1 Account.java

```java
package socialmedia;
import java.io.Serializable;

/**
 * Account provides attributes and methods to permit the creation of users on the platform.
 *
 * @author Students: 720014004, 720033851
 * @version 1.0
 */

public class Account implements Serializable {

    //Private instance attributes

    private int id; //Sequential unique ID
    private String handle; //Unique handle
    private String description;


    /**
    * Constructor for Account class, setting handle and ID instance attributes.
    * <p>
    * @param handle the account's handle
    * @param id the account's sequential ID
    */

    public Account(String handle, int id) {
        this.id = id; //Assign ID
        this.handle = handle; //Assign handle
    }

    /**
    * Constructor for Account class, setting handle, ID, and description instance attributes.
    * <p>
    * @param handle the account's handle
    * @param id the account's sequential ID
    * @param description the account's description text
    */

    public Account(String handle, int id, String description){
        this.id = id; //Assign ID
        this.handle = handle; //Assign handle
        this.description = description; //Assign description
    }

    //Public instance attribute getters

    public int getID() {
        return id;
    }

    public String getHandle() {
```

```
53          return handle;
54      }
55
56      public String getDescription() {
57          return description;
58      }
59
60      //Public instance attribute setters
61
62      public void setHandle(String handle) {
63          this.handle = handle;
64      }
65
66      public void setDescription(String description) {
67          this.description = description;
68      }
69
70      /**
71      * Helper method, generates a formated string displaying details
72      * of an Account instance, provided the number of authored posts and recieved endorsements.
73      * <p>
74      * @param noPosts the number of posts associated with the account instance
75      * @param noEndorsements the number of endorsements recieved by the account instance
76      * @return a formatted string containing the account details.
77      */
78
79      public String generateAccountDetails(int noPosts, int noEndorsements) {
80
81          String accountSummary = String.format("ID: %s \n" +
82          "Handle: %s \n" +
83          "Description: %s \n" +
84          "Post count: %s \n" +
85          "Endorse count: %s",
86          id, handle, description, noPosts, noEndorsements);
87
88          return accountSummary;
89      }
90  }
```

2 BasePost.java

```
1   package socialmedia;
2   import java.io.Serializable;
3   import java.util.ArrayList;
4
5   /**
6    * BasePost represents a generic post (of any type), and provides attributes and methods shared
7    * between normal, endorsement, and comment posts.
8    *
9    * @author Students: 720014004, 720033851
10   * @version 1.0
11   */
12
13  public class BasePost implements Serializable {
```

```
14
15      private int id; //Sequential unique ID
16      //Post type - 'normal' (0), 'comment' (1), 'endorsement' (2), 'generic empty post' (3)
17      private int postType;
18      private String message;
19      private String author; //Author's account handle
20
21
22      //BasePost Getter methods
23
24      public int getID() {
25          return id;
26      }
27
28      public int getPostType() {
29          return postType;
30      }
31
32      public String getMessage() {
33          return message;
34      }
35
36
37      public String getAuthor() {
38          return author;
39      }
40
41     /**
42      * The method returns the comments of a post, when provided the platform data.
43      * <p>
44      * @param socialPlatform - the Platform object storing the posts,
45       * enables searching for child comments.
46      * @return an arrayList containing comment BasePost objects
47      */
48
49      public ArrayList<BasePost> getComments(Platform socialPlatform) {
50
51          ArrayList<BasePost> comments = new ArrayList<BasePost>(); //Hold comment BasePost objects
52
53         for (int i=0; i < socialPlatform.getPosts().size(); i++) { //Iterate through all posts
54
55              BasePost iterationPost = socialPlatform.getPosts().get(i);
56
57              if (iterationPost instanceof CommentPost) {
58                  //Safely downcast generic BasePost to CommentPost and discard endorsements
59                  CommentPost iterationComment = (CommentPost)iterationPost;
60
61                  if (iterationComment.getParentID() == id) { //Comment is a child of this post
62                      comments.add(iterationComment); //Add comment to returned array
63                  }
64              }
65          }
66
67          return comments;
68      }
```

```
69
70      /**
71       * The method returns the posts endorsing a post, when provided the platform data.
72       * <p>
73       * @param socialPlatform - the Platform object storing the posts,
74        * enables searching for endorsements.
75       * @return an arrayList containing endorsement BasePost objects
76       */
77
78      public ArrayList<BasePost> getEndorsements(Platform socialPlatform) {
79
80          //Hold endorsement BasePost objects
81          ArrayList<BasePost> endorsements = new ArrayList<BasePost>();
82
83         for (int i=0; i < socialPlatform.getPosts().size(); i++) { //Iterate through all posts
84
85              BasePost iterationPost = socialPlatform.getPosts().get(i);
86
87              if (iterationPost instanceof EndorsementPost) {
88                  //Safely downcast generic BasePost to EndorsementPost and discard comments
89                  EndorsementPost iterationEndorsement = (EndorsementPost)iterationPost;
90
91                  if (iterationEndorsement.getParentID() == id) { //Endorsement is child of this post
92                      endorsements.add(iterationEndorsement); //Add endorsement to array
93                  }
94              }
95          }
96
97          return endorsements;
98      }
99
100     //BasePost Setter methods
101
102     public void setID(int id) {
103         this.id = id;
104     }
105
106     public void setPostType(int postType){
107         this.postType = postType;
108     }
109
110     public void setMessage(String message) {
111         this.message = message;
112     }
113
114     public void setAuthor(String author) {
115         this.author = author;
116     }
117
118     /**
119     * Generates a formatted string displaying the details of an Post instance.
