1 Account.java

```java
package socialmedia;

import java.util.ArrayList;
import java.io.Serializable;

/**
 * Account class creates a social media account with a unique ID and saves it to
 * the system
 */
public class Account implements Serializable {
    private int id;
    private String handle;
    private String description;
    private int numberOfEndorsements = 0;
    private int numberOfComments = 0;
    private ArrayList<Post> posts = new ArrayList<Post>();
    private static ArrayList<String> handles = new ArrayList<String>();
    private static int idCount = 10000000;
    private static final long serialVersionUID = 1L;

    /**
     * Creates a social media account object with a specified handle and a unique ID
     *
     * @param handle the social media handle, already checked for uniqueness
     */
    public Account(String handle) {

        this.handle = handle;
        handles.add(handle);
        id = idCount;
        idCount += 1;

    }

    public int getId() {
        return id;
    }

    public String getHandle() {
        return handle;
    }

    public void setHandle(String handle) {
        this.handle = handle;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
```

```
    }

    /**
     * Checks if the handle is already in the system
     */
    public static Boolean isHandleUnique(String handle) {
        return !(handles.contains(handle));
    }

    /**
     * Checks the handle is not empty, longer than 30 characters or contains
     * whitespace
     *
     * @param handle the handle being checked
     * @return true if the handle is valid, false otherwise
     */
    public static Boolean isHandleValid(String handle) {
        if (handle.equals("")) {
            return false;
        }
        if (handle.length() > 30) {
            return false;
        }
        if (handle.contains(" ")) {
            return false;
        }

        return true;
    }

    public ArrayList<Post> getPosts() {
        return posts;
    }

    public void setPosts(ArrayList<Post> posts) {
        this.posts = posts;
    }

    public void addPostToAccount(Post p) {
        posts.add(p);
    }

    /**
     * Removes a post from the account
     *
     * @param p the post being removed
     */
    public void removePostFromAccount(Post p) {
        if (posts.contains(p)) {
            posts.remove(p);
        }

    }

    /**
```

```
108      * Increments the number of endorsements associated with this account by 1
109      */
110     public void incrementEndorsements() {
111         numberOfEndorsements += 1;
112     }
113 
114     /**
115      * Decrements the number of endorsements associated with this account by 1
116      */
117     public void decrementEndorsements() {
118         numberOfEndorsements -= 1;
119     }
120 
121     /**
122      * Increments the number of comments associated with this account by 1
123      */
124     public void incrementComments() {
125         numberOfComments += 1;
126     }
127 
128     /**
129      * Decrements the number of endorsements associated with this account by 1
130      */
131     public void decrementComments() {
132         numberOfComments -= 1;
133     }
134 
135     public int getNumberOfEndorsements() {
136         return numberOfEndorsements;
137     }
138 
139     public int getNumberOfComments() {
140         return numberOfComments;
141     }
142 
143     public static int getIdCount() {
144         return idCount;
145     }
146 
147     public static void setIdCount(int idCount) {
148         Account.idCount = idCount;
149     }
150 
151 }
```

2 Post.java

```
1  package socialmedia;
2  
3  import java.util.ArrayList;
4  import java.io.Serializable;
5  
6  /**
7   * Post class creates a social media post
```

```
 */
public class Post implements Serializable {

    private static final long serialVersionUID = 1L;
    protected int postID;
    protected Account author;
    protected String message;
    protected int numberOfEndorsements = 0;
    protected int numberOfComments = 0;
    protected ArrayList<Post> replies = new ArrayList<Post>();
    private static int chronologicalId = 0;

    /**
     * Creates a Post Object with a unique ID
     *
     * @param author the author of the post
     * @param message the post message body
     */
    public Post(Account author, String message) {
        this.author = author;
        this.message = message;
        postID = chronologicalId;
        incrementId();
    }

    public int getPostID() {
        return postID;
    }

    public Account getAuthor() {
        return author;
    }

    public void setAuthor(Account author) {
        this.author = author;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public ArrayList<Post> getReplies() {
        return replies;
    }

    public void setReplies(ArrayList<Post> replies) {
        this.replies = replies;
    }

    /**
     * Increments the static ID for the Post class so that post has a unique,
```

```
63          * chronolgical ID
64          */
65         private static void incrementId() {
66             chronologicalId += 1;
67         }
68
69         public void addReply(Post post) {
70             replies.add(post);
71         }
72
73         public void removeReply(Post post) {
74             replies.remove(post);
75         }
76
77         /**
78          * Increments the number of endorsements associated with this Post by 1
79          */
80         public void incrementEndorsements() {
81             numberOfEndorsements += 1;
82         }
83
84         /**
85          * Decrements the number of endorsements associated with this Post by 1
86          */
87         public void decrementEndorsements() {
88             numberOfEndorsements -= 1;
89         }
90
91         /**
92          * Increments the number of Comments associated with this Post by 1
93          */
94         public void incrementComments() {
95             numberOfComments += 1;
96         }
97
98         /**
