opt) {
87          String[] values = getValues(opt);
88          return values == null ? null : values.length == 0 ? "" : values[0];
89      }
90
91      /**
92       * Return true if the program is launched with the specified option, false otherwise.
93       * @param opt the option in the short or the long format (will return the same list).
94       * @return true if the program is launched with the specified option, false otherwise.
95       */
96      public boolean isPresent(String opt) {
97          for (CliFlag cliFlag : cliFlags)
98              if (cliFlag.isEqual(opt))
99                  return cliFlag.isPresent();
100         return false;
101     }
102
103     /**
104      * Return a formatted help message showing the usage. The message has this format:
105      * usage:
106      * -o --longopt    an option description
107      * -h --help       shows help
108      * @return String of the message.
```

```
109         */
110     public String getHelpMessage() {
111         // create a sorted collection with all the objects
112         ArrayList<CliObject> cliObjects = new ArrayList<>();
113         cliObjects.addAll(cliOptions);
114         cliObjects.addAll(cliFlags);
115         cliObjects.sort(Comparator.comparing(o -> o.shortOpt));
116 
117         // prepare the help string
118         StringBuilder stringBuilder = new StringBuilder("Usage:\n");
119         String initialString = "";
120         for (CliObject cliObject : cliObjects) {
121             stringBuilder.append(initialString).append(cliObject.shortOpt).append("
                ↪  ").append(cliObject.longOpt);
122             // append enough spaces to align the descriptions plus a tab (4 spaces) as
                ↪  separator
123             for (int i = 0; i < maxLongOptLength - cliObject.longOpt.length() + 4; i++)
124                 stringBuilder.append(" ");
125             stringBuilder.append(cliObject.description);
126             initialString = "\n";   // from now append a line break before the new line
127         }
128         return stringBuilder.toString();
129     }
130 
131     /**
132      * Return a missing option message with this format:
133      * Missing an option: -o -option    description of the option.
134      * @param opt the short or long code of the option that you want (i.e. "o");
135      * @return String of the message.
136      */
137     public String getMissingOptionMessage(String opt) {
138         StringBuilder stringBuilder = new StringBuilder();
139         Stream.concat(cliOptions.stream(),cliFlags.stream()).forEachOrdered(cliObject -> {
140             if (cliObject.isEqual(opt))
141                 stringBuilder.append("Missing an
                    ↪  option:\n").append(cliObject.shortOpt).append(" ")
142                         .append(cliObject.longOpt).append("
                            ↪  ").append(cliObject.description);
143         });
144         return stringBuilder.toString();
145     }
146 
147     // Internal auxiliary method
148     private boolean isNotAnOption(String string) {
149         return !string.contains("--") && !string.contains("-");
150     }
151 
152 
153     /* Auxiliary classes */
154 
155     private class CliObject {
156         final String shortOpt;
157         final String longOpt;
158         String description;
159 
160         CliObject(String shortOpt, String longOpt, String description) {
161             // add minuses if not present in the head of the string
162             this.shortOpt = shortOpt.length() >= 1 && shortOpt.substring(0, 1).equals("-") ?
                ↪  shortOpt : "-" + shortOpt;
```

```
163             this.longOpt = longOpt.length() >= 2 && longOpt.substring(0, 2).equals("--") ?
                ↪  longOpt : "--" + longOpt;
164
165             if (shortOpt.length() > 2) throw new IllegalArgumentException("Short option must be
                ↪  a single letter.");
166             if (longOpt.length() > maxLongOptLength) maxLongOptLength = longOpt.length();
167
168             this.description = description;
169         }
170
171         boolean isEqual(String opt) {
172             if (opt.length() >= 2 && opt.substring(0, 2).equals("--")) return
                ↪  longOpt.equals(opt);  // is a long option
173             if (opt.length() >= 1 && opt.substring(0, 1).equals("-")) return
                ↪  shortOpt.equals(opt);  // is a short option
174             return shortOpt.equals("-" + opt) || longOpt.equals("--" + opt);    // does not
                ↪  contains minuses
175         }
176     }
177
178     private class CliOption extends CliObject {
179
180         final ArrayList<String> values = new ArrayList<>();
181
182         CliOption(String shortOpt, String longOpt, String description) {
183             super(shortOpt, longOpt, description);
184         }
185
186         boolean parse(String opt, String value) {
187             if (isEqual(opt)) {
188                 values.add(value);
189                 return true;
190             }
191             return false;
192         }
193
194         String[] getValues() {
195             return values.toArray(new String[0]);
196         }
197
198     }
199
200     private class CliFlag extends CliObject {
201
202         boolean found;
203
204         CliFlag(String shortOpt, String longOpt, String description) {
205             super(shortOpt, longOpt, description);
206         }
207
208         boolean parse(String opt) {
209             if (isEqual(opt)) {
210                 found = true;
211                 return true;
212             }
213             return false;
214         }
215
216         boolean isPresent() {
```

