

File: Account.java

```
001: /**
002:  * Represents an account, located in the social media package
003:  * An account is made up of a user ID (uid), a handle a description field and a list of all
004:  * posts it has made.
005:  * @author Jeroen Mijer, Alexander Robertson
006:  * @version 1.0
007:  * @since 1.0
008:  */
009: package socialmedia;
010:
011: import java.util.ArrayList;
012: import java.io.Serializable;
013:
014:
015: public class Account implements Serializable {
016:     private Integer uid;
017:     private String handle;
018:     private String descField;
019:     private ArrayList<Post> posts;
020:
021:     /**
022:      * Creates an account
023:      * Sets the arraylist of posts to empty, since an account can only make posts after it
024:      * has been created
025:      * @param handle code of the account
026:      * @param descField description field of the account
027:      * @param uid user ID of the account
028:      */
029:     public Account(String handle, String descField, Integer uid) {
030:         this.uid = uid;
031:         this.handle = handle;
032:         this.descField = descField;
033:         this.posts = new ArrayList<Post>();
034:     }
035:
036:
037:     /**
038:      * Gets account description field
039:      *
040:      * @return description field of the account
041:      */
042:     public String getDescField() {
043:         return descField;
044:     }
```

```
045:
046:
047:  /** Gets user id
048:   *
049:   * @return User ID of the account
050:   */
051:  public Integer getUID() {
052:      return uid;
053:  }
054:
055:
056:  /** Gets account handle
057:   *
058:   * @return Handle of the account
059:   */
060:  public String getHandle() {
061:      return handle;
062:  }
063:
064:
065:  /** Gets arraylist of posts account has made
066:   *
067:   * @return ArrayList of posts account has made
068:   */
069:  public ArrayList<Post> getPosts() {
070:      return posts;
071:  }
072:
073:
074:  /** Sets description field of account
075:   *
076:   * @param descField new description field of the account
077:   */
078:  public void setDescField(String descField) {
079:      this.descField = descField;
080:  }
081:
082:
083:  /** Sets handle of the account
084:   *
085:   * @param handle new handle of the account
086:   */
087:  public void setHandle(String handle) {
088:      this.handle = handle;
089:  }
090:
091:
```

```
092:  /** Adds post to arraylist of posts contained within account
093:  *
094:  * @param post Post being appended to the arraylist
095:  */
096:  public void addPost(Post post) {
097:      this.posts.add(post);
098:  }
099:
100:
101:  /** Removes original post from arraylist of posts contained within account
102:  *
103:  * @param post (OriginalPost) Post being removed from Arraylist
104:  */
105:  public void removePost(OriginalPost post) {
106:      this.posts.remove(post);
107:  }
108:
109:  /** Removes comment from arraylist of posts contained within account
110:  *
111:  * @param post (Comment) Post being removed from Arraylist
112:  */
113:  public void removePost(Comment post) {
114:      this.posts.remove(post);
115:  }
116:
117:  /** Removes endorsement from arraylist of posts contained within account
118:  *
119:  * @param post (Endorsement)Post being removed from Arraylist
120:  */
121:  public void removePost(Endorsement post) {
122:      this.posts.remove(post);
123:  }
124: }
```

```

01: /**
02:  * Represents a comment, which extends the class Post, which is located in the social
03:  * media package
04:  * An account is made up of a message, author, post ID (pid), arraylist of comments,
05:  * arraylist of endorsements, a postpointer(points to parent post)
06:  *
07:  * @author Jeroen Mijer, Alexander Robertson
08:  * @version 1.0
09:  * @since 1.0
10: */
11: package socialmedia;
12: import java.util.ArrayList;
13:
14: public class Comment extends Post {
15:
16:     Integer postPointer; // holds the postID that the comment is replying to
17:     ArrayList <Endorsement> endorsements;
18:     ArrayList <Comment> comments;
19:
20:     /**
21:      * Creates a comment
22:      * Sets the arraylist of comments and endorsements to empty, since a comment can
23:      * only be commented on or endorsed after creation
24:      * @param message message of the comment
25:      * @param author author of the comment
26:      * @param postPointer post ID of the parent post
27:      * @param pid post ID of the comment
28:      */
29:     public Comment(String message, Account author, Integer postPointer, Integer pid) {
30:         super(message, author, pid);
31:         this.endorsements = new ArrayList<Endorsement>();
32:         this.comments = new ArrayList<Comment>();
33:         this.postPointer = postPointer;
34:     }
35:
36:     /** Gets post pointer
37:      *
38:      * @return post pointer of the comment
39:      */
40:     public Integer getPostPointer() {
41:         return postPointer;
42:     }
43:
44:     /** Gets arraylist of endorsements
45:      *
46:      * @return arraylist of endorsements
47:      */
48:     public ArrayList<Endorsement> getEndorsements() {
49:         return endorsements;
50:     }
51:
52:     /** Adds endorsement
53:      *
54:      * @param endorsement endorsement to be added
55:      */
56:     public void addEndorsement(Endorsement endorsement) {
57:         endorsements.add(endorsement);
58:     }
59:
60:     /** Gets arraylist of comments
61:      *
62:      * @return arraylist of comments
63:      */
64:     public ArrayList<Comment> getComments() {
65:         return comments;
66:     }
67:
68:     /** Adds comment
69:      *
70:      * @param comment comment to be added
71:      */
72:     public void addComment(Comment comment) {
73:         comments.add(comment);
74:     }
75:
76:     /** Gets post pointer
77:      *
78:      * @return post pointer of the comment
79:      */
80:     public Integer getPostPointer() {
81:         return postPointer;
82:     }
83:
84:     /** Sets post pointer
85:      *
86:      * @param postPointer post pointer to be set
87:      */
88:     public void setPostPointer(Integer postPointer) {
89:         this.postPointer = postPointer;
90:     }
91:
92:     /** Gets arraylist of endorsements
93:      *
94:      * @return arraylist of endorsements
95:      */
96:     public ArrayList<Endorsement> getEndorsements() {
97:         return endorsements;
98:     }
99:
100:     /** Sets arraylist of endorsements
101:      *
102:      * @param endorsements arraylist of endorsements to be set
103:      */
104:     public void setEndorsements(ArrayList<Endorsement> endorsements) {
105:         this.endorsements = endorsements;
106:     }
107:
108:     /** Gets arraylist of comments
109:      *
110:      * @return arraylist of comments
111:      */
112:     public ArrayList<Comment> getComments() {
113:         return comments;
114:     }
115:
116:     /** Sets arraylist of comments
117:      *
118:      * @param comments arraylist of comments to be set
119:      */
120:     public void setComments(ArrayList<Comment> comments) {
121:         this.comments = comments;
122:     }
123:
124:     /** Gets message
125:      *
126:      * @return message of the comment
127:      */
128:     public String getMessage() {
129:         return message;
130:     }
131:
132:     /** Sets message
133:      *
134:      * @param message message to be set
135:      */
136:     public void setMessage(String message) {
137:         this.message = message;
138:     }
139:
140:     /** Gets author
141:      *
142:      * @return author of the comment
143:      */
144:     public Account getAuthor() {
145:         return author;
146:     }
147:
148:     /** Sets author
149:      *
150:      * @param author author to be set
151:      */
152:     public void setAuthor(Account author) {
153:         this.author = author;
154:     }
155:
156:     /** Gets post ID
157:      *
158:      * @return post ID of the comment
159:      */
160:     public Integer getPid() {
161:         return pid;
162:     }