99          * Decrements the number of Comments associated with this Post by 1
100         */
101        public void decrementComments() {
102            numberOfComments -= 1;
103        }
104
105        public int getNumberOfEndorsements() {
106            return numberOfEndorsements;
107        }
108
109        public int getNumberOfComments() {
110            return numberOfComments;
111        }
112
113        public static int getChronologicalId() {
114            return chronologicalId;
115        }
116
117        public static void setChronologicalId(int chronologicalId) {
```

```
        Post.chronologicalId = chronologicalId;
    }

}
```

3 Comment.java

```
package socialmedia;

/**
 * Comment class creates a comment object
 */
public class Comment extends Post {
    private static final long serialVersionUID = 1L;
    private Post replyingTo;

    /**
     * Creates a Comment object extending a Post object
     *
     * @param author    the author of the comment
     * @param message   the message body of the comment
     * @param replyingTo the post being commented on
     */
    public Comment(Account author, String message, Post replyingTo) {
        super(author, message);
        this.replyingTo = replyingTo;
    }

    public Post getReplyingTo() {
        return replyingTo;
    }

    public void setReplyingTo(Post replyingTo) {
        this.replyingTo = replyingTo;
    }

}
```

4 Endorsement.java

```
package socialmedia;

/**
 * Endorsement class to create an Endorsement object
 */
public class Endorsement extends Post {
    private static final long serialVersionUID = 1L;
    Post refersTo;

    /**
     * Creates an Endorsement object extending a Post object with a predefined
     * message
     *
     * @param author the author of the endorsement
```

```
     * @param refersTo the post being endorsed
     */
    public Endorsement(Account author, Post refersTo) {
        super(author, "EP@" + author.getHandle() + ": " + refersTo.getMessage());
        this.refersTo = refersTo;
    }

    public Post getRefersTo() {
        return refersTo;
    }

    public void setRefersTo(Post refersTo) {
        this.refersTo = refersTo;
    }

}
```

5 GenericEmptyPost.java

```
package socialmedia;

/**
 * Generic Empty Post class creates a placeholder post
 */
public class GenericEmptyPost extends Post {

    private static final long serialVersionUID = 1L;
    private Post replyingTo = null;

    /**
     * Creates a GenericEmptyPost object extending a Post object as a placeholder
     *
     * @param originalPost the post being replaced
     */
    public GenericEmptyPost(Post originalPost) {
        super(null, "The original content was removed from the system and is no longer available.");
        if (originalPost instanceof Comment) {
            replyingTo = ((Comment) originalPost).getReplyingTo();
        }
        this.replies = originalPost.getReplies();
    }

    public int getPostID() {
        return postID;
    }

    public String getMessage() {
        return message;
    }

    public Post getReplyingTo() {
        return replyingTo;
    }

```

```
}
```

6 AccountIDNotRecognisedException.java

```
package socialmedia;

/**
 * Exception: if the Account ID is not in the system
 */
public class AccountIDNotRecognisedException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public AccountIDNotRecognisedException(String errorMessage) {
        super(errorMessage);
    }
}
```

7 HandleNotRecognisedException.java

```
package socialmedia;

/**
 * Exception: if the handle is not in the system
 */
public class HandleNotRecognisedException extends RuntimeException {
  private static final long serialVersionUID = 1L;

  public HandleNotRecognisedException(String errorMessage) {
    super(errorMessage);
  }
}
```

8 IllegalHandleException.java

```
package socialmedia;

/**
 * Exception: If the new handle already exists in the platform.
 */
public class IllegalHandleException extends RuntimeException {

  private static final long serialVersionUID = 1L;

  public IllegalHandleException(String errorMessage) {
    super(errorMessage);
  }
}
```

9 InvalidHandleException.java

```
package socialmedia;

```

```java
/**
 * Exception: if the handle is empty, has more than 30 characters, or has white
 * spaces.
 */
public class InvalidHandleException extends RuntimeException {
  private static final long serialVersionUID = 1L;

  public InvalidHandleException(String errorMessage) {
    super(errorMessage);
  }
}
```

10 InvalidPostException.java

```java
package socialmedia;

/**
 * Exception: if the Post message is empty, has more than 100 characters.
 */
public class InvalidPostException extends RuntimeException {
  private static final long serialVersionUID = 1L;

  public InvalidPostException(String errorMessage) {
    super(errorMessage);
  }
}
```

11 NotActionablePostException.java

```java
package socialmedia;

/**
 * Exception: if trying to endorse or comment on an endorsement post
 */
public class NotActionablePostException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public NotActionablePostException(String errorMessage) {
        super(errorMessage);
    }
}
```

12 PostIDNotRecognisedException.java

```java
package socialmedia;

/**
 * Exception: if the Post ID is not on the system.
 */
public class PostIDNotRecognisedException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public PostIDNotRecognisedException(String errorMessage) {
```

```
        super(errorMessage);
    }
}
```

13 SocialMedia.java

```
package socialmedia;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

/**
 * SocialMedia is an implemetor of the SocialMediaPlatform interface
 */
public class SocialMedia implements SocialMediaPlatform {

   private static final long serialVersionUID = 1L;
   private static ArrayList<Account> accounts = new ArrayList<Account>();
   private static ArrayList<Post> posts = new ArrayList<Post>();

   /**
    * The method creates an account in the platform with the given handle.