120     * (the post's ID, it's author, the number of endorsements, the number of comments,
121     * and the post's message)
122     * <p>
123     * The number of endorsements and the number of comments are determined by finding
```

```
124     * the size of the return from the getEndorsements and getComments methods.
125     * Other post information is directly read from the instance attributes.
126     * <p>
127     * @return a formatted string containing the post details.
128     */
129
130     public String genPostDetails(Platform socialPlatform) {
131
132         String message = String.format("ID: %s \n" +
133       "Account: %s \n" +
134       "No. endorsements: %s | No. comments: %s \n" +
135       "%s",
136       id, author, getEndorsements(socialPlatform).size(),
137       getComments(socialPlatform).size(), this.message);
138
139         return message;
140     }
141
142 }
```

3 CommentPost.java

```
1  package socialmedia;
2
3  /**
4   * CommentPost provides attributes and methods to permit the creation of comments on the platform.
5   *
6   * @author Students: 720014004, 720033851
7   * @version 1.0
8   */
9
10 public class CommentPost extends BasePost {
11     private int parentID; //ID of parent post (standard/normal or comment type)
12
13     /**
14     * Constructor for a new CommentPost in the platform, setting handle, ID, parentID,
15     * and message instance attributes.
16     * <p>
17     * @param handle - the handle of the account authoring the comment
18     * @param ID - the comment's sequential ID
19     * @param parentID - the ID of the parent post, which this post is commenting on (a child of)
20     * @param message - the comment's message
21     */
22
23     public CommentPost(String handle, int ID, Integer parentID, String message) {
24         setID(ID); //Assign ID
25         setPostType(1); //Set post type to 'comment'
26         setMessage(message); //Set message to provided string
27         setAuthor(handle); //Set author to provided handle
28         this.parentID = parentID; //Set parentID attribute to ID of parent post
29     }
30
31     //ParentID instance attribute getter
32     public int getParentID() {
```

```
        return parentID;
    }

    //ParentID instance attribute setter
    public void setParentID(int parentID) {
        this.parentID = parentID;
    }
}
```

4 EndorsementPost.java

```
package socialmedia;

/**
 * EndorsementPost provides attributes and methods to permit the endorsement of posts
 * on the platform.
 * @author Students: 720014004, 720033851
 * @version 1.0
 */

public class EndorsementPost extends BasePost {
    private int parentID; //ID of parent post (standard/normal or comment type)

    /**
    * Constructor for a new EndorsementPost in the platform, intialising handle, ID, parentID,
    * parentHandle, and parentMessage instance attributes.
    * <p>
    * @param handle - the handle of the account authoring the endorsement
    * @param ID - the endorsement's sequential ID
    * @param parentID - the ID of the parent post, which this post is endorsing (a child of)
    * @param parentHandle - the handle of the parent post's author, which this post is endorsing
    * @param parentMessage - the parent post's message content
    */

    public EndorsementPost(String handle, int ID,
        Integer parentID, String parentHandle, String parentMessage) {

        setID(ID); //Assign sequential ID
        setPostType(2); //Set post type to 2 -'endorsement'

        //Generate endorsement message in required format
        setMessage(String.format("EP@%s: %s", parentHandle, parentMessage));
        setAuthor(handle); //Set the endorsement's author

        //Set the endorsement's parentID to the ID of the parent post/comment
        this.parentID = parentID;
    }

    //ParentID instance attribute getter
    public int getParentID() {
        return parentID;
    }

    //ParentID instance attribute setter
```

```
    public void setParentID(int parentID) {
        this.parentID = parentID;
    }
}
```

5 Platform.java

```
package socialmedia;
import java.io.Serializable;
import java.util.ArrayList;

/**
 * Platform stores all data relating to a specific social media platform instance.
 * This includes all accounts, posts, and counters.
 * When the social media platform is saved, the platform class can be serialised
 * to allow this data to persist.
 * <p>
 * The class also provides methods to retrieve platform statistics, in addition to
 * validity checks.
 * @author Students: 720014004, 720033851
 * @version 1.0
 */

public class Platform implements Serializable {

    //Accounts
    private ArrayList<Account> accounts = new ArrayList<Account>(); //Store account objects
    private int accountIDCounter = 0; //Sequentially count accounts, provide unique ID

    //Posts
    //Store BasePost objects (normal, comment, endorsement, posts)
    private ArrayList<BasePost> posts = new ArrayList<BasePost>();

    //Sequentially count accounts, provide unique ID.
    private int postIDCounter = 1; //Begin at 1, generic empty post assigned ID 0

    //Platform private instance attribute 'getter' methods


    /**
    * Getter method, returns the list of accounts in the platform.
    * <p>
    * @return Returns the arrayList containing the Account objects
    */

    public ArrayList<Account> getAccounts() {
        return accounts;
    }

    public int getAccountIDCounter() {
        return accountIDCounter;
    }

    /**
```

```
48      * Getter method, returns the list of posts (of all types) in the platform.