163:
164:     /** Sets post ID
165:      *
166:      * @param pid post ID to be set
167:      */
168:     public void setPid(Integer pid) {
169:         this.pid = pid;
170:     }
171:
172:     /** Gets post pointer
173:      *
174:      * @return post pointer of the comment
175:      */
176:     public Integer getPostPointer() {
177:         return postPointer;
178:     }
179:
180:     /** Sets post pointer
181:      *
182:      * @param postPointer post pointer to be set
183:      */
184:     public void setPostPointer(Integer postPointer) {
185:         this.postPointer = postPointer;
186:     }
187:
188:     /** Gets arraylist of comments
189:      *
190:      * @return arraylist of comments
191:      */
192:     public ArrayList<Comment> getComments() {
193:         return comments;
194:     }
195:
196:     /** Sets arraylist of comments
197:      *
198:      * @param comments arraylist of comments to be set
199:      */
200:     public void setComments(ArrayList<Comment> comments) {
201:         this.comments = comments;
202:     }
203:
204:     /** Gets arraylist of endorsements
205:      *
206:      * @return arraylist of endorsements
207:      */
208:     public ArrayList<Endorsement> getEndorsements() {
209:         return endorsements;
210:     }
211:
212:     /** Sets arraylist of endorsements
213:      *
214:      * @param endorsements arraylist of endorsements to be set
215:      */
216:     public void setEndorsements(ArrayList<Endorsement> endorsements) {
217:         this.endorsements = endorsements;
218:     }
219:
220:     /** Gets message
221:      *
222:      * @return message of the comment
223:      */
224:     public String getMessage() {
225:         return message;
226:     }
227:
228:     /** Sets message
229:      *
230:      * @param message message to be set
231:      */
232:     public void setMessage(String message) {
233:         this.message = message;
234:     }
235:
236:     /** Gets author
237:      *
238:      * @return author of the comment
239:      */
240:     public Account getAuthor() {
241:         return author;
242:     }
243:
244:     /** Sets author
245:      *
246:      * @param author author to be set
247:      */
248:     public void setAuthor(Account author) {
249:         this.author = author;
250:     }
251:
252:     /** Gets post ID
253:      *
254:      * @return post ID of the comment
255:      */
256:     public Integer getPid() {
257:         return pid;
258:     }
259:
260:     /** Sets post ID
261:      *
262:      * @param pid post ID to be set
263:      */
264:     public void setPid(Integer pid) {
265:         this.pid = pid;
266:     }
267:
268:     /** Gets post pointer
269:      *
270:      * @return post pointer of the comment
271:      */
272:     public Integer getPostPointer() {
273:         return postPointer;
274:     }
275:
276:     /** Sets post pointer
277:      *
278:      * @param postPointer post pointer to be set
279:      */
280:     public void setPostPointer(Integer postPointer) {
281:         this.postPointer = postPointer;
282:     }
283:
284:     /** Gets arraylist of comments
285:      *
286:      * @return arraylist of comments
287:      */
288:     public ArrayList<Comment> getComments() {
289:         return comments;
290:     }
291:
292:     /** Sets arraylist of comments
293:      *
294:      * @param comments arraylist of comments to be set
295:      */
296:     public void setComments(ArrayList<Comment> comments) {
297:         this.comments = comments;
298:     }
299:
300:     /** Gets arraylist of endorsements
301:      *
302:      * @return arraylist of endorsements
303:      */
304:     public ArrayList<Endorsement> getEndorsements() {
305:         return endorsements;
306:     }
307:
308:     /** Sets arraylist of endorsements
309:      *
310:      * @param endorsements arraylist of endorsements to be set
311:      */
312:     public void setEndorsements(ArrayList<Endorsement> endorsements) {
313:         this.endorsements = endorsements;
314:     }
315:
316:     /** Gets message
317:      *
318:      * @return message of the comment
319:      */
320:     public String getMessage() {
321:         return message;
322:     }
323:
324:     /** Sets message
325:      *
326:      * @param message message to be set
327:      */
328:     public void setMessage(String message) {
329:         this.message = message;
330:     }
331:
332:     /** Gets author
333:      *
334:      * @return author of the comment
335:      */
336:     public Account getAuthor() {
337:         return author;
338:     }
339:
340:     /** Sets author
341:      *
342:      * @param author author to be set
343:      */
344:     public void setAuthor(Account author) {
345:         this.author = author;
346:     }
347:
348:     /** Gets post ID
349:      *
350:      * @return post ID of the comment
351:      */
352:     public Integer getPid() {
353:         return pid;
354:     }
355:
356:     /** Sets post ID
357:      *
358:      * @param pid post ID to be set
359:      */
360:     public void setPid(Integer pid) {
361:         this.pid = pid;
362:     }
363:
364:     /** Gets post pointer
365:      *
366:      * @return post pointer of the comment
367:      */
368:     public Integer getPostPointer() {
369:         return postPointer;
370:     }
371:
372:     /** Sets post pointer
373:      *
374:      * @param postPointer post pointer to be set
375:      */
376:     public void setPostPointer(Integer postPointer) {
377:         this.postPointer = postPointer;
378:     }
379:
380:     /** Gets arraylist of comments
381:      *
382:      * @return arraylist of comments
383:      */
384:     public ArrayList<Comment> getComments() {
385:         return comments;
386:     }
387:
388:     /** Sets arraylist of comments
389:      *
390:      * @param comments arraylist of comments to be set
391:      */
392:     public void setComments(ArrayList<Comment> comments) {
393:         this.comments = comments;
394:     }
395:
396:     /** Gets arraylist of endorsements
397:      *
398:      * @return arraylist of endorsements
399:      */
400:     public ArrayList<Endorsement> getEndorsements() {
401:         return endorsements;
402:     }
403:
404:     /** Sets arraylist of endorsements
405:      *
406:      * @param endorsements arraylist of endorsements to be set
407:      */
408:     public void setEndorsements(ArrayList<Endorsement> endorsements) {
409:         this.endorsements = endorsements;
410:     }
411:
412:     /** Gets message
413:      *
414:      * @return message of the comment
415:      */
416:     public String getMessage() {
417:         return message;
418:     }
419:
420:     /** Sets message
421:      *
422:      * @param message message to
```

```
44:  * @return arraylist of endorsements that endorsed a comment
45:  */
46:  public ArrayList<Endorsement> getEndorsements() {
47:      return endorsements;
48:  }
49:
50:
51:  /** Gets arraylist of comments
52:   *
53:   * @return arraylist of comments that commented on a comment
54:   */
55:  public ArrayList<Comment> getComments() {
56:      return comments;
57:  }
58:
59:
60:  /** Adds an endorsement to the arraylist of endorsements to a comment
61:   *
62:   * @param endorsement endorsement to be added to the arraylist of endorsements
63:   */
64:  public void addEndorsement(Endorsement endorsement) {
65:      this.endorsements.add(endorsement);
66:  }
67:
68:
69:  /** Adds a comment to the arraylist of comments in a comment
70:   *
71:   * @param comment comment to be added to the arraylist of endorsements
72:   */
73:  public void addComment(Comment comment) {
74:      this.comments.add(comment);
75:  }
76:
77:
78:  /** Sets post pointer of the comment
79:   * Used only when the parent comment is deleted and the post pointer now refers to
the generic deleted message
80:   *
81:   * @param postPointer new postpointer that the comment will now contain
82:   */
83:  public void setPostPointer(Integer postPointer) {
84:      this.postPointer = postPointer;
85:  }
86: }
```