    * <p>
    * The state of this SocialMediaPlatform must be be unchanged if any exceptions
    * are thrown.
    *
    * @param handle account's handle.
    * @throws IllegalHandleException if the handle already exists in the platform.
    * @throws InvalidHandleException if the new handle is empty, has more than 30
    *                                characters, or has white spaces.
    * @return the ID of the created account.
    *
    */
   public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
      int id = 0;

      checkAccountHandle(handle);
      Account account = new Account(handle);
      accounts.add(account);
      id = account.getId();
      return id;
   }

   /**
    * The method creates an account in the platform with the given handle and
    * description.
    * <p>
    * The state of this SocialMediaPlatform must be be unchanged if any exceptions
    * are thrown.
```

```
49      *
50      * @param handle    account's handle.
51      * @param description account's description.
52      * @throws IllegalHandleException if the handle already exists in the platform.
53      * @throws InvalidHandleException if the new handle is empty, has more than 30
54      *                                characters, or has white spaces.
55      * @return the ID of the created account.
56      */
57     public int createAccount(String handle, String description) throws IllegalHandleException,
            InvalidHandleException {
58
59        checkAccountHandle(handle);
60        Account account = new Account(handle);
61        account.setDescription(description);
62        accounts.add(account);
63        return account.getId();
64
65     }
66
67     /**
68      * The method removes the account with the corresponding ID from the platform.
69      * When an account is removed, all of their posts and likes should also be
70      * removed.
71      * <p>
72      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
73      * are thrown.
74      *
75      * @param id ID of the account.
76      * @throws AccountIDNotRecognisedException if the ID does not match to any
77      *                                         account in the system.
78      */
79     public void removeAccount(int id) throws AccountIDNotRecognisedException {
80        Boolean accountIdRecognised = false;
81        Account removeAccount = null;
82
83        for (Account account : accounts) {
84           if (account.getId() == id) {
85              removeAccount = account;
86              accountIdRecognised = true;
87              break;
88           }
89        }
90        accounts.remove(removeAccount);
91        if (!accountIdRecognised) {
92           throw new AccountIDNotRecognisedException("The account ID " + id + " has not been recognised.");
93        }
94
95        for (Post post : posts) {
96           if (post.getAuthor() == removeAccount) {
97
98              deletePost(post.getPostID());
99
100          }
101       }
102
```

```
103     }
104
105     /**
106      * The method removes the account with the corresponding handle from the
107      * platform. When an account is removed, all of their posts and likes should
108      * also be removed.
109      * <p>
110      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
111      * are thrown.
112      *
113      * @param handle account's handle.
114      * @throws HandleNotRecognisedException if the handle does not match to any
115      *                                      account in the system.
116      */
117     public void removeAccount(String handle) throws HandleNotRecognisedException {
118
119        Account a = findAccountFromHandle(handle);
120        accounts.remove(a);
121        for (Post post : posts) {
122           if (post.getAuthor() == a) {
123              try {
124                 deletePost(post.getPostID());
125              } catch (PostIDNotRecognisedException e) {
126                 e.printStackTrace();
127              }
128           }
129        }
130     }
131
132     /**
133      * The method replaces the oldHandle of an account by the newHandle.
134      * <p>
135      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
136      * are thrown.
137      *
138      * @param oldHandle account's old handle.
139      * @param newHandle account's new handle.
140      * @throws HandleNotRecognisedException if the old handle does not match to any
141      *                                      account in the system.
142      * @throws IllegalHandleException   if the new handle already exists in the
143      *                                  platform.
144      * @throws InvalidHandleException   if the new handle is empty, has more
145      *                                  than 30 characters, or has white spaces.
146      */
147     public void changeAccountHandle(String oldHandle, String newHandle)
148           throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
149
150        if (checkAccountHandle(newHandle)) {
151           Account a = findAccountFromHandle(oldHandle);
152           a.setHandle(newHandle);
153        }
154     }
155
156     /**
157      * The method creates a formatted string summarising the stats of the account
```

```
158      * identified by the given handle. The template should be:
159      *
160      * <pre>
161      * ID: [account ID]
162      * Handle: [account handle]
163      * Description: [account description]
164      * Post count: [total number of posts, including endorsements and replies]
165      * Endorse count: [sum of endorsements received by each post of this account]
166      * </pre>
167      *
168      * @param handle handle to identify the account.
169      * @return the account formatted summary.
170      * @throws HandleNotRecognisedException if the handle does not match to any
171      *                                      account in the system.
172      */
173     public String showAccount(String handle) throws HandleNotRecognisedException {
174        Account a = findAccountFromHandle(handle);
175        int numberOfPosts = a.getPosts().size();
176
177        return "ID: " + a.getId() + "\nHandle: " + a.getHandle() + "\nDescription: " + a.getDescription()
178              + "\nPost count: " + numberOfPosts + "\nEndorse count: " + countEndorsements(a);
179     }
180
181     /**
182      * The method updates the description of the account with the respective handle.
183      * <p>
184      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
185      * are thrown.
186      *
187      * @param handle    handle to identify the account.
188      * @param description new text for description.
189      * @throws HandleNotRecognisedException if the handle does not match to any
190      *                                      account in the system.