49      * <p>
50      * @return Returns the arrayList containing the BasePost objects
51      */
52
53      public ArrayList<BasePost> getPosts() {
54          return posts;
55      }
56
57      public int getPostIDCounter() {
58          return postIDCounter;
59      }
60
61      //Platform private instance attribute 'setter' methods
62
63
64      /**
65      * Setter method, increments the postIDCounter by 1.
66      */
67
68      public void incrementPostIDCounter() {
69          postIDCounter++;
70      }
71
72      /**
73      * Setter method, increments the accountIDCounter by 1.
74      */
75
76      public void incrementAccountIDCounter() {
77          accountIDCounter++;
78      }
79
80      /**
81      * Setter method, resets the accountIDCounter to 0.
82      */
83
84      public void resetAccountIDCounter() {
85          accountIDCounter = 0;
86      }
87
88
89      /**
90      * Setter method, resets the postIDCounter to 1.
91      */
92
93      public void resetPostIDCounter(){
94          postIDCounter = 1;
95      }
96
97
98      //Check methods
99
100     /**
101     * Method to check whether a handle is legal to create (does not exist)
102     * <p>
```

```
103     * @param handle - the handle to determine the legality of
104     * @return Returns <b>false</b> if a handle cannot be created (not unique),
105     * and <b>true</b> if it is legal to create.
106     */
107
108     public boolean checkHandleLegal(String handle) {
109         //Check if handle is an attribute in ArrayList of objects
110         for (int i=0; i < accounts.size(); i++) {
111             if (accounts.get(i).getHandle().equals(handle)) { //Handle already exists
112                 return false;
113             }
114         }
115         return true; //Handle does not exist
116     }
117
118
119     /**
120     * Method to check whether a handle is valid.
121     * A handle cannot be more than 30 characters, cannot
122     * contain whitespace, or be empty (invalid).
123     * <p>
124     * @param handle - the handle to determine the validity of
125     * @return Returns <b>false</b> if the handle is invalid,
126     * or <b>true</b> if the handle is valid.
127     */
128
129     public boolean checkHandleValid(String handle) {
130
131         //Check if handle is greater than 30 characters
132         if (handle.length() > 30) {
133             return false;
134         }
135
136         //Check if handle contains whitespace
137         else if (handle.contains(" ")) {
138             return false;
139         }
140
141         //Check if handle is empty
142         else if (handle.length() == 0) {
143             return false;
144         }
145
146         //Handle is valid
147         else {
148             return true;
149         }
150     }
151
152     /**
153     * Method to check whether a post message is valid.
154     * A post message cannot be more than 100 characters, and
155     * cannot be empty (invalid).
156     * <p>
157     * @param postMessage - the post message to determine the validity of
```

```
158     * @return Returns <b>false</b> if the post message is invalid,
159     * or <b>true</b> if the post message is valid.
160     */
161
162     public boolean checkPostValid(String postMessage) {
163
164         //Check if post is less than 100 characters
165         if(postMessage.length() > 100) {
166             return false;
167         }
168
169         //Check if post is empty
170         else if(postMessage.length() == 0) {
171             return false;
172         }
173
174         //Post is valid
175         else {
176             return true;
177         }
178     }
179
180     /**
181     * Method to check whether a post is actionable.
182     * An actionable post is either a normal post or a comment post
183     * (endorsement posts are non-actionable).
184     * <p>
185     * @param id - the ID of the post to check
186     * @return Returns <b>false</b> if the post is non-actionable,
187     * or <b>true</b> if the post is actionable.
188     */
189
190     public boolean checkPostActionable(int id) {
191
192         for (int i=0; i < posts.size(); i++) {
193             if (posts.get(i).getID() == id) { //Located post in list with matching ID
194                 //Normal/comment post
195                 if (posts.get(i).getPostType() == 0 || posts.get(i).getPostType() == 1) {
196                     return true;
197                 }
198             }
199         }
200         return false;
201     }
202
203
204     /**
205     * Method to check whether a post ID exists in the system.
206     * <p>
207     * @param id - the ID of the post to check
208     * @return Returns <b>false</b> if the ID exists in the system,
209     * or <b>true</b> if the ID does not exist in the system.
210     */
211
212     public boolean checkPostIDLegal(int id) {
```

```
213
214         //Check if post exists with matching 'id'attribute in ArrayList of objects
215         for (int i=0; i < posts.size(); i++) {
216             if (posts.get(i).getID() == id) { //ID already exists
217                 return false; //as ID exists
218             }
219         }
220         //Post ID not in system
221         return true;
222     }
223
224     //Platform getter methods
225
226     public int getNumberOfAccounts() {
227         return accounts.size();
228     }
229
230     /**
231     * Method to determine the number of original posts in the platform.
232     * Iterates over all posts, including them in the count provided they
233     * have type '0' (normal/original post).
234     * <p>
235     * @return the number of original posts in the platform
236     */
237
238     public int getTotalOriginalPosts() {
239         int numPosts = 0;
240
241         for (int i=0; i < posts.size(); i++) {
242             if (posts.get(i).getPostType() == 0) { //Original posts have type '0'
243                 numPosts++;
244             }
245         }
246         return numPosts;
247     }
248
249     /**
250     * Method to determine the number of endorsement posts in the platform.
251     * Iterates over all posts, including them in the count provided they
252     * have type '2' (endorsement post).