File: Endorsement.java

```
01: /**
02:  * Represents an endorsement, which extends the class Post, which is located in the
social media package
03:  * An endorsement is made up of a message, author, post ID (pid) and a
postpointer(points to parent post)
04:  *
05:  * @author Jeroen Mijer, Alexander Robertson
06:  * @version 1.0
07:  * @since 1.0
08:  */
09: package socialmedia;
10: public class Endorsement extends Post {
11:
12:     Integer postPointer; // holds the postID that the endorsement is endorsing
13:
14:     /**
15:      * Creates an endorsement
16:      *
17:      * @param message message of the endorsement
18:      * @param author author of the endorsement
19:      * @param postPointer post ID of the parent post
20:      * @param pid post ID of the endorsement
21:      */
22:     public Endorsement(String message, Account author, Integer postPointer, Integer pid)
{
23:         super(message, author, pid);
24:         this.postPointer = postPointer;
25:
26:     }
27:
28:
29:     /** Gets the post pointer of an endorsement
30:      *
31:      * @return post pointer of an endorsement
32:      */
33:     public Integer getPostPointer() {
34:         return postPointer;
35:     }
36:
37: }
```

File: OriginalPost.java

```
01: /**
02:  * Represents an original post, which extends the class Post, which is located in the
social media package
03:  * An original post is made up of a message, author, post ID (pid), arraylist of comments
and an arraylist of endorsements
04:  *
05:  * @author Jeroen Mijer, Alexander Robertson
06:  * @version 1.0
07:  * @since 1.0
08:  */
09:
10: package socialmedia;
11: import java.util.ArrayList;
12:
13: public class OriginalPost extends Post {
14:     ArrayList <Endorsement> endorsements;
15:     ArrayList <Comment> comments;
16:
17:     /**
18:      * Creates an original post
19:      * Sets the arraylist of comments and endorsements to empty, since an original post
can only be commented on or endorsed after creation
20:      * @param message message of the original post
21:      * @param author author of the original post
22:      * @param pid post id of the original post
23:      */
24:     public OriginalPost(String message, Account author, Integer pid) {
25:         super(message, author, pid);
26:         this.endorsements = new ArrayList<Endorsement>();
27:         this.comments = new ArrayList<Comment>();
28:
29:     }
30:
31:
32:     /** Gets arraylist of endorsements
33:      *
34:      * @return arraylist of endorsements that endorsed an original post
35:      */
36:     public ArrayList<Endorsement> getEndorsements() {
37:         return endorsements;
38:     }
39:
40:
41:     /** Gets arraylist of comments
42:      *
43:      * @return arraylist of comments that commented on an original post
```

```
44:  */
45:  public ArrayList<Comment> getComments() {
46:      return comments;
47:  }
48:
49:
50:  /** Adds an endorsement to the arraylist of endorsements in an original post
51:   *
52:   * @param endorsement endorsement to be added to the arraylist of endorsements
53:   * of an original post
54:   */
55:  public void addEndorsement(Endorsement endorsement) {
56:      this.endorsements.add(endorsement);
57:  }
58:
59:  /** Adds a comment to the arraylist of comments in an original post
60:   *
61:   * @param comment comment to be added to the arraylist of comments
62:   */
63:  public void addComment(Comment comment) {
64:      this.comments.add(comment);
65:  }
66: }
```


