DOCUMENT_HEADING

5 files

src\socialmedia\SocialMedia.java
src\socialmedia\Post.java
src\socialmedia\Endorsement.java
src\socialmedia\Comment.java
src\socialmedia\Account.java

src\socialmedia\SocialMedia.java

```java
package socialmedia;

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

/**
 * SocialMedia is an implementor of the SocialMediaPlatform interface.
 * It provides functionality for handling accounts and posts on the system, as
 * well as analytics and system management methods.
 *
 * @author Diogo Pacheco
 * @version 1.0
 */
public class SocialMedia implements SocialMediaPlatform {
    /**
     * List of all active accounts within the system
     */
    private ArrayList<Account> accounts = new ArrayList<>();
    /**
     * List of all posts active within the system (including comments and endorsements)
     */
    private ArrayList<Post> posts = new ArrayList<>();

    @Override
    public int createAccount(String handle) throws IllegalHandleException,
InvalidHandleException {
        // Check handle is valid (not null; no whitespace; <=30 characters)
        if (handle == "") {
            throw new InvalidHandleException("Handle cannot be empty!");
        } else if (handle.matches(".*\\s+.*")) {
            throw new InvalidHandleException("Handle '"+handle+"' contains invalid
characters (whitespace)!");
        } else if (handle.length() > 30) {
            throw new InvalidHandleException("Handle '"+handle+"' is too long (>30
characters)!");
        }

        // Check if handle is unique (doesn't already exist)
        for (int i=0; i < accounts.size(); i++) {
            if (accounts.get(i).getHandle() == handle) {
                throw new IllegalHandleException("Handle '"+handle+"' already exists!");
            }
```

```
        }

        // Create new account object using given handle
        Account newAccount = new Account(handle);
        accounts.add(newAccount);
        return newAccount.getIdentifier();
    }

    @Override
    public int createAccount(String handle, String description) throws IllegalHandleException, InvalidHandleException {
        // Check handle is valid (not null; no whitespace; <=30 characters)
        if (handle == "") {
            throw new InvalidHandleException("Handle cannot be empty!");
        } else if (handle.matches(".*\\s+.*")) {
            throw new InvalidHandleException("Handle '"+handle+"' contains invalid characters (whitespace)!");
        } else if (handle.length() > 30) {
            throw new InvalidHandleException("Handle '"+handle+"' is too long (>30 characters)!");
        }

        // Check if handle is unique (doesn't already exist)
        for (int i=0; i < accounts.size(); i++) {
            if (accounts.get(i).getHandle() == handle) {
                throw new IllegalHandleException("Handle '"+handle+"' already exists!");
            }
        }

        // Create new account object using given handle and description
        Account newAccount = new Account(handle, description);
        accounts.add(newAccount);
        return newAccount.getIdentifier();
    }

    @Override
    public void removeAccount(int id) throws AccountIDNotRecognisedException {
        // Get account with given identifier
        Account account = null;
        for (int i=0; i < accounts.size(); i++) {
            if (accounts.get(i).getIdentifier() == id) {
                account = accounts.get(i);
                accounts.remove(i);
                break;
            }
        }

        // Throw AccountIDNotRecognisedException if account cannot be found with given handle
        if (account == null) {
            throw new AccountIDNotRecognisedException("Un-able to find an account with id: "+id+"!");
        }

        // Get all posts associated with account & delete each one
        ArrayList<Post> accountPosts = account.getPosts();
        for (int i=0; i < accountPosts.size(); i++) {
            Post post = accountPosts.get(i);
            post.delete();

            int postID = post.getIdentifier();
```

```
            for (int j=0; j < posts.size(); j++) {
                if (posts.get(j).getIdentifier() == postID) {
                    posts.remove(j);
                    break;
                }
            }
        }
    }

    @Override
    public void removeAccount(String handle) throws HandleNotRecognisedException {
        // Get account with given handle
        Account account = null;
        for (int i=0; i < accounts.size(); i++) {
            if (accounts.get(i).getHandle() == handle) {
                account = accounts.get(i);
                accounts.remove(i);
                break;
            }
        }

        // Throw HandleNotRecognisedException if account cannot be found with given handle
        if (account == null) {
            throw new HandleNotRecognisedException("Un-able to find account with handle: "+handle+"!");
        }

        // Get all posts associated with account & delete each one
        ArrayList<Post> posts = account.getPosts();
        for (int i=0; i < posts.size(); i++) {
            Post post = posts.get(i);
            post.delete();

            int postID = post.getIdentifier();
            for (int j=0; j < posts.size(); j++) {
                if (posts.get(j).getIdentifier() == postID) {
                    posts.remove(j);
                    break;
                }
            }
        }
    }

    @Override
    public void changeAccountHandle(String oldHandle, String newHandle)
            throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
        // Get account with given handle
        Account account = null;
        for (int i=0; i < accounts.size(); i++) {
            if (accounts.get(i).getHandle() == oldHandle) {