253     * <p>
254     * @return the number of endorsement posts in the platform
255     */
256
257     public int getTotalEndorsmentPost() {
258         int numEndorsements = 0;
259
260         for (int i=0; i < posts.size(); i++) {
261             if (posts.get(i).getPostType() == 2) { //Endorsements posts have type '2'
262             numEndorsements++;
263             }
264         }
265         return numEndorsements;
266     }
267
```

```
268     /**
269     * Method to determine the number of comment posts in the platform.
270     * Iterates over all posts, including them in the count provided they
271     * have type '1' (comment post).
272     * <p>
273     * @return the number of comment posts in the platform
274     */
275
276     public int getTotalCommentPosts() {
277         int numComments = 0;
278
279         for (int i=0; i < posts.size(); i++) {
280             if (posts.get(i).getPostType() == 1) { //Comment posts have type '1'
281             numComments++;
282             }
283         }
284         return numComments;
285     }
286
287     /**
288     * Method to determine the most endorsed post in the platform.
289     * <p>
290     * @param socialPlatform - the Platform object storing the posts,
291     * enables retrieving a post's endorsements (getEndorsements)
292     * @return the ID of the post in the platform with the most endorsements.
293     * In the case of a draw, the last post in the list is returned (highest ID).
294     * In the case there are no posts in the system, -1 is returned.
295     * In the case there are no endorsed posts, the most recent post (highest ID) is returned.
296     */
297
298     public int getMostEndorsedPost(Platform socialPlatform) {
299         int maxNumEndorsements = 0; //Counter for number of original posts
300         int idMaxPost = 0; //ID. of the post with max no. of endorsements
301
302         if (posts.size() == 1) { //No posts in the system (disregarding the generic empty post)
303             return -1; //Denotes error
304         }
305
306         for (int i=0; i < posts.size(); i++) {
307             //For each post, get no. of endorsements
308             int numEndorsements = posts.get(i).getEndorsements(socialPlatform).size();
309
310             //Post 'i' has current highest no. of endorsements
311             if (numEndorsements >= maxNumEndorsements) {
312             maxNumEndorsements = numEndorsements; //Update maximum recorded endorsements
313             idMaxPost = posts.get(i).getID(); //Update ID
314             }
315         }
316
317         assert(idMaxPost != 0): "0 is never a valid return (ID of generic empty post)";
318
319         return idMaxPost;
320     }
321
322
```

```
323     /**
324     * Method to determine the most endorsed account in the platform.
325     * <p>
326     * @param socialPlatform - the Platform object storing the posts,
327     * enables retrieving a post's endorsements (getEndorsements)
328     * @return the ID of the account in the platform with the most endorsements.
329     * In the case of a draw, the last account in the list is returned (highest ID).
330     * In the case there are no accounts in the system, -1 is returned.
331     * In the case there are no accounts with endorsements in the system, 0 is returned.
332     */
333
334     public int getMostEndorsedAccount(Platform socialPlatform) {
335         int idMaxAccountEndorsements = 0; //ID of account with most endorsements
336         int maxNumEndorsements = 0; //Counter
337
338         if (accounts.size() == 0) { //No accounts in the system
339             return -1; //Denotes error
340         }
341
342         for (int i=0; i < accounts.size(); i++) { //Iterate through accounts
343
344             String accountHandle = accounts.get(i).getHandle(); //Get account's handle
345             int accountID = accounts.get(i).getID(); //Get account ID
346
347             int accountEndorsements = 0; //Count an account's endorsements total
348
349             for (int j=1; j < posts.size(); j++) { //Iterate through posts
350
351                 if (posts.get(j).getAuthor().equals(accountHandle)) { //Post authored by account
352                     //Add post's no. of endorsements to account's endorsements total
353                     accountEndorsements = accountEndorsements +
354                         posts.get(j).getEndorsements(socialPlatform).size();
355                 }
356             }
357
358             //The current account's no. endorsements is greater than maxNumEndorsements
359             if (accountEndorsements > maxNumEndorsements) {
360                 maxNumEndorsements = accountEndorsements; //Update values
361                 idMaxAccountEndorsements = accountID;
362             }
363         }
364         return idMaxAccountEndorsements;
365     }
366 }
```

6 Post.java

```
1  package socialmedia;
2
3  /**
4   * Post provides constructors to permit the creation of standard posts on the platform,
5   * in addition to a no-arg contstructor permitting the creation of the generic empty post.
6   * @author Students: 720014004, 720033851
7   * @version 1.0
```

```
 */

public class Post extends BasePost {

    /**
    * Constructor for the generic empty post, always assigning ID 0, type '3' (empty post),
    * and the provided empty post message.
    * <p>
    * When a post with comments is deleted, the parentID attributes of its comments are set to 0,
    * referring them to this post.
    */

    public Post() {
        setID(0); //Assign ID 0
        setPostType(3); //Set post type to 3 - 'generic empty post'

        //Provided empty post message
        setMessage("The original content was removed from the system and is no longer available.");
    }

    /**
    * Constructor for a new standard Post in the platform, initialising
    * handle, ID, and message instance attributes.
    * <p>
    * @param handle - the handle of the account authoring the post
    * @param id - the post's sequential ID
    * @param message - the post's message
    */

    public Post(String handle, int id, String message) {
        setID(id); //Assign ID
        setPostType(0); //Set post type to 'normal'
        setMessage(message); //Set message to provided string
        setAuthor(handle); //Set author to provided handle
    }
}
```

7 SocialMedia.java

```
package socialmedia;
import java.util.ArrayList;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 * SocialMedia is an implementor of the SocialMediaPlatform interface.