191      */
192     public void updateAccountDescription(String handle, String description) throws
          HandleNotRecognisedException {
193        Boolean handleRecognised = false;
194
195        for (Account a : accounts) {
196           if (a.getHandle().equals(handle)) {
197              a.setDescription(description);
198              handleRecognised = true;
199              break;
200           }
201        }
202        if (!handleRecognised) {
203           throw new HandleNotRecognisedException("The handle " + handle + " is not recognised.");
204        }
205     }
206
207     /**
208      * The method creates a post for the account identified by the given handle with
209      * the following message.
210      * <p>
211      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
```

```
212      * are thrown.
213      *
214      * @param handle handle to identify the account.
215      * @param message post message.
216      * @throws HandleNotRecognisedException if the handle does not match to any
217      *                                      account in the system.
218      * @throws InvalidPostException     if the message is empty or has more than
219      *                                      100 characters.
220      * @return the sequential ID of the created post.
221      */
222     public int createPost(String handle, String message) throws HandleNotRecognisedException,
         InvalidPostException {
223       Account a = findAccountFromHandle(handle);
224       if (message.equals("") || message.length() > 100) {
225          throw new InvalidPostException("The post : " + message + " is invalid");
226       } else {
227          Post post = new Post(a, message);
228          posts.add(post);
229          a.addPostToAccount(post);
230          return post.getPostID();
231       }
232
233     }
234
235     /**
236      * The method creates an endorsement post of an existing post, similar to a
237      * retweet on Twitter. An endorsement post is a special post. It contains a
238      * reference to the endorsed post and its message is formatted as:
239      * <p>
240      * <code>"EP@" + [endorsed account handle] + ": " + [endorsed message]</code>
241      * <p>
242      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
243      * are thrown.
244      *
245      * @param handle of the account endorsing a post.
246      * @param id    of the post being endorsed.
247      * @return the sequential ID of the created post.
248      * @throws HandleNotRecognisedException if the handle does not match to any
249      *                                      account in the system.
250      * @throws PostIDNotRecognisedException if the ID does not match to any post in
251      *                                      the system.
252      * @throws NotActionablePostException if the ID refers to a endorsement post.
253      *                                      Endorsement posts are not endorsable.
254      *                                      Endorsements are not transitive. For
255      *                                      instance, if post A is endorsed by post
256      *                                      B, and an account wants to endorse B, in
257      *                                      fact, the endorsement must refers to A.
258      */
259     public int endorsePost(String handle, int id)
260          throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException {
261
262       Account a = findAccountFromHandle(handle);
263       Post postEndorsed = getPostFromId(id);
264       if (postEndorsed instanceof Endorsement) {
265          throw new NotActionablePostException("Endorsement posts are not endorsable.");
```

```
266         }
267         for (Post p : posts) {
268             if (p instanceof Endorsement) {
269                 if (p.getAuthor() == a && ((Endorsement) p).getRefersTo() == postEndorsed) {
270                     return p.getPostID();
271                 }
272             }
273         }
274         Endorsement endorsement = new Endorsement(a, postEndorsed);
275         postEndorsed.addReply(endorsement);
276         postEndorsed.incrementEndorsements();
277         postEndorsed.getAuthor().incrementEndorsements();
278         a.addPostToAccount(endorsement);
279         posts.add(endorsement);
280         return endorsement.getPostID();
281 
282     }
283 
284     /**
285      * The method creates a comment post referring to an existing post, similarly to
286      * a reply on Twitter. A comment post is a special post. It contains a reference
287      * to the post being commented upon.
288      * <p>
289      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
290      * are thrown.
291      *
292      * @param handle of the account commenting a post.
293      * @param id     of the post being commented.
294      * @param message the comment post message.
295      * @return the sequential ID of the created post.
296      * @throws HandleNotRecognisedException if the handle does not match to any
297      *                                      account in the system.
298      * @throws PostIDNotRecognisedException if the ID does not match to any post in
299      *                                      the system.
300      * @throws NotActionablePostException if the ID refers to a endorsement post.
301      *                                      Endorsement posts are not endorsable.
302      *                                      Endorsements cannot be commented. For
303      *                                      instance, if post A is endorsed by post
304      *                                      B, and an account wants to comment B, in
305      *                                      fact, the comment must refers to A.
306      * @throws InvalidPostException     if the comment message is empty or has
307      *                                      more than 100 characters.
308      */
309     public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
310             PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
311         Post postCommentedOn = getPostFromId(id);
312         Account commenter = findAccountFromHandle(handle);
313         if (postCommentedOn instanceof Endorsement) {
314             throw new NotActionablePostException("You cannot comment on an Endorsement.");
315         }
316 
317         if (message.equals("") || message.length() > 100) {
318 
319             throw new InvalidPostException("The post : " + message + " is invalid");
320         } else {
```

```
321         Comment comment = new Comment(commenter, message, postCommentedOn);
322         postCommentedOn.addReply((Post) comment);
323         postCommentedOn.incrementComments();
324         commenter.addPostToAccount(comment);
325         postCommentedOn.getAuthor().incrementComments();
326         posts.add(comment);
327         return comment.getPostID();
328       }
329
330     }
331
332     /**
333      * The method removes the post from the platform. When a post is removed, all
334      * its endorsements should be removed as well. All replies to this post should
335      * be updated by replacing the reference to this post by a generic empty post.