File: Post.java

```
01: /**
02:  * Represents a Post, which is used as the parent class of any type of Post in the social
03:  * media package, which is located in the social media package
04:  * A Post is made up of a message, an author and a post ID (pid)
05:  *
06:  * @author Jeroen Mijer, Alexander Robertson
07:  * @version 1.0
08:  */
09: package socialmedia;
10: import java.io.Serializable;
11:
12: public class Post implements Serializable {
13:     private int pid;
14:     private String message;
15:     private Account author;
16:
17:     /**
18:      * Creates a Post
19:      *
20:      * @param message message of the Post
21:      * @param author author of the Post
22:      * @param pid post id of the Post
23:      */
24:     public Post (String message, Account author, Integer pid){
25:         this.pid = pid;
26:         this.message = message;
27:         this.author = author;
28:     }
29:
30:
31:     /** Gets the author of a post
32:      *
33:      * @return `Account that created the post (author)
34:      */
35:     public Account getAuthor() {
36:         return author;
37:     }
38:
39:     /** Gets the message of a post
40:      *
41:      * @return message of a post
42:      */
43:     public String getMessage() {
44:         return message;
45:     }
}
```

```
46:
47:  /** Gets the post id (pid) of a post
48:   *
49:   * @return post id (pid) of a post
50:   */
51:  public int getPid() {
52:      return pid;
53:  }
54: }
```

File: SocialMedia.java

```
001: package socialmedia;
002:
003: import java.io.FileInputStream;
004: import java.io.FileOutputStream;
005: import java.io.IOException;
006: import java.io.ObjectInputStream;
007: import java.io.ObjectOutput;
008: import java.io.ObjectOutputStream;
009: import java.util.ArrayList;
010: import java.util.Optional;
011:
012: /**
013:  * SocialMedia is a compiling implementor of the SocialMediaPlatform interface.
014:  * SocialMedia is made up of an arraylist of all accounts created on the
015:  * platform and an arraylist of all posts made on the platform
016:  *
017:  * @author Jeroen Mijer
018:  * @author Alex Robertson
019:  * @version 1.0
020:  */
021: public class SocialMedia implements SocialMediaPlatform {
022:
023:     // Public list of all accounts
024:     public transient ArrayList<Account> accounts;
025:     // public list of all posts
026:     public transient ArrayList<Post> posts;
027:
028:     public SocialMedia() {
029:         /**
030:          * Creates a Social Media object Instantiates 2 empty arraylists of accounts
031:          and
032:          * posts because no account can be created or post can be posted until there
033:          is
034:          * a platform to be stored on
035:          *
036:          */
037:         this.accounts = new ArrayList<Account>();
038:         this.posts = new ArrayList<Post>();
039:
040:         // create generic post with author "admin" to contain the generic error
041:         message
042:         Account genericAccount = new Account("admin", "", 1);
043:         OriginalPost genericPost = new OriginalPost(
044:             "The original content was removed from the system and is no
045:             longer available.", genericAccount, 1);
046:         // add generic post and account to system arraylists
```

```

043:         this.accounts.add(genericAccount);
044:         this.posts.add(genericPost);
045:     }
046:
047:     @Override
048:     public int createAccount(String handle) throws IllegalArgumentException,
InvalidHandleException {
049:         // check if handle is valid
050:         // empty, more than 30 characters, has white spaces then is invalid
051:         if (handle.isEmpty() || handle.length() > 30 || handle.contains(" ")) {
052:             throw new InvalidHandleException("empty, OR more than 30
characters, OR has white spaces then is invalid");
053:         }
054:
055:         // check if handle is unique
056:         for (Account account : accounts) {
057:             if (account.getHandle().equals(handle)) {
058:                 throw new IllegalArgumentException("That handle is not
unique");
059:             }
060:         }
061:
062:         // If handle input is valid
063:         // create account with descfield=""
064:         Integer uid = accounts.size() + 1;
065:         Account accountTemp = new Account(handle, "", uid);
066:
067:         // add account to account list
068:         accounts.add(accountTemp);
069:
070:         // return uid
071:         return accountTemp.getUID();
072:
073:     }
074:
075:     @Override
076:     public int createAccount(String handle, String description) throws
IllegalArgumentException, InvalidHandleException {
077:         // check if handle is valid
078:         // empty, more than 30 characters, has white spaces then is invalid
079:         if (handle.isEmpty() || handle.length() > 30 || handle.contains(" ")) {
080:             throw new InvalidHandleException(
081:                 "Handle empty, OR more than 30 characters, OR has
white spaces then is invalid");
082:         }
083:
084:         // check if handle is unique