 *
 * @author Students: 720014004, 720033851
 * @version 1.0
 */

```

```
16 public class SocialMedia implements SocialMediaPlatform {
17
18    //Hold platform instance
19    private Platform socialPlatform;
20
21    /**
22     * No-arg constructor for the initialisation of an empty platform,
23     * where the Platform object 'socialPlatform' and generic empty post are created.
24     */
25
26    public SocialMedia() {
27       socialPlatform = new Platform(); //Create platform instance
28       BasePost genericEmptyPost = new Post(); //Create generic empty post
29       socialPlatform.getPosts().add(genericEmptyPost); //Add generic empty post to array
30
31       assert(socialPlatform.getPosts().size() == 1):
32       "The generic empty post was not correctly created";
33    }
34
35    @Override
36    public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
37
38       //Check if handle is legal (not pre-existing)
39         if (socialPlatform.checkHandleLegal(handle) == false) {
40             throw new IllegalHandleException("This handle already exists in the system");
41         }
42
43         // Check if handle is valid (less than 30 characters, no whitespace and not empty)
44         if (socialPlatform.checkHandleValid(handle) == false) {
45             throw new InvalidHandleException
46             ("Your handle must be valid (less than 30 characters, 0 whitespace, not empty).");
47         }
48
49       int accountIDCounter = socialPlatform.getAccountIDCounter(); //Get account ID counter
50
51       Account platformUser = new Account(handle, accountIDCounter); //Create Account, set ID
52       socialPlatform.getAccounts().add(platformUser); //Store Account object
53       socialPlatform.incrementAccountIDCounter(); //Increment counter
54
55       assert(socialPlatform.getAccounts().size() != 0 && socialPlatform.getAccountIDCounter() > 0):
56        "An account has been created but not stored or the accounts counter was not incremented";
57
58       return platformUser.getID(); //Return ID of generated account
59    }
60
61    @Override
62    public int createAccount(String handle, String description) throws IllegalHandleException,
          InvalidHandleException {
63
64         //Check if handle is legal (not pre-existing)
65         if (socialPlatform.checkHandleLegal(handle) == false) {
66             throw new IllegalHandleException
67          ("This handle already exists in the system, please choose another.");
68         }
69
```

```
70          // Check if handle is valid (less than 30 characters, no whitespace and not empty)
71          if (socialPlatform.checkHandleValid(handle) == false) {
72              throw new InvalidHandleException
73              ("Your handle must be valid (less than 30 characters, 0 whitespace, not empty).");
74          }
75
76        int accountIDCounter = socialPlatform.getAccountIDCounter(); //Get account ID counter
77        //Create account (with decription), set ID
78        Account platformUser = new Account(handle, accountIDCounter, description);
79        socialPlatform.getAccounts().add(platformUser); //Store account object
80        socialPlatform.incrementAccountIDCounter(); //Increment counter
81
82        assert(socialPlatform.getAccounts().size() != 0 && socialPlatform.getAccountIDCounter() > 0):
83         "An account has been created but not stored or the accounts counter was not incremented";
84
85        return platformUser.getID(); //Return 'id' of generated account
86    }
87
88    /**
89     * Method for the reloading of a specified account (via its handle),
90    * returning the Account object.
91    * <p>
92    * @param handle - the handle of the Account to reload
93    * @return the Account object with the specified handle
94    * @throws HandleNotRecognisedException thrown when attempting to reload an account where the
95    * provided handle does not exist
96     */
97
98    public Account reloadAccount(String handle) throws HandleNotRecognisedException {
99
100       for (int i=0; i < socialPlatform.getAccounts().size(); i++) {
101            if (socialPlatform.getAccounts().get(i).getHandle().equals(handle)) { //Matching ID
102            Account reloadedAccount = socialPlatform.getAccounts().get(i);
103            return reloadedAccount;
104            }
105         }
106
107       //Account with matching handle not found, cannot exist
108       throw new HandleNotRecognisedException
109       ("An account with this handle does not exist in the system.");
110   }
111
112   /**
113    * Method for the reloading of a specified account (via its ID),
114   * returning the Account object.
115   * <p>
116   * @param id - the ID of the Account to reload
117   * @return the Account object with the specified ID
118   * @throws AccountIDNotRecognisedException thrown when attempting to reload an account where the
119   * provided ID does not exist
120    */
121
122   public Account reloadAccount(int id) throws AccountIDNotRecognisedException {
123
124       for (int i=0; i < socialPlatform.getAccounts().size(); i++) {
```

```
125             if (socialPlatform.getAccounts().get(i).getID() == id) { //Matching ID
126             Account reloadedAccount = socialPlatform.getAccounts().get(i);
127             return reloadedAccount;
128             }
129         }
130 
131        throw new AccountIDNotRecognisedException
132        ("An account with this ID does not exist in the system.");
133     }
134 
135     @Override
136     public void removeAccount(int id) throws AccountIDNotRecognisedException {
137 
138        Account reloadedAccount = reloadAccount(id); //Reload account (to be deleted), by ID
139 
140         ArrayList<Integer> delPostIDs = new ArrayList<Integer>(); //Store ID's of posts to delete
141 
142         for (int j=1; j < socialPlatform.getPosts().size(); j++) {
143 
144          //Check if post authored by account being deleted
145             if (socialPlatform.getPosts().get(j).getAuthor().equals(reloadedAccount.getHandle())) {
146                 int postID = socialPlatform.getPosts().get(j).getID(); //Get ID of post (to delete)
147                 delPostIDs.add(postID);
148             }
149         }
150 
151         for (int j=0; j < delPostIDs.size(); j++) {
152             try {
153                 deletePost(delPostIDs.get(j)); //Attempt to delete post
154 
155             } catch (PostIDNotRecognisedException e) {
156             /* Fine to ignore this exception. In this case, it is acceptable to throw a
157              * PostIDNotRecognisedException as it is being thrown due to the post already
158              * having been deleted as a child of another post in delPostIDs.