336      * <p>
337      * The generic empty post message should be "The original content was removed
338      * from the system and is no longer available.". This empty post is just a
339      * replacement placeholder for the post which a reply refers to. Empty posts
340      * should not be linked to any account and cannot be acted upon, i.e., it cannot
341      * be available for endorsements or replies.
342      * <p>
343      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
344      * are thrown.
345      *
346      * @param id ID of post to be removed.
347      * @throws PostIDNotRecognisedException if the ID does not match to any post in
348      *                                      the system.
349      */
350     public void deletePost(int id) throws PostIDNotRecognisedException {
351       Post deletedPost = getPostFromId(id);
352       GenericEmptyPost genericEmptyPost = new GenericEmptyPost(deletedPost);
353       Account deletedPostAccount = deletedPost.getAuthor();
354       Post parentOfDeletedPost = null;
355
356       if (deletedPost instanceof Endorsement) {
357
358         parentOfDeletedPost = ((Endorsement) deletedPost).getRefersTo();
359         posts.remove(deletedPost);
360         deletedPostAccount.getPosts().remove(deletedPost);
361         parentOfDeletedPost.decrementEndorsements();
362         parentOfDeletedPost.removeReply(deletedPost);
363         parentOfDeletedPost.getAuthor().decrementEndorsements();
364
365       } else if (deletedPost instanceof Comment) {
366
367         posts.remove(deletedPost);
368         deletedPostAccount.getPosts().remove(deletedPost);
369         parentOfDeletedPost = ((Comment) deletedPost).getReplyingTo();
370         parentOfDeletedPost.decrementComments();
371         parentOfDeletedPost.removeReply(deletedPost);
372         parentOfDeletedPost.getAuthor().decrementComments();
373         if (!deletedPost.getReplies().isEmpty()) {
374           parentOfDeletedPost.addReply(genericEmptyPost);
375         }
```

```
376         for (Post p : deletedPost.getReplies()) {
377            if (p instanceof Comment) {
378               ((Comment) p).setReplyingTo(genericEmptyPost);
379               deletedPostAccount.decrementComments();
380            } else if (p instanceof Endorsement) {
381               ((Endorsement) p).setRefersTo(genericEmptyPost);
382               deletedPostAccount.decrementEndorsements();
383            }
384         }
385      } else {
386
387         posts.remove(deletedPost);
388         deletedPostAccount.getPosts().remove(deletedPost);
389         for (Post p : deletedPost.getReplies()) {
390            if (p instanceof Comment) {
391               ((Comment) p).setReplyingTo(genericEmptyPost);
392               deletedPostAccount.decrementComments();
393            } else if (p instanceof Endorsement) {
394               ((Endorsement) p).setRefersTo(deletedPost);
395               deletedPostAccount.decrementEndorsements();
396            }
397         }
398      }
399
400   }
401
402   /**
403    * The method generates a formated string containing the details of a single
404    * post. The format is as follows:
405    *
406    * <pre>
407    * ID: [post ID]
408    * Account: [account handle]
409    * No. endorsements: [number of endorsements received by the post] | No. comments: [number of comments
         received by the post]
410    * [post message]
411    * </pre>
412    *
413    * @param id of the post to be shown.
414    * @return a formatted string containing post's details.
415    * @throws PostIDNotRecognisedException if the ID does not match to any post in
416    *                                      the system.
417    */
418   public String showIndividualPost(int id) throws PostIDNotRecognisedException {
419      Post post = getPostFromId(id);
420
421      if (post instanceof Endorsement) {
422         return "";
423
424      } else {
425         return "ID: " + post.getPostID() + "\nAccount: " + post.getAuthor().getHandle() + "\nNo.
               endorsements: "
426               + post.getNumberOfEndorsements() + " | No. comments: " + post.getNumberOfComments() + "\n"
427               + post.getMessage() + "\n";
428      }
```

```
429
430     }
431
432     /**
433      * (Overloaded method: accounts for indentation) The method generates a formated
434      * string containing the details of a single post. The format is as follows:
435      *
436      * <pre>
437      * ID: [post ID]
438      * Account: [account handle]
439      * No. endorsements: [number of endorsements received by the post] | No. comments: [number of comments
             received by the post]
440      * [post message]
441      * </pre>
442      *
443      * @param id of the post to be shown.
444      * @return a formatted string containing post's details.
445      * @throws PostIDNotRecognisedException if the ID does not match to any post in
446      *                                      the system.