```

```

085:         for (Account account : accounts) {
086:             if (account.getHandle().equals(handle)) {
087:                 throw new IllegalHandleException("That handle is not
unique");
088:             }
089:         }
090:
091:         // If handle is valid
092:         // create account with descfield=description
093:         Integer uid = accounts.size() + 1;
094:         Account accountTemp = new Account(handle, description, uid);
095:
096:         // add account to account list
097:         accounts.add(accountTemp);
098:
099:         // return uid
100:         return accountTemp.getUID();
101:     }
102:
103:     @Override
104:     public void removeAccount(int id) throws AccountIDNotRecognisedException {
105:         boolean found = false;
106:         for (int i = 0; i < accounts.size(); i++) {
107:             if (accounts.get(i).getUID() == id) {
108:                 // for each post in the account delete the post using
deletePost method
109:                 for (Post post : accounts.get(i).getPosts()) {
110:                     try {
111:                         deletePost(post.getPid());
112:                     } catch (PostIDNotRecognisedException e) {
113:                         e.printStackTrace();
114:                     }
115:                 }
116:                 // remove account from the arraylist accounts
117:                 accounts.remove(id - 1);
118:                 found = true;
119:                 break;
120:             }
121:         }
122:         if (!found) {
123:             throw new AccountIDNotRecognisedException("Account ID not
recognised");
124:         }
125:     }
126:
127:     @Override
128:     public void removeAccount(String handle) throws HandleNotRecognisedException {

```

```

129:         boolean found = false;
130:         for (int i = 0; i < accounts.size(); i++) {
131:             if (accounts.get(i).getHandle().equals(handle)) {
132:
133:                 for (Post post : accounts.get(i).getPosts()) {
134:                     try {
135:                         deletePost(post.getPid());
136:                     } catch (PostIDNotRecognisedException e) {
137:                         e.printStackTrace();
138:                     }
139:                 }
140:                 // remove account from the arraylist accounts
141:                 accounts.remove(i);
142:                 found = true;
143:                 break;
144:             }
145:         }
146:         if (!found) {
147:             throw new HandleNotRecognisedException("Handle not recognised");
148:         }
149:     }
150:
151:     @Override
152:     public void changeAccountHandle(String oldHandle, String newHandle)
153:         throws HandleNotRecognisedException, IllegalHandleException,
154:         InvalidHandleException {
155:         // Check if new handle is valid
156:         if (newHandle.isEmpty() || newHandle.length() > 30 || newHandle.contains("
157:         ")) {
158:             throw new InvalidHandleException("empty, OR more than 30
159:         characters, OR has white spaces then is invalid");
160:         }
161:
162:         // check if new handle is unique
163:         for (Account account : accounts) {
164:             if (account.getHandle().equals(newHandle)) {
165:                 throw new IllegalHandleException("That handle is not
166:         unique");
167:             }
168:         }
169:
170:         // if input handle is valid
171:         // Change handle from oldHandle to newHandle
172:         boolean found = false;
173:         for (Account account : accounts) {
174:             if (account.getHandle().equals(oldHandle)) {
175:                 account.setHandle(newHandle);

```

```

172:             found = true;
173:         }
174:     }
175:     // If input handle does not match any existing handle found throw handle not
176:     // recognised exception
177:     if (!found) {
178:         throw new HandleNotRecognisedException("Handle not recognised");
179:     }
180:
181: }
182:
183: @Override
184: public void updateAccountDescription(String handle, String description) throws
HandleNotRecognisedException {
185:     boolean found = false;
186:     for (Account account : accounts) {
187:         if (account.getHandle().equals(handle)) {
188:             account.setDescField(description);
189:             found = true;
190:         }
191:     }
192:     // If input handle does not match any existing handle found throw handle not
193:     // recognised exception
194:     if (!found) {
195:         throw new HandleNotRecognisedException("Handle not recognised");
196:     }
197: }
198:
199: @Override
200: public String showAccount(String handle) throws HandleNotRecognisedException {
201:     boolean found = false;
202:     Account account = null;
203:     for (Account searchaccount : accounts) {
204:         if (searchaccount.getHandle().equals(handle)) {
205:             account = searchaccount;
206:             found = true;
207:         }
208:     }
209:     // If input handle does not match any existing handle found throw handle not
210:     // recognised exception
211:     if (!found) {
212:         throw new HandleNotRecognisedException("Handle not recognised");
213:     }
214:
215:     Integer endorseCount = 0;
216:     // for all posts in account check the amount of endorsements and add them
to

```

```

217:         // endorseCount
218:         for (Post accPost : account.getPosts()) {
219:             // if post is an original post (in order to downcast)
220:             if (accPost instanceof OriginalPost) {
221:                 endorseCount += ((OriginalPost)
accPost).getEndorsements().size();
222:             }
223:             // else if post is Comment (in order to downcast)
224:             else if (accPost instanceof Comment) {
225:                 endorseCount += ((Comment)
accPost).getEndorsements().size();
226:             }
227:
228:         }
229:         // returns the account in the desired format
230:         return String.format("
231:             ID: %d
232:             Handle: %s
233:             Description: %s
234:             Post Count: %d
235:             Endorse Count: %d
236:             ", account.getUID(), account.getHandle(),
account.getDescField(), account.getPosts().size(),
237:             endorseCount);
238:     }
239:
240:     @Override
241:     public int createPost(String handle, String message) throws
HandleNotRecognisedException, InvalidPostException {
242:         // verification of message
243:         if (message.length() <= 100 && !(message.isEmpty())) {
244:             // look through list of accounts
245:             for (Account account : accounts) {
246:                 // find account with matching handle
247:                 if (account.getHandle().equals(handle)) {
248:                     // Create a new original post
249:                     Integer pid = posts.size() + 1;
250:                     OriginalPost post = new OriginalPost(message, account,
pid);
251:                     // Add the post to the list of posts in the account that
created the post
252:                     account.addPost(post);
253:                     // Add the post to the list of posts in social media
254:                     this.posts.add(post);
255:                     return pid;
256:                 }
257:             }