```
217             return found;
218         }
219
220     }
221
222 }
```

FileExchange.java

```
1  package com.aldodaquino.javautils;
2
3  import java.io.*;
4  import java.net.*;
5  import java.nio.ByteBuffer;
6  import java.nio.channels.FileChannel;
7  import java.nio.channels.ServerSocketChannel;
8  import java.nio.channels.SocketChannel;
9  import java.nio.file.StandardOpenOption;
10
11 /**
12  * Utility for exchanging file.
13  * Contains method to receive data from a Socket and save it to a File and to read a file and
   ↪  write data to the socket.
14  * @author Aldo D'Aquino.
15  * @version 1.2.
16  */
17 public class FileExchange {
18
19     /**
20      * Receive and save a file.
21      * @param destFile the destination file where save data.
22      * @param hostname of the sender.
23      * @param port of the sender.
24      */
25     public static void receiveFile(File destFile, String hostname, int port) {
26         Thread asyncWriter = new Thread(() -> {
27             int failedCount = 0;
28             boolean stop = false;
29             do {
30                 try (SocketChannel socket = SocketChannel.open(new InetSocketAddress(hostname,
                   ↪  port))) {
31                     System.out.println("Started download: " + destFile.getAbsolutePath() +
                       ↪  ".");
32                     writeFile(socket, destFile);
33                     System.out.println("Download finished.");
34                     stop = true;
35                 } catch (IOException e) {
36                     if (failedCount < 3) {
37                         try {
38                             failedCount++;
39                             Thread.sleep(5000);
40                         } catch (InterruptedException intExc) {
41                             intExc.printStackTrace();
42                             break;
43                         }
44                     }
45                     else {
46                         stop = true;
```

```
                        System.err.println("Can't connect to the sender.");
                        e.printStackTrace();

                    }
                }
            } while (!stop);
        });
        asyncWriter.start();
    }

    /**
     * Opens a socket on which waits an incoming connection from the file recipient.
     * @return a ServerSocketChannel.
     */
    public static ServerSocketChannel openFileSocket() {
        try {
            ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
            serverSocketChannel.bind(null);
            return serverSocketChannel;
        }
        catch (IOException e) {
            System.err.println("Error while opening a socket for sending file.");
            e.printStackTrace();
        }

        return null;
    }

    /**
     * When the other client has connected to the socket it behaves as a server and sends the
    ↪ file.
     * @param serverSocket the ServerSocketChannel opened in openFileSocket.
     * @param file the File to be sent.
     * @param callback optional callback to be run when the upload is finished.
     */
    public static void startFileSender(ServerSocketChannel serverSocket, File file, Runnable
    ↪ callback) {
        Thread listener = new Thread(() -> {
            try {
                serverSocket.socket().setSoTimeout(60000);  // 1 minute
                SocketChannel socketChannel = serverSocket.accept();
                System.out.println("Started upload: " + file.getAbsolutePath() + ".");
                readFile(file, socketChannel);
                System.out.println("Upload finished.");
                if (callback != null) callback.run();
            } catch (IOException e) {
                System.err.println("Error while accepting connection to send file.");
                e.printStackTrace();
            }
        });
        listener.start();
    }

    /**
     * Read a File using NIO channels. Send the file to a socket.
     * @param file File object, the file to be read.
     * @param outChannel Socket where the file will be sent.
     * @throws IOException if the file not exists or is not readable.
     */
```

```
104     public static void readFile(File file, SocketChannel outChannel) throws IOException {
105         FileChannel inChannel = FileChannel.open(file.toPath(), StandardOpenOption.READ);
106         long size = inChannel.size();
107         ByteBuffer sizeBuffer = ByteBuffer.allocate(8);
108         sizeBuffer.putLong(size);
109         sizeBuffer.flip();
110         while (sizeBuffer.hasRemaining()) outChannel.write(sizeBuffer);
111 
112         long transferred = 0;
113         while (size - transferred > 0)
114             transferred += inChannel.transferTo(transferred, size - transferred, outChannel);
115     }
116 
117     /**
118      * Read the specified Socket using NIO, and save the data to File.
119      * @param inChannel Socket from where data will be read.
120      * @param file File object, destination of the data.
121      * @throws IOException if the file not exists or is not writeable.
122      */
123     public static void writeFile(SocketChannel inChannel, File file) throws IOException {
124         FileChannel outChannel = FileChannel.open(file.toPath(),
125                 StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING,
                    ↪  StandardOpenOption.WRITE);
126         ByteBuffer sizeBuffer = ByteBuffer.allocate(8);
127         while (sizeBuffer.hasRemaining())
128             inChannel.read(sizeBuffer);
129         sizeBuffer.flip();
130 
131         long size = sizeBuffer.getLong();
132         long transferred = 0;
133 
134         while (size - transferred > 0)
135             transferred += outChannel.transferFrom(inChannel, transferred, size - transferred);
136     }
137 
138 }
```

HttpHelper.java