                account = accounts.get(i);
                break;
            }
        }

        // Throw HandleNotRecognisedException if account cannot be found with given handle
        if (account == null) {
            throw new HandleNotRecognisedException("Un-able to find account with handle: "+oldHandle+"!");
        }
```

```
159
160         // Check handle is valid (not null; no whitespace; <=30 characters)
161         if (newHandle == "") {
162             throw new InvalidHandleException("Handle cannot be empty!");
163         } else if (newHandle.matches(".*\\s+.*")) {
164             throw new InvalidHandleException("Handle '"+newHandle+"' contains invalid
characters (whitespace)!");
165         } else if (newHandle.length() > 30) {
166             throw new InvalidHandleException("Handle '"+newHandle+"' is too long (>30
characters)!");
167         }
168
169         // Check if handle is unique (doesn't already exist)
170         for (int i=0; i < accounts.size(); i++) {
171             if (accounts.get(i).getHandle() == newHandle) {
172                 throw new IllegalHandleException("Handle '"+newHandle+"' already
exists!");
173             }
174         }
175         account.setHandle(newHandle);
176     }
177
178     @Override
179     public void updateAccountDescription(String handle, String description) throws
HandleNotRecognisedException {
180         // Get account with given handle
181         Account account = null;
182         for (int i=0; i < accounts.size(); i++) {
183             if (accounts.get(i).getHandle() == handle) {
184                 account = accounts.get(i);
185                 break;
186             }
187         }
188
189         // Throw HandleNotRecognisedException if account cannot be found with given handle
190         if (account == null) {
191             throw new HandleNotRecognisedException("Un-able to find account with handle:
"+handle+"!");
192         }
193         account.setDescription(description);
194     }
195
196     @Override
197     public String showAccount(String handle) throws HandleNotRecognisedException {
198         // Search through the list of post objects, and get post info when account found
199         for (int i=0; i < accounts.size(); i++) {
200             if (accounts.get(i).getHandle() == handle) {
201                 Account account = accounts.get(i);
202                 StringBuilder accountInfo = account.getInfo();
203                 return accountInfo.toString();
204             }
205         }
206
207         // Throw PostIDNotRecognisedException if no post found matching given id
208         throw new HandleNotRecognisedException("Un-able to find account with handle:
"+handle+"!");
209     }
210
211     @Override
212     public int createPost(String handle, String message) throws
HandleNotRecognisedException, InvalidPostException {
```

```
213         // Get account with given handle
214         Account account = null;
215         for (int i=0; i < accounts.size(); i++) {
216             if (accounts.get(i).getHandle() == handle) {
217                 account = accounts.get(i);
218                 break;
219             }
220         }
221 
222         // Throw HandleNotRecognisedException if account cannot be found with given handle
223         if (account == null) {
224             throw new HandleNotRecognisedException("Un-able to find account with handle:
"+handle+"!");
225         }
226 
227         // Checks to see if the post is valid.
228         if (message.length() > 100) {
229             throw new InvalidPostException("The post message is too long! (>100
characters)");
230         } else if (message.length() < 1) {
231             throw new InvalidPostException("The post message is too short! (Cannot be
empty!)");
232         }
233 
234         // Create new post
235         Post newPost = account.createPost(message);
236         posts.add(newPost);
237         return newPost.getIdentifier();
238     }
239 
240     @Override
241     public int endorsePost(String handle, int id) throws HandleNotRecognisedException,
PostIDNotRecognisedException, NotActionablePostException {
242 
243         // Get account with given handle
244         Account account = null;
245         for (int i=0; i < accounts.size(); i++) {
246             if (Objects.equals(accounts.get(i).getHandle(), handle)) {
247                 account = accounts.get(i);
248                 break;
249             }
250         }
251 
252         // Throw HandleNotRecognisedException if account cannot be found with given handle
253         if (account == null) {
254             throw new HandleNotRecognisedException("Un-able to find account with handle:
"+handle+"!");
255         }
256         else {
257             // Check post is actionable and exists in the system
258             for (int i=0; i < posts.size(); i++) {
259                 if (posts.get(i).getIdentifier() == id) {
260                     if (posts.get(i) instanceof Endorsement) {
261                         throw new NotActionablePostException("Un-able to endorse a
endorsement");
262                     }
263 
264                     // Create new endorsement
265                     Endorsement newEndorsement = account.createEndorsement(posts.get(i));
266                     posts.add(newEndorsement);
267                     return newEndorsement.getIdentifier();
```

```
                    }
                }
            }
            // Throw PostIDNotRecognisedException if no post found with matching id
            throw new PostIDNotRecognisedException("Un-able to find post with id: "+id+"!");
    }

    @Override
    public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {

        // Checks to see if the post is valid.