447      */
448     public String showIndividualPost(Post post, int indentation, Boolean firstReply)
449           throws PostIDNotRecognisedException {
450
451       String indent = "";
452       for (int i = 0; i < indentation; i++) {
453          indent = indent + "  ";
454       }
455
456       if (post instanceof Endorsement) {
457
458          return "";
459
460       } else if (post instanceof GenericEmptyPost) {
461          if (firstReply) {
462             return indentationToString(indentation - 1) + "|\n" + indentationToString(indentation - 1) + "|
                   > ID: "
463                   + post.getPostID() + "\n" + indent + "Account: null" + "\n" + indent + "No. endorsements: "
464                   + post.getNumberOfEndorsements() + " | No. comments: " + post.getNumberOfComments() + "\n"
465                   + indent + post.getMessage() + "\n";
466          } else {
467             return indentationToString(indentation - 1) + "| > ID: " + post.getPostID() + "\n" + indent
468                   + "Account: null" + "\n" + indent + "No. endorsements: " + post.getNumberOfEndorsements()
469                   + " | No. comments: " + post.getNumberOfComments() + "\n" + indent + post.getMessage() +
                         "\n";
470          }
471
472       } else {
473          getPostFromId(post.getPostID());
474          if (firstReply) {
475             return indentationToString(indentation - 1) + "|\n" + indentationToString(indentation - 1) + "|
                   > ID: "
476                   + post.getPostID() + "\n" + indent + "Account: " + post.getAuthor().getHandle() + "\n" +
                         indent
477                   + "No. endorsements: " + post.getNumberOfEndorsements() + " | No. comments: "
478                   + post.getNumberOfComments() + "\n" + indent + post.getMessage() + "\n";
```

```
479         } else {
480            return indentationToString(indentation - 1) + "| > ID: " + post.getPostID() + "\n" + indent
481                  + "Account: " + post.getAuthor().getHandle() + "\n" + indent + "No. endorsements: "
482                  + post.getNumberOfEndorsements() + " | No. comments: " + post.getNumberOfComments() + "\n"
483                  + indent + post.getMessage() + "\n";
484         }
485
486      }
487
488   }
489
490   /**
491    * The method builds a StringBuilder showing the details of the current post and
492    * all its children posts. The format is as follows:
493    *
494    * <pre>
495    * {@link #showIndividualPost(int) showIndividualPost(id)}
496    * |
497    * [for reply: replies to the post sorted by ID]
498    * |  > {@link #showIndividualPost(int) showIndividualPost(reply)}
499    * </pre>
500    *
501    * See an example:
502    *
503    * <pre>
504    * ID: 1
505    * Account: user1
506    * No. endorsements: 2 | No. comments: 3
507    * I like examples.
508    * |
509    * | > ID: 3
510    *     Account: user2
511    *     No. endorsements: 0 | No. comments: 1
512    *     No more than me...
513    *     |
514    *     | > ID: 5
515    *         Account: user1
516    *         No. endorsements: 0 | No. comments: 1
517    *         I can prove!
518    *         |
519    *         | > ID: 6
520    *             Account: user2
521    *             No. endorsements: 0 | No. comments: 0
522    *             prove it
523    * | > ID: 4
524    *     Account: user3
525    *     No. endorsements: 4 | No. comments: 0
526    *     Can't you do better than this?
527    *
528    * | > ID: 7
529    *     Account: user5
530    *     No. endorsements: 0 | No. comments: 1
531    *     where is the example?
532    *     |
533    *     | > ID: 10
```

```
534      *        Account: user1
535      *        No. endorsements: 0 | No. comments: 0
536      *        This is the example!
537      * </pre>
538      *
539      * Continuing with the example, if the method is called for post ID=5
540      * ({@code showIndividualPost(5)}), the return would be:
541      *
542      * <pre>
543      * ID: 5
544      * Account: user1
545      * No. endorsements: 0 | No. comments: 1
546      * I can prove!
547      * |
548      * | > ID: 6
549      *    Account: user2
550      *    No. endorsements: 0 | No. comments: 0
551      *    prove it
552      * </pre>
553      *
554      * @param id of the post to be shown.
555      * @return a formatted StringBuilder containing the details of the post and its
556      *        children.
557      * @throws PostIDNotRecognisedException if the ID does not match to any post in
558      *                                      the system.
559      * @throws NotActionablePostException if the ID refers to an endorsement post.
560      *                                      Endorsement posts do not have children
561      *                                      since they are not endorsable nor
562      *                                      commented.
563      */
564     public StringBuilder showPostChildrenDetails(int id)
565         throws PostIDNotRecognisedException, NotActionablePostException {
566       StringBuilder postChildrenDetails = new StringBuilder();
567       Post originalPost = getPostFromId(id);
568       if (originalPost instanceof Endorsement) {
569
570         throw new NotActionablePostException("Endorsement posts do not have children.");
571       }
572
573       Post currentPost = originalPost;
574       int nodeIndexOnCurrentLevel = 0;
575       int indent = 0;
576       Boolean firstReply = false;
577
578       if (!hasChildren(currentPost)) {
579         postChildrenDetails.append(showIndividualPost(currentPost.getPostID()));
580         currentPost = null;
581       } else {
582         Boolean onlyEndorsementChildren = true;
583         for (Post post : currentPost.getReplies()) {
584           if (post instanceof Comment) {
585             onlyEndorsementChildren = false;
586             break;
587           }
588           if (post instanceof GenericEmptyPost) {
```

```
589                onlyEndorsementChildren = false;
590                break;
591             }
592          }
593
594          if (onlyEndorsementChildren) {
595             postChildrenDetails.append(showIndividualPost(currentPost.getPostID()));
596             currentPost = null;
597          }
598
599       }
600
601       while (currentPost != null) {
602          if (indent == 0) {
603             postChildrenDetails.append(showIndividualPost(currentPost.getPostID()));
604          } else {
605             postChildrenDetails.append(showIndividualPost(currentPost, indent, firstReply));
606          }
607
608          // can you go down?