159              * The post ID is certain to be valid as it was retrieved within the method.*/
160             }
161         }
162 
163        socialPlatform.getAccounts().remove(reloadedAccount); //Remove account
164 
165     }
166 
167     @Override
168     public void removeAccount(String handle) throws HandleNotRecognisedException {
169        Account reloadedAccount = reloadAccount(handle); //Reload account (to be deleted)
170 
171         ArrayList<Integer> delPostIDs = new ArrayList<Integer>(); //Store ID's of posts (to delete)
172 
173         for (int j=1; j < socialPlatform.getPosts().size(); j++) {
174 
175          //Check if post authored by account being deleted
176             if (socialPlatform.getPosts().get(j).getAuthor().equals(handle)) {
177                 int postID = socialPlatform.getPosts().get(j).getID(); //Get ID of post (to delete)
178                 delPostIDs.add(postID); //Add to list of ID's to delete
179 
```

```
180             }
181         }
182 
183         for (int j=0; j < delPostIDs.size(); j++) {
184             try {
185                 deletePost(delPostIDs.get(j)); //Attempt to delete post
186 
187             } catch (PostIDNotRecognisedException e) { //Post already deleted (as a child)
188             }
189         }
190 
191     socialPlatform.getAccounts().remove(reloadedAccount); //Remove account
192 }
193 
194 @Override
195 public void changeAccountHandle(String oldHandle, String newHandle)
196       throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
197 
198     //Check author's handle exists
199     if (socialPlatform.checkHandleLegal(newHandle) == false) {
200           throw new IllegalHandleException
201        ("This handle already exists in the system, please choose another.");
202       }
203 
204     // Check if handle is valid (less than 30 characters, no whitespace and not empty)
205       if (socialPlatform.checkHandleValid(newHandle) == false) {
206           throw new InvalidHandleException
207        ("Ensure your handle is valid (less than 30 characters, 0 whitespace, not empty).");
208       }
209 
210 
211     Account platformUserReload = reloadAccount(oldHandle); //Reload account (to be deleted)
212     platformUserReload.setHandle(newHandle); //Change the handle to the new specified one
213 
214     //Assert handle not identical to previous, and equal to newHandle when retrieved
215     assert(platformUserReload.getHandle() != oldHandle && platformUserReload.getHandle() == newHandle):
216     "The account handle was not correctly changed";
217 
218 }
219 
220 @Override
221 public void updateAccountDescription(String handle, String description) throws
        HandleNotRecognisedException {
222 
223     Account platformUserReload = reloadAccount(handle); //Reload account (to be deleted)
224     platformUserReload.setDescription(description); //Update description attribute
225 
226     assert(platformUserReload.getDescription() == description):
227        "The description was not correctly changed";
228 }
229 
230 @Override
231 public String showAccount(String handle) throws HandleNotRecognisedException {
232 
233     Account reloadedAccount = reloadAccount(handle); //Reload account (to be deleted)
```

```
234
235         //Counters for posts and endorsements
236           int postCounter = 0;
237           int endorsedCounter = 0;
238
239         //Get no. of posts (authored by account)
240           for (int j=1; j < socialPlatform.getPosts().size(); j++) {
241               if (socialPlatform.getPosts().get(j).getAuthor().equals(handle)) {
242                   postCounter++; //Increment the post counter
243                   endorsedCounter = endorsedCounter + socialPlatform.getPosts().get(j)
244                  .getEndorsements(socialPlatform).size(); //Count endorsement posts
245               }
246           }
247
248         //Call the helper function to generate account details and return
249         return reloadedAccount.generateAccountDetails(postCounter, endorsedCounter);
250     }
251
252     @Override
253     public int createPost(String handle, String message) throws HandleNotRecognisedException,
          InvalidPostException {
254
255         //Check author's handle exists
256         if (socialPlatform.checkHandleLegal(handle) == true) {
257             throw new HandleNotRecognisedException("This handle does not exist in the system!");
258         }
259
260         //Check post is valid (less than 100 characters and not empty)
261           if (socialPlatform.checkPostValid(message) == false) {
262               throw new
263               InvalidPostException("Your post must be valid (less than 100 chars, not empty)");
264           }
265
266         int postIDCounter = socialPlatform.getPostIDCounter(); //Get the Post ID Counter
267
268           BasePost platformPost = new Post(handle, postIDCounter, message); //Create post, initialise
269           socialPlatform.getPosts().add(platformPost); //Save post
270
271         socialPlatform.incrementPostIDCounter(); //Increment counter
272
273         assert(socialPlatform.getPosts().size() > 1 && socialPlatform.getPostIDCounter() > 1):
274         "An post has been created but not stored or the posts counter was not incremented";
275
276         return platformPost.getID(); //Return ID of post
277     }
278
279     /**
280      * Method for the reloading of a specified post (via its ID),
281     * returning the BasePost object.