```
1  package com.aldodaquino.javautils;
2 
3  import com.sun.net.httpserver.HttpExchange;
4 
5  import java.io.*;
6  import java.net.HttpURLConnection;
7  import java.net.URL;
8  import java.net.URLDecoder;
9  import java.util.HashMap;
10 import java.util.Map;
11 
12 /**
13  * Contains method that help to make http request.
14  * Works with JSON body for POST request and query-style GET parameters.
15  * @author Aldo D'Aquino.
16  * @version 1.1.
17  */
18 public class HttpHelper {
19 
```

```
20      /* SERVER SIDE */
21
22      /**
23       * Parse a GET request and return a Map with keys equals to parameters name and values
    ↪   equals to the parameters
24       * values.
25       * @param request the HttpExchange request received by the handler.
26       * @return Map where both keys and values are strings containing the parameters.
27       */
28      public static Map<String, String> parseGET(HttpExchange request) {
29          String query = request.getRequestURI().getRawQuery();
30          if (query == null || query.length() == 0) throw new IllegalArgumentException("Invalid
            ↪  query: null.");
31          return parseQuery(query);
32      }
33
34      /**
35       * Parse a query and return a Map with keys equals to parameters name and values equals to
    ↪   the parameters values.
36       * @param query the String representing the query.
37       * @return Map where both keys and values are strings containing the parameters.
38       */
39      public static Map<String, String> parseQuery(String query) {
40          // HashMap to be filled with all parameters in the query
41          Map<String, String> parameters = new HashMap<>();
42
43          // Split the query in pairs key=value
44          String pairs[] = query.split("[&]");
45          // Split each pair in key and value and put them in the Map
46          for (String pair : pairs) {
47              String param[] = pair.split("[=]");
48              if (param.length > 0) {
49                  try {
50                      String key = URLDecoder.decode(param[0],
                        ↪  System.getProperty("file.encoding"));
51                      String value = null;
52                      if (param.length > 1) value = URLDecoder.decode(param[1],
                        ↪  System.getProperty("file.encoding"));
53                      parameters.put(key, value);
54                  } catch (UnsupportedEncodingException e) {
55                      e.printStackTrace();
56                  }
57              }
58          }
59          return parameters;
60      }
61
62      /**
63       * Parse a POST request and return a Map with keys equals to parameters name and values
    ↪   equals to the parameters
64       * values.
65       * @param request the HttpExchange request received by the handler.
66       * @return Map where both keys and values are strings containing the parameters.
67       */
68      public static Map<String, String> parsePOST(HttpExchange request) {
69          String json = null;
70          try {
71              InputStreamReader isr = new InputStreamReader(request.getRequestBody(), "utf-8");
72              json = new BufferedReader(isr).readLine();
```

```
73          } catch (IOException e) {
74              e.printStackTrace();
75          }
76          return parseJson(json);
77      }
78
79      /**
80       * Parse a JSON and return a Map with keys equals to parameters name and values equals to
    ↪  the parameters values.
81       * @param json the String representing the stringified JSON.
82       * @return Map where both keys and values are strings containing the parameters.
83       */
84      public static Map<String, String> parseJson(String json) {
85          // parse the JSON body.
86          if (json == null || json.length() == 0) throw new IllegalArgumentException("Invalid
            ↪  query: null.");
87
88          // HashMap to be filled with all parameters in the query
89          Map<String, String> parameters = new HashMap<>();
90
91          // remove parenthesis and quotes
92          json = json.replace("{", "").replace("}", "").replaceAll("\"", "");
93
94          // Split the query in pairs key=value
95          String pairs[] = json.split("[,]");
96          // Split each pair in key and value and put them in the Map
97          for (String pair : pairs) {
98              String param[] = pair.split("[:]");
99              if (param.length > 0) {
100                 String key = param[0];
101                 String value = null;
102                 if (param.length > 1) value = param[1];
103                 parameters.put(key, value);
104             }
105         }
106         return parameters;
107     }
108
109     /**
110      * Send a response to an HttpExchange request.
111      * @param request the request.
112      * @param response a String containing the response.
113      * @param code the status code of the response.
114      */
115     public static void sendResponse(HttpExchange request, String response, int code) {
116         try {
117             request.sendResponseHeaders(code, response.length());
118             OutputStream os = request.getResponseBody();
119             os.write(response.getBytes());
120             os.close();
121         } catch (IOException e) {
122             e.printStackTrace();
123         }
124     }
125
126     /**
127      * Send a response to an HttpExchange request.
128      * @param request the request.
129      * @param response a String containing the response..
```