```



```

258:             throw new HandleNotRecognisedException("Handle not recognised");
259:         } else {
260:             throw new InvalidPostException("Message is over 100 characters long
OR is empty");
261:         }
262:
263:     }
264:
265:     private void addEndToPost(Post post, Endorsement endorsement) {
266:         if (post instanceof OriginalPost) {
267:             ((OriginalPost) post).addEndorsement(endorsement);
268:         } else {
269:             ((Comment) post).addEndorsement(endorsement);
270:         }
271:     }
272:
273:     @Override
274:     public int endorsePost(String handle, int pid)
275:         throws HandleNotRecognisedException,
PostIDNotRecognisedException, NotActionablePostException {
276:         // look through list of accounts
277:
278:         for (Account account : accounts) {
279:             if (account.getHandle().equals(handle)) {
280:                 // find post that wants to be endorsed using pid
281:                 for (Post post : posts) {
282:                     // check if endorsing an endorsement
283:                     if (post.getPid() == pid) {
284:                         if (!(post instanceof Endorsement)) {
285:                             // Create the endorsement message in
the desired format
286:                             String message = "EP@" +
post.getAuthor().getHandle() + ": " + post.getMessage();
287:                             int endPID = posts.size() + 1;
288:                             Endorsement endorsement = new
Endorsement(message, account, pid, endPID);
289:                             // Append endorsement to list of posts
in social media
290:                             this.posts.add(endorsement);
291:                             // Add endorsement to list of posts of
the account making the endorsement
292:                             account.addPost(endorsement);
293:                             addEndToPost(post, endorsement);
294:                             // check if endorsing an original post or
a comment in order to downcast
295:                             return endorsement.getPid();
296:                         }

```

```

297:                                     throw new
NotActionablePostException("Cannot endorse an endorsement");
298:                                     }
299:                                     }
300:                                     throw new PostIDNotRecognisedException("Post ID not
recognised");
301:                                     }
302:                                     }
303:                                     throw new HandleNotRecognisedException("Handle is not recognised");
304:     }
305:
306: /**
307:  * This method is called exclusively from the commentPost method. It is to avoid
308:  * too many nested loops in one method This is mainly due to needing to downcast
309:  * from a post to either a comment or an originalpost
310:  *
311:  * @param message message of the comment
312:  * @param account account that is commenting on a post
313:  * @param pid    post ID of the post that is being commented on
314:  * @param post   post that is being commented on
315:  * @return post ID of the comment
316:  * @throws NotActionablePostException
317:  */
318: private int cognitiveComplexityReducer(String message, Account account, int pid,
Post post)
319:         throws NotActionablePostException {
320:     if (post instanceof Endorsement) {
321:         throw new NotActionablePostException("Cannot comment on an
endorsement");
322:     }
323:     int comPID = posts.size() + 1;
324:     Comment comment = new Comment(message, account, pid, comPID);
325:     // have to add endorsement to list of posts, account and make it a child of
326:     // other post
327:     this.posts.add(comment);
328:     account.addPost(comment);
329:     // check if endorsing an original post or a comment in order to downcast
330:     // Add comment to arraylist of comments of parent post
331:     if (post instanceof OriginalPost) {
332:         ((OriginalPost) post).addComment(comment);
333:     }
334:     if (post instanceof Comment) {
335:         ((Comment) post).addComment(comment);
336:     }
337:     return comment.getPid();
338:
339: }

```

```

340:
341:  @Override
342:  public int commentPost(String handle, int pid, String message) throws
HandleNotRecognisedException,
343:          PostIDNotRecognisedException, NotActionablePostException,
InvalidPostException {
344:      // test if message is valid
345:      if (!(message.length() <= 100 && !message.isEmpty())) {
346:          throw new InvalidPostException("Message of post is greater than 100
characters OR is empty");
347:      }
348:
349:      boolean found = false;
350:      Account accountCommentor = null;
351:      for (Account account : accounts) {
352:          if (account.getHandle().equals(handle)) {
353:              accountCommentor = account;
354:              found = true;
355:          }
356:      }
357:      // If input handle does not match any existing handle found throw handle not
358:      // recognised exception
359:      if (!(found)) {
360:          throw new HandleNotRecognisedException("Handle not recognised");
361:      }
362:
363:      for (Post post : posts) {
364:          if (post.getPid() == pid) {
365:              return cognitiveComplexityReducer(message,
accountCommentor, pid, post);
366:          }
367:      }
368:      throw new PostIDNotRecognisedException("Post ID not recognised");
369:
370:  }
371:
372:  @Override
373:  public void deletePost(int pid) throws PostIDNotRecognisedException {
374:      boolean found = false;
375:      for (Post post : posts) {
376:          if (post.getPid() == pid) {
377:              // Check what kind of post is being deleted and call respective
function
378:              if (post instanceof OriginalPost) {
379:                  deleteOriginalPost((OriginalPost) post);
380:                  found = true;
381:                  break;

```

```

382:                 } else if (post instanceof Comment) {
383:                     deleteComment((Comment) post);
384:                     found = true;
385:                     break;
386:                 } else if (post instanceof Endorsement) {
387:                     deleteEndorsement((Endorsement) post);
388:                     found = true;
389:                     break;
390:                     // all types are stored in socialmedia.posts
391:                     // account has an arraylist of all posts
392:                 }
393:             }
394:         }
395:         if (!found) {
396:             throw new PostIDNotRecognisedException("Post ID not recognised");
397:         }
398:
399:     }
400:
401:     /**
402:      * Deletes an original post
403:      *
404:      * @param post post that is being deleted
405:      */
406:     private void deleteOriginalPost(OriginalPost post) {
407:         for (int i = 0; i < post.getEndorsements().size(); i++) {
408:             deleteEndorsement(post.getEndorsements().get(i));
409:         }
410:
411:         OriginalPost genPost = (OriginalPost) posts.get(0);
412:         for (Comment comment : post.getComments()) {
413:             /// set the post pointer of any comments that commented on this
post to 1
414:             // a post pointer of 1 corresponds to the generic deleted message
415:             comment.setPostPointer(1);
416:             // add comment to generic post
417:             genPost.addComment(comment);
418:         }
419:         // remove the original post from the account that created it
420:         Account author = post.getAuthor();
421:         author.removePost(post);
422:         // remove the original post from the arraylist of posts in social media
423:         this.posts.remove(post);
424:
425:     }
426:
427:     /**