609          if (hasChildren(currentPost)) {
610             nodeIndexOnCurrentLevel = 0;
611             currentPost = currentPost.getReplies().get(nodeIndexOnCurrentLevel);
612             indent += 1;
613             firstReply = true;
614          } else {
615
616             // can't go down?
617             // can you go adjacent?
618             Boolean canGoAdjacent = false;
619             while (!(canGoAdjacent)) {
620                nodeIndexOnCurrentLevel += 1;
621                Post parent = null;
622
623                if (currentPost instanceof Comment) {
624
625                   parent = ((Comment) currentPost).getReplyingTo();
626                } else if (currentPost instanceof Endorsement) {
627
628                   parent = ((Endorsement) currentPost).getRefersTo();
629                } else if (currentPost instanceof GenericEmptyPost) {
630
631                   parent = ((GenericEmptyPost) currentPost).getReplyingTo();
632
633                }
634                // checking if you can go adjacent:
635                if (nodeIndexOnCurrentLevel < parent.getReplies().size()) {
636
637                   // you go adjacent
638                   currentPost = parent.getReplies().get(nodeIndexOnCurrentLevel);
639                   canGoAdjacent = true;
640                   firstReply = false;
641                   // break out of while loop, go down as far as possible
642                } else {
643                   canGoAdjacent = false;
```

```
                // can't go adjacent, can you go up
                if (parent == originalPost) {
                   // can't go up. break
                   currentPost = null;
                   indent = 0;
                   break;
                } else {
                   // can go up:
                   Post grandparent = null;
                   if (parent instanceof GenericEmptyPost) {
                      grandparent = ((GenericEmptyPost) parent).getReplyingTo();
                   } else {
                      grandparent = ((Comment) parent).getReplyingTo();
                   }
                   nodeIndexOnCurrentLevel = grandparent.getReplies().indexOf((parent));
                   currentPost = parent;
                   if (indent > 1) {
                      indent -= 1;
                   }

                }
             }

          }
       }

    }

    return postChildrenDetails;

}

// End Post-related methods ****************************************

// Analytics-related methods ****************************************

/**
 * This method returns the current total number of accounts present in the
 * platform. Note, this is NOT the total number of accounts ever created since
 * the current total should discount deletions.
 *
 * @return the total number of accounts in the platform.
 */
public int getNumberOfAccounts() {
   return accounts.size();
}

/**
 * This method returns the current total number of original posts (i.e.,
 * disregarding endorsements and comments) present in the platform. Note, this
 * is NOT the total number of posts ever created since the current total should
 * discount deletions.
 *
 * @return the total number of original posts in the platform.
 */
```

```
699     public int getTotalOriginalPosts() {
700        int totalOriginalPosts = 0;
701        for (Post p : posts) {
702           if (!(p instanceof Endorsement) && !(p instanceof Comment)) {
703              totalOriginalPosts += 1;
704           }
705        }
706        return totalOriginalPosts;
707     }
708
709     /**
710      * This method returns the current total number of endorsement posts present in
711      * the platform. Note, this is NOT the total number of endorsements ever created
712      * since the current total should discount deletions.
713      *
714      * @return the total number of endorsement posts in the platform.
715      */
716     public int getTotalEndorsmentPosts() {
717        int totalEndorsementPosts = 0;
718        for (Post p : posts) {
719           if (p instanceof Endorsement) {
720              totalEndorsementPosts += 1;
721           }
722        }
723        return totalEndorsementPosts;
724
725     }
726
727     /**
728      * This method returns the current total number of comments posts present in the
729      * platform. Note, this is NOT the total number of comments ever created since
730      * the current total should discount deletions.
731      *
732      * @return the total number of comments posts in the platform.
733      */
734     public int getTotalCommentPosts() {
735        int totalCommentedPosts = 0;
736        for (Post p : posts) {
737           if (p instanceof Comment) {
738              totalCommentedPosts += 1;
739           }
740        }
741        return totalCommentedPosts;
742     }
743
744     /**
745      * This method identifies and returns the post with the most number of
746      * endorsements, a.k.a. the most popular post.
747      *
748      * @return the ID of the most popular post.
749      */
750     public int getMostEndorsedPost() {
751        Post mostEndorsedPost = null;
752        int mostEndorsements = 0;
753        for (Post p : posts) {
```

```
754             if (p.getNumberOfEndorsements() > mostEndorsements) {
755                mostEndorsedPost = p;
756                mostEndorsements = p.getNumberOfEndorsements();
757             }
758          }
759          return mostEndorsedPost.getPostID();
760       }
761
762       /**
763        * This method identifies and returns the account with the most number of
764        * endorsements, a.k.a. the most popular account.
765        *
766        * @return the ID of the most popular account.
767        */
768       public int getMostEndorsedAccount() {
769          int mostEndorsements = 0;
770          Account mostEndorsedAccount = null;
771
772          for (Account account : accounts) {
773             if (account.getNumberOfEndorsements() > mostEndorsements) {
774                mostEndorsedAccount = account;
775                mostEndorsements = account.getNumberOfEndorsements();
776             }
777          }
778          return mostEndorsedAccount.getId();
779
780       }
781
782       // Management-related methods ****************************************
783
784       /**
785        * Method empties this SocialMediaPlatform of its contents and resets all
786        * internal counters.