282     * <p>
283     * @param id - the ID of the Post to reload
284     * @return the Post object with matching ID
285     * @throws PostIDNotRecognisedException thrown when attempting to reload an post where the
286     * provided ID does not exist
287      */
```

```
288
289     public BasePost reloadPost(int id) throws PostIDNotRecognisedException {
290
291        for (int i=1; i < socialPlatform.getPosts().size(); i++) {
292             if (socialPlatform.getPosts().get(i).getID() == id) { //Post has matching ID
293
294             BasePost reloadedPost = socialPlatform.getPosts().get(i);
295             return reloadedPost; //Return BasePost object
296
297             }
298          }
299
300        throw new PostIDNotRecognisedException("A post with this ID does not exist in the system");
301     }
302
303     @Override
304     public int endorsePost(String handle, int id)
305          throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException {
306
307        BasePost reloadedParentPost = reloadPost(id); //Reload parent
308
309        //Check endorsement author's handle exists
310          if (socialPlatform.checkHandleLegal(handle) == true) {
311             throw new HandleNotRecognisedException("This handle does not exist in the system.");
312          }
313
314          //Check parent actionable (not endorsement/empty)
315          if (socialPlatform.checkPostActionable(id) == false) {
316             throw new
317             NotActionablePostException("The parent is not actionable, so cannot be endorsed");
318          }
319
320        int postIDCounter = socialPlatform.getPostIDCounter(); //Get the Post ID Counter
321
322        BasePost platformEndorsement = new EndorsementPost(handle, postIDCounter, id,
323           reloadedParentPost.getAuthor(), reloadedParentPost.getMessage());
324              //Construct an endorsement post object
325          socialPlatform.getPosts().add(platformEndorsement); //Save post
326
327        socialPlatform.incrementPostIDCounter(); //Increment counter
328
329        assert(socialPlatform.getPosts().size() > 2 && socialPlatform.getPostIDCounter() > 2):
330        "An endorsement has been created but not stored or the post counter was not incremented";
331
332        return platformEndorsement.getID(); //Return the ID of the created endorsement post
333     }
334
335     @Override
336     public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
337          PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
338
339        //Check comment author's handle exists
340          if (socialPlatform.checkHandleLegal(handle) == true) {
341             throw new HandleNotRecognisedException("This handle does not exist in the system!");
342          }
```

```
343
344         //Check parent exists
345           if (socialPlatform.checkPostIDLegal(id) == true) {
346               throw new PostIDNotRecognisedException("The parent post does not exist in the system");
347           }
348
349           //Check parent actionable (not endorsement/empty, can comment)
350           if (socialPlatform.checkPostActionable(id) == false) {
351               throw new NotActionablePostException
352               ("The parent post is not actionable, and cannot be commented");
353           }
354
355           //Check post is valid (less than 100 characters and not empty)
356           if (socialPlatform.checkPostValid(message) == false) {
357               throw new
358               InvalidPostException("Ensure our post is valid (less than 100 chars, not empty)");
359           }
360
361         int postIDCounter = socialPlatform.getPostIDCounter(); //Get the Post ID Counter
362
363         //Construct a comment post object
364         BasePost platformComment = new CommentPost(handle, postIDCounter, id, message);
365           socialPlatform.getPosts().add(platformComment); //Save post
366
367         socialPlatform.incrementPostIDCounter(); //Increment counter
368
369         assert(socialPlatform.getPosts().size() > 2 && socialPlatform.getPostIDCounter() > 2):
370         "An comment has been created but not stored or the post counter was not incremented";
371
372         return platformComment.getID(); //Return the ID of the created comment post
373     }
374
375     @Override
376     public void deletePost(int id) throws PostIDNotRecognisedException {
377
378         BasePost reloadedPost = reloadPost(id); //Load post (to be deleted)
379
380         //Delete endorsements
381         if (reloadedPost.getEndorsements(socialPlatform).size() != 0) { //Post has endorsements
382
383             //Iterate through endorsements
384             for (int i=0; i < reloadedPost.getEndorsements(socialPlatform).size(); i++) {
385                 //Get orphan endorsement
386                 BasePost reloadOrpEndorsement = reloadedPost.getEndorsements(socialPlatform).get(i);
387                 socialPlatform.getPosts().remove(reloadOrpEndorsement); //Remove endorsement
388             }
389         }
390
391         //Update commments
392         if (reloadedPost.getComments(socialPlatform).size() != 0) { //Post has comments
393             for (int i=0; i < reloadedPost.getComments(socialPlatform).size(); i++) { //Iterate through comments
394                 BasePost reloadOrpComment = reloadedPost.getComments(socialPlatform).get(i); //Get orphan comment
395
396                 if (reloadOrpComment instanceof CommentPost) {
397                     CommentPost relComment = (CommentPost)reloadOrpComment;
```

```
398
399             //Update comment parentID to generic empty post's ID (0)
400             relComment.setParentID(0);
401
402             assert(relComment.getParentID() == 0): "The orphan parentID was not set to 0";
403         }
404       }
405     }
406
407     socialPlatform.getPosts().remove(reloadedPost); //Delete post
408   }
409
410   @Override
411   public String showIndividualPost(int id) throws PostIDNotRecognisedException {
412
413     BasePost reloadedPost = reloadPost(id); //Load post (to be deleted)
414
415     //Call the helper function to generate post details and return
416       return reloadedPost.genPostDetails(socialPlatform);
417   }
418
419   @Override
420   public StringBuilder showPostChildrenDetails(int id)
421        throws PostIDNotRecognisedException, NotActionablePostException {
422
423     BasePost reloadedParentPost = reloadPost(id); //Reload the parent post
424
425       //Check if parent is actionable
426       if (socialPlatform.checkPostActionable(id) == false) {
427           throw new NotActionablePostException("The parent post is not actionable");
428       }
429
430     //Execute helper function, pass in parent post
431     StringBuilder returnedPostDetails = helpShowPostChildrenDetails(reloadedParentPost);
432
433       return returnedPostDetails; //Return StringBuilder
434    }
435
436   /**
437    * Private helper method for showPostChildrenDetails.