```

```

428:  * Deletes a comment
429:  *
430:  * @param comment comment that is being deleted
431:  */
432:  private void deleteComment(Comment comment) {
433:      OriginalPost genPost = (OriginalPost) posts.get(0);
434:      for (Comment commentChild : comment.getComments()) {
435:          // set the post pointer of any comments that commented on this post
to 1
436:          // a post pointer of 1 corresponds to the generic message
437:          commentChild.setPostPointer(1);
438:          commentChild.setPostPointer(1);
439:          // add comment to generic post
440:          genPost.addComment(comment);
441:      }
442:      //remove the comment from the account that created it
443:      Account author = comment.getAuthor();
444:      author.removePost(comment);
445:      //remove the comment from the arraylist of posts in social media
446:      this.posts.remove(comment);
447:
448:      //remove comment from the arraylist of comments of the post that it
commented on
449:      for (Post post : posts) {
450:          if (comment.getPostPointer() == post.getPid()) {
451:              if (post instanceof OriginalPost) {
452:                  ((OriginalPost)
post).getComments().remove(comment);
453:                  break;
454:              } else if (post instanceof Comment) {
455:                  ((Comment) post).getComments().remove(comment);
456:                  break;
457:              }
458:          }
459:      }
460:  }
461:
462:  /**
463:   * Deletes an endorsement
464:   * @param endorsement endorsement that is being deleted
465:   */
466:  private void deleteEndorsement(Endorsement endorsement) {
467:
468:      //remove the endorsement from the account that created it
469:      Account author = endorsement.getAuthor();
470:      author.removePost(endorsement);
471:

```

```

472:         //remove the endorsement from the arraylist of posts in social media
473:         this.posts.remove(endorsement);
474:
475:         //remove endorsement from the arraylist of comments of the post that it
commented on
476:         for (Post post : posts) {
477:             if (endorsement.getPostPointer() == post.getPid()) {
478:                 if (post instanceof OriginalPost) {
479:                     ((OriginalPost)
post).getEndorsements().remove(endorsement);
480:                 } else if (post instanceof Comment) {
481:                     ((Comment)
post).getEndorsements().remove(endorsement);
482:                 }
483:             }
484:         }
485:
486:     }
487:
488:     @Override
489:     public String showIndividualPost(int pid) throws PostIDNotRecognisedException {
490:         for (Post post : posts) {
491:             if (post.getPid() == pid) {
492:                 String id = "ID: " + pid + "\n";
493:                 String account = "Account: " + posts.get(pid - 1).getAuthor() +
"\n";
494:                 int numberOfEndorsements;
495:                 int numberOfComments;
496:                 //If post is an endorsement it will have no endorsements or
comments
497:                 if (post instanceof Endorsement) {
498:                     numberOfEndorsements = 0;
499:                     numberOfComments = 0;
500:                 } else if (post instanceof OriginalPost) {
501:                     numberOfEndorsements = ((OriginalPost)
post).getEndorsements().size();
502:                     numberOfComments = ((OriginalPost)
post).getComments().size();
503:                 } else {
504:                     numberOfEndorsements = ((Comment)
post).getEndorsements().size();
505:                     numberOfComments = ((Comment)
post).getComments().size();
506:                 }
507:                 String number = "No. endorsements: " +
numberOfEndorsements + " | No. Comments: " + numberOfComments
508:                     + "\n";

```

```

509:                //combine elements of individual post into one string
510:                // formatting was done in the form of adding new line
characters where relevant
511:                return id + account + number + post.getMessage();
512:            }
513:        }
514:        throw new PostIDNotRecognisedException("Post ID not recognised");
515:    }
516:
517:    /** Appends each individual post to its parent post in order to show post children
details
518:    * Deals with formatting inside the function
519:    * Is recursive and calls itself
520:    * @param indentation the indentation that the post should be displayed at
521:    * @param post the parent post that the child post is being appended to
522:    * @return A post and the details of all its children
523:    * @throws PostIDNotRecognisedException
524:    */
525:    private StringBuilder recursionFunc(Integer indentation, Comment post) throws
PostIDNotRecognisedException {
526:        StringBuilder formatCom = new StringBuilder();
527:        // increment indentation level
528:        indentation++;
529:        // get post message
530:        String message = showIndividualPost(post.getPid());
531:        // indent accordingly
532:        message = message.indent(indentation * 4);
533:        // change first indent to match format
534:        message = message.replaceFirst("  ID", "| > ID");
535:        // if length of arraylist comments is greater than 0, add an extra | underneath
536:        if (!post.getComments().isEmpty()) {
537:            String spaces = " ";
538:            message += spaces.repeat(indentation * 4) + "|\\n";
539:        }
540:
541:        // append message to stringbuilder
542:        formatCom.append(message);
543:
544:        // repeat process for each comment
545:        for (Comment comment : post.getComments()) {
546:            formatCom.append(recursionFunc(indentation, comment));
547:        }
548:        return formatCom;
549:    }
550:
551:    @Override