787        */
788       public void erasePlatform() {
789          posts.clear();
790          accounts.clear();
791          Post.setChronologicalId(0);
792          Account.setIdCount(10000000);
793
794       }
795
796       /**
797        * Method saves this SocialMediaPlatform's contents into a serialised file, with
798        * the filename given in the argument.
799        *
800        * @param filename location of the file to be saved
801        * @throws IOException if there is a problem experienced when trying to save the
802        *                     store contents to the file
803        */
804       public void savePlatform(String filename) throws IOException {
805          String fileName = filename + ".ser";
806          File outFile = new File(fileName);
807          try {
808             ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(outFile));
```

```
809         oos.writeObject(accounts);
810         oos.writeObject(posts);
811         oos.flush();
812         oos.close();
813      } catch (Exception e) {
814         e.printStackTrace();
815      }
816
817   }
818
819   /**
820    * Method should load and replace this SocialMediaPlatform's contents with the
821    * serialised contents stored in the file given in the argument.
822    * <p>
823    * The state of this SocialMediaPlatform's must be be unchanged if any
824    * exceptions are thrown.
825    *
826    * @param filename location of the file to be loaded
827    * @throws IOException         if there is a problem experienced when trying
828    *                             to load the store contents from the file
829    * @throws ClassNotFoundException if required class files cannot be found when
830    *                             loading
831    */
832   public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
833      try (ObjectInputStream objStream = new ObjectInputStream(new FileInputStream(filename))) {
834         Object obj = objStream.readObject();
835
836         if (obj instanceof ArrayList<?>) {
837            @SuppressWarnings("unchecked")
838            ArrayList<Account> importedAccounts = (ArrayList<Account>) obj;
839            accounts = importedAccounts;
840
841         }
842
843         Object obj2 = objStream.readObject();
844         if (obj2 instanceof ArrayList<?>) {
845            @SuppressWarnings("unchecked")
846            ArrayList<Post> importedPosts = (ArrayList<Post>) obj2;
847            posts = importedPosts;
848
849         }
850
851      } catch (IOException e) {
852         e.printStackTrace();
853      }
854
855   }
856
857   // End Management-related methods ****************************************
858
859   // Internal methods ******************************************
860
861   /**
862    * Finds and returns an account with the given handle, if it exists.
863    *
```

```
     * @param handle the handle being searched for
     * @return the associated Account
     * @throws HandleNotRecognisedException if the handle is not in the system
     */
    private Account findAccountFromHandle(String handle) throws HandleNotRecognisedException {

        for (Account a : accounts) {
            if (a.getHandle().equals(handle)) {
                return a;
            }
        }

        throw new HandleNotRecognisedException("The handle " + handle + " is not recognised.");

    }

    /**
     * Finds and returns a particular post from an ID
     *
     * @param id the id of the post being searched for
     * @return the Post with the associated ID
     * @throws PostIDNotRecognisedException if the ID does not match any posts in
     *                                      the system
     */
    private Post getPostFromId(int id) throws PostIDNotRecognisedException {

        for (Post p : posts) {

            if (p.getPostID() == id) {
                return p;
            }
        }

        throw new PostIDNotRecognisedException("The post ID " + id + " is not recognised.");

    }

    /**
     * Checks the validity of an account handle
     *
     * @param handle the handle being checked
     * @return true if the handle is valid, false if not
     * @throws InvalidHandleException if the handle is empty, has more than 30
     *                                characters, or has white spaces.
     * @throws IllegalHandleException if the handle is already associated with an
     *                                existing account
     */
    private static Boolean checkAccountHandle(String handle) throws InvalidHandleException,
        IllegalHandleException {
        if (Account.isHandleUnique(handle)) {
            if (Account.isHandleValid(handle)) {
                return true;
            } else {

                throw new InvalidHandleException("Handle is invalid: " + handle);
```

```
918
919             }
920         } else {
921
922             throw new IllegalHandleException("Handle already exists: " + handle);
923         }
924
925     }
926
927     /**
928      * Totals the number of endorsements for a particular account i.e. the number of
929      * endorsments on the posts of this account
930      *
931      * @param a the account being counted for endorsements
932      * @return the number of endorsements
933      */
934     private int countEndorsements(Account a) {
935         int totalEndorsements = 0;
936         for (Post p : a.getPosts()) {
937             for (Post q : p.getReplies()) {
938                 if (q instanceof Endorsement) {
939                     totalEndorsements += 1;
940                 }
941             }
942
943         }
944         return totalEndorsements;
945     }
946
947     /**
948      * Checks if a post has children i.e. comments or endorsements
949      *
950      * @param post the post being checked
951      * @return true if the post has children, false if not
952      */
953     private Boolean hasChildren(Post post) {
954         if (post.getReplies().size() == 0) {
955             return false;
956         } else {
957             return true;
958         }
959
960     }
961
962     /**
963      * Changes an int of the level of intended indentation into a string with that
964      * level of indent
965      *
966      * @param indent the number of indentations, where one indentation is " "
967      * @return the total string of indentations
968      */
969     private String indentationToString(int indent) {
970         String indentation = "";
971         for (int i = 0; i < indent; i++) {
972             indentation += "  ";
```

```
973         }
974         return indentation;
975     }
976
977 }
```