438   * Removes the requirement to repeat reduntant checks (comments always actionable).
439   * <p>
440   * @param postObj - the BasePost object (for which the childen details are displayed)
441   * @return the stringBuilder containing the post details in required 'tree' format
442   * @throws PostIDNotRecognisedException thrown when attempting to reload an post that
443   * does not exist
444    */
445
446   private StringBuilder helpShowPostChildrenDetails(BasePost postObj) throws PostIDNotRecognisedException {
447
448     StringBuilder postDetails = new StringBuilder(); //Create stringBuilder
449
450     if (postObj.getComments(socialPlatform).size() != 0) { //Check if post has comments
451
452       //Add post details (with required formatting) to stringBuilder
```

```
453         postDetails.append(showIndividualPost(postObj.getID()) + System.lineSeparator() + "|");
454
455         //Iterate through comments, recursively calling the helper on each
456         for (int j=0; j < postObj.getComments(socialPlatform).size(); j++) {
457
458            //Recursive call, convert returned stringbuilder to string
459            String childString = helpShowPostChildrenDetails
460               (postObj.getComments(socialPlatform).get(j)).toString();
461
462            //Split the recursive return into lines (at newline character)
463            String linesSplit[] = childString.split(System.lineSeparator());
464
465            boolean firstLine = true;
466            for (int k=0; k < linesSplit.length; k++) {
467               if (firstLine == true) {
468                  //Add correct formatting to first line
469                  postDetails.append(System.lineSeparator() + "| > "
470                     + linesSplit[k] + System.lineSeparator());
471                  firstLine = false; //First line formatting completed, do not re-run
472               } else {
473                  //Pad all other lines with 4 chars of whitespace
474                  postDetails.append("  " + linesSplit[k] + System.lineSeparator());
475               }
476            }
477         }
478
479      } else {
480         //Add details of parent post to stringBulder
481         postDetails.append(showIndividualPost(postObj.getID()));
482      }
483
484      return postDetails;
485   }
486
487   @Override
488   public int getNumberOfAccounts() {
489      return socialPlatform.getNumberOfAccounts();
490   }
491
492   @Override
493   public int getTotalOriginalPosts() {
494      return socialPlatform.getTotalOriginalPosts();
495   }
496
497   @Override
498   public int getTotalEndorsmentPosts() {
499      return socialPlatform.getTotalEndorsmentPost();
500   }
501
502   @Override
503   public int getTotalCommentPosts() {
504      return socialPlatform.getTotalCommentPosts();
505   }
506
507   @Override
```

```
508     public int getMostEndorsedPost() {
509        return socialPlatform.getMostEndorsedPost(socialPlatform);
510     }
511
512     @Override
513     public int getMostEndorsedAccount() {
514        return socialPlatform.getMostEndorsedAccount(socialPlatform);
515     }
516
517     @Override
518     public void erasePlatform() {
519         //Accounts arrayList/counters
520         socialPlatform.getAccounts().clear(); //Clear the accounts array
521         socialPlatform.resetAccountIDCounter(); //Reset Account's ID counter
522
523         //Posts arrayList/counters
524         socialPlatform.getPosts().clear(); //Clear the posts array
525         socialPlatform.resetPostIDCounter(); //Reset Post's ID counter
526
527        BasePost genericEmptyPost = new Post(); //Recreate generic empty post
528        socialPlatform.getPosts().add(genericEmptyPost); //Add generic empty post to array
529
530        assert(socialPlatform.getAccounts().size() == 0 &&
531           socialPlatform.getAccountIDCounter() == 0 && socialPlatform.getPosts().size() == 1
532              && socialPlatform.getPostIDCounter() == 1): "The platform contents/counters were not correctly
                   reset";
533
534     }
535
536     @Override
537     public void savePlatform(String filename) throws IOException {
538        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename));
539        oos.writeObject(socialPlatform); //Write object to file
540        oos.close();
541     }
542
543     @Override
544     public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
545        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename));
546        Object obj = ois.readObject(); //Write platform object
547        ois.close();
548
549        //Safely downcast (if deserialised obj is a 'Platform' instance)
550        if (obj instanceof Platform) {
551           socialPlatform = (Platform) obj; //Overwrite socialPlatform with deserialised object
552
553           assert(socialPlatform.getPosts().size() > 0):
554           "The re-loaded platform does not contain expected data"; //Generic empty post missing
555        }
556     }
557  }
```