```

```

552: public StringBuilder showPostChildrenDetails(int id)
553:         throws PostIDNotRecognisedException, NotActionablePostException {
554:
555:         Optional<Post> postOpt;// an optional can be null as well
556:         postOpt = posts.stream().filter(p -> p.getPid() == id).findFirst();// finds the
first (and only) instance a post
557:
// with matching pid
558:         if (!postOpt.isPresent()) { // if it was found
559:             throw new PostIDNotRecognisedException();
560:         }
561:         Post post = postOpt.get();// actually gets the post from the 'optional'
562:         if (post instanceof Endorsement) { // if endorsement
563:             throw new NotActionablePostException();
564:         }
565:
566:         StringBuilder format = new StringBuilder();
567:         format.append(showIndividualPost(id));// print original post/comment
568:
569:         // idc if comment or original post, go through comments and call
recursiveFunc
570:         // to print tree
571:         if (post instanceof OriginalPost) {
572:             if (!((OriginalPost) post).getComments().isEmpty()) {
573:                 format.append("\n|\n");
574:             }
575:             for (Comment comment : ((OriginalPost) post).getComments()) {
576:                 format.append(recursionFunc(0, comment));
577:             }
578:         } else {
579:             if (!((Comment) post).getComments().isEmpty()) {
580:                 format.append("\n|\n");
581:             }
582:             for (Comment comment : ((Comment) post).getComments()) {
583:                 format.append(recursionFunc(0, comment));
584:             }
585:         }
586:
587:         return format;
588:     }
589:
590:     @Override
591:     public int getNumberOfAccounts() {
592:         return accounts.size();
593:     }
594:
595:     @Override

```



```

596: public int getTotalOriginalPosts() {
597:     //If a post is an instance of an original post increase the counter by 1
598:     long counter = posts.stream().filter(p -> p instanceof OriginalPost).count();
599:     return (int) counter;
600: }
601:
602: @Override
603: public int getTotalEndorsmentPosts() {
604:     // DO NOT alter typo in method declaration or the tests wont run properly
605:     //If a post is an instance of an endorsement increase the counter by 1
606:     long counter = posts.stream().filter(p -> p instanceof Endorsement).count();
607:     return (int) counter;
608: }
609:
610: @Override
611: public int getTotalCommentPosts() {
612:     //If a post is an instance of a comment increase the counter by 1
613:     long counter = posts.stream().filter(p -> p instanceof Comment).count();
614:     return (int) counter;
615: }
616:
617: @Override
618: public int getMostEndorsedPost() {
619:     //If there are no posts with endorsements then will return an invalid post ID
of -1
620:     Integer maxEnd = -1;
621:     Integer maxPID = -1;
622:     // For each post check the number of endorsements
623:     // If the post has a larger number than the previous maximum number of
endorsements update the counter and post ID for most endorsed post accordingly
624:     for (Post post : posts) {
625:         if (post instanceof OriginalPost) {
626:             long counter = ((OriginalPost)
post).getEndorsements().stream().count();
627:             if ((int) counter > maxEnd) {
628:                 maxEnd = (int) counter;
629:                 maxPID = post.getPid();
630:             }
631:         }
632:         if (post instanceof Comment) {
633:             long counter = ((Comment)
post).getEndorsements().stream().count();
634:             if ((int) counter > maxEnd) {
635:                 maxEnd = (int) counter;
636:                 maxPID = post.getPid();
637:             }
638:         }

```

```

639:
640:     }
641:     return maxPID;
642: }
643:
644: @Override
645: public int getMostEndorsedAccount() {
646:     //If there are no accounts containing posts with endorsements then will
return an invalid post ID of -1
647:     Integer maxEnd = -1;
648:     Integer maxUID = -1;
649:     for (Account account : accounts) {
650:         //for each account set its endorsement counter to 0
651:         long counter = 0L;
652:         // iterate through each post in an account and add the number of
endorsements to the account counter
653:         for (Post post : account.getPosts()) {
654:             if (post instanceof OriginalPost) {
655:                 counter += ((OriginalPost)
post).getEndorsements().stream().count();
656:             }
657:             if (post instanceof Comment) {
658:                 counter += ((Comment)
post).getEndorsements().stream().count();
659:             }
660:         }
661:         // if the account counter is larger than the largest previous counter,
update the counter and post ID to the new most endorsed account
662:         if ((int) counter > maxEnd) {
663:             maxEnd = (int) counter;
664:             maxUID = account.getUID();
665:         }
666:     }
667:     return maxUID;
668: }
669:
670:
671: @Override
672: public void erasePlatform() {
673:     // Method empties this SocialMediaPlatform of its contents and resets all
674:     // internal counters.
675:     // set lists to empty
676:     this.accounts.clear();
677:     this.posts.clear();
678:
679: }
680:

```

```

681:  @Override
682:  public void savePlatform(String filename) throws IOException {
683:      // Prepend platform content to file filename
684:      try (FileOutputStream file = new FileOutputStream(filename, false);
685:           ObjectOutputStream stream = new ObjectOutputStream(file)) {
686:          Integer length = accounts.size();//
687:
688:          stream.writeObject(length);
689:          for (Account account : accounts) {
690:              stream.writeObject(account);
691:          }
692:
693:          length = posts.size();
694:
695:          stream.writeObject(length);
696:          for (Post post : posts) {
697:              stream.writeObject(post);
698:          }
699:
700:
701:          stream.flush();
702:
703:      }
704:
705:  }
706:
707:  @Override
708:  public void loadPlatform(String filename) throws IOException,
ClassNotFoundException {
709:
710:      try (FileInputStream file = new FileInputStream(filename);
711:           ObjectInputStream stream = new ObjectInputStream(file);) {
712:
713:          Integer length = (Integer) stream.readObject();
714:          for (int i = 0; i < length; i++) {
715:              accounts.add((Account) stream.readObject());
716:          }
717:
718:          length = (Integer) stream.readObject();
719:          for (int j = 0; j < length; j++) {
720:              posts.add((Post) stream.readObject());
721:          }
722:      }
723:  }
724: }

```