```
130         */
131     public static void sendResponse(HttpExchange request, String response) {
132         sendResponse(request, response, 200);
133     }
134 
135 
136     /* CLIENT SIDE */
137 
138     /**
139      * Make a GET request on the specified url.
140      * @param url the url to be called.
141      * @param parameters a map containing all the parameters that you want to be passed when
    ↪  the url is called.
142      * @return a {@link Response} object.
143      */
144     public static Response makeGet(String url, Map<String, String> parameters) {
145         return makeRequest(url + querifyParameters(parameters), "GET", "");
146     }
147 
148     /**
149      * Make a POST request on the specified url.
150      * @param url the url to be called.
151      * @param parameters a map containing all the parameters that you want to be passed in the
    ↪  body.
152      * @return a {@link Response} object.
153      */
154     public static Response makePost(String url, Map<String, String> parameters) {
155         return makeRequest(url, "POST", jsonifyParameters(parameters));
156     }
157 
158     // internal function
159     private static Response makeRequest(String url, String method, String parameters) {
160         HttpURLConnection connection = null;
161         int status = -1;
162         try {
163             //Create connection
164             connection = (HttpURLConnection) new URL(url).openConnection();
165             connection.setDoOutput(true);
166             connection.setRequestMethod(method);
167             connection.setRequestProperty("Content-Type", "application/json");
168             connection.setRequestProperty("Content-Length",
                ↪  Integer.toString(parameters.getBytes().length));
169             connection.setRequestProperty("Content-Language", "en-US");
170             connection.setConnectTimeout(5000);
171             connection.setReadTimeout(5000);
172 
173             //Send request
174             if(!parameters.equals("")) {
175                 DataOutputStream outputStream = new
                    ↪  DataOutputStream(connection.getOutputStream());
176                 outputStream.writeBytes(parameters);
177                 outputStream.close();
178             }
179 
180             //Get Response
181             status = connection.getResponseCode();
182             BufferedReader bufferedReader = new BufferedReader(new
                ↪  InputStreamReader(connection.getInputStream()));
183             StringBuilder responseData = new StringBuilder();
```

```
184             String line;
185             String separator = "";
186             while ((line = bufferedReader.readLine()) != null) {
187                 responseData.append(separator).append(line);
188                 separator = "\n";
189             }
190             bufferedReader.close();
191 
192             return new Response(status, responseData.toString());
193         }
194         catch (IOException e) {
195             e.printStackTrace();
196             return status < 0 ? null : new Response(status, "");
197         }
198         finally {
199             if (connection != null) connection.disconnect();
200         }
201     }
202 
203     /**
204      * Return a stringified JSON with the passed parameters.
205      * @param parameters a Map where both keys and values are strings containing the
    ↪  parameters.
206      * @return the stringified JSON.
207      */
208     public static String jsonifyParameters(Map<String, String> parameters) {
209         if (parameters == null) return "";
210         StringBuilder stringBuilder = new StringBuilder();
211         stringBuilder.append("{");
212         String separator = "";
213         for (Map.Entry<String, String> entry : parameters.entrySet()) {
214             stringBuilder.append(separator)
215 
                    ↪  .append("\"").append(entry.getKey()).append("\":\"").append(entry.getValue()).appe
216             separator = ",";
217         }
218         stringBuilder.append("}");
219         return stringBuilder.toString();
220     }
221 
222     /**
223      * Return a string containing the parameters in the query format, ready to be appended to
    ↪  an url for a GET request.
224      * @param parameters a Map where both keys and values are strings containing the
    ↪  parameters.
225      * @return the String in the url format.
226      */
227     public static String querifyParameters(Map<String, String> parameters) {
228         if (parameters == null || parameters.size() == 0) return "";
229         StringBuilder stringBuilder = new StringBuilder();
230         stringBuilder.append("?");
231         String separator = "";
232         for (Map.Entry<String, String> entry : parameters.entrySet()) {
233             stringBuilder.append(separator)
234                     .append(entry.getKey()).append("=").append(entry.getValue());
235             separator = "&";
236         }
237         return stringBuilder.toString();
238     }
```

```
239
240     /**
241      * Response class returned by the makeGet and makePost method.
242      * Contains two field: the response code and the data String of the response.
243      * @author Aldo D'Aquino.
244      * @version 1.0.
245      */
246     public static class Response {
247         public final int code;
248         public final String data;
249         private Response (int code, String data) {
250             this.code = code;
251             this.data = data;
252         }
253
254         /**
255          * Returns this object as a stringified JSON.
256          * @return a String representing the stringified JSON.
257          */
258         @Override
259         public String toString() {
260             return "{\"code\":\"" + code + "\",\"data\":\"" + data + "\"}";
261         }
262     }
263
264 }
```