        if (message.length() > 100) {
            throw new InvalidPostException("The post message is too long! (>100 characters)");
        } else if (message.length() < 1) {
            throw new InvalidPostException("The post message is too short! (Cannot be empty!)");
        }

        // Get account with given handle
        Account account = null;
        for (int i=0; i < accounts.size(); i++) {
            if (Objects.equals(accounts.get(i).getHandle(), handle)) {
                account = accounts.get(i);
                break;
            }
        }

        // Throw HandleNotRecognisedException if account cannot be found with given handle
        if (account == null) {
            throw new HandleNotRecognisedException("Un-able to find account with handle: "+handle+"!");
        } else {
            // Check post is actionable and exists in the system
            for (int i=0; i < posts.size(); i++) {
                if (posts.get(i).getIdentifier() == id) {
                    if (posts.get(i) instanceof Endorsement) {
                        throw new NotActionablePostException("Un-able to endorse a endorsement");
                    }
                    // Create new comment
                    Comment newComment = account.createComment(posts.get(i), message);
                    posts.add(newComment);
                    return newComment.getIdentifier();
                }
            }
            // Throw PostIDNotRecognisedException if no post found with matching id
            throw new PostIDNotRecognisedException("Un-able to find post with id: "+id+"!");
        }
    }

    @Override
    public void deletePost(int id) throws PostIDNotRecognisedException {
        // Get post with matching id
        Post post = null;
        for (int i=0; i < posts.size(); i++)
            if (posts.get(i).getIdentifier() == id) {
                post = posts.get(i);
```

```
322                 break;
323             }
324 
325         // Throw PostIDNotRecognisedException if no post found with matching id
326         if (post == null) {
327             throw new PostIDNotRecognisedException("Un-able to find post with id:
"+id+"!");
328         }
329 
330         // Go through posts endorsements and remove each of them from the system
331         ArrayList<Endorsement> endorsements = post.getEndorsements();
332         for (int i=0; i < endorsements.size(); i++) {
333             int endorsementID = endorsements.get(i).getIdentifier();
334             for (int j=0; j < posts.size(); j++) {
335                 if (posts.get(j).getIdentifier() == endorsementID) {
336                     posts.remove(j);
337                     break;
338                 }
339             }
340         }
341 
342         // Tell post to delete and remove it from the system.
343         post.delete();
344         for (int i=0; i < posts.size(); i++) {
345             if (posts.get(i).getIdentifier() == id) {
346                 posts.remove(i);
347                 break;
348             }
349         }
350     }
351 
352     @Override
353     public String showIndividualPost(int id) throws PostIDNotRecognisedException {
354         // Find post with matching id and get posts formatted information
355         for (int i=0; i < posts.size(); i++) {
356             if (posts.get(i).getIdentifier() == id) {
357                 Post post = posts.get(i);
358                 StringBuilder postInfo = post.getInfo(0);
359                 return postInfo.toString();
360             }
361         }
362 
363         // Throw PostIDNotRecognisedException if no post found matching given id
364         throw new PostIDNotRecognisedException("Un-able to find post with id: "+id+"!");
365     }
366 
367     @Override
368     public StringBuilder showPostChildrenDetails(int id) throws
PostIDNotRecognisedException, NotActionablePostException {
369         // Get post with matching id
370         Post post = null;
371         for (int i=0; i < posts.size(); i++) {
372             if (posts.get(i).getIdentifier() == id) {
373 
374                 // Check if post is actionable, ie: not an endorsement
375                 if (posts.get(i) instanceof Endorsement) {
376                     throw new NotActionablePostException("Un-able to
showPostChildrenDetails of a endorsement");
377                 } else {
378                     post = posts.get(i);
379                     break;
```

```
                }
            }
        }

        // Throw PostIDNotRecognisedException if no post found matching given id
        if (post == null) {
            throw new PostIDNotRecognisedException("Un-able to find post with id: "+id+"!");
        }

        // Create string builder by calling recursive child info function of root post
        StringBuilder details = new StringBuilder();
        post.getChildInfo(details, 0);
        return details;
    }

    @Override
    public int getNumberOfAccounts() {
        return accounts.size();
    }

    @Override
    public int getTotalOriginalPosts() {
        // Count all posts of type Post (and not comments or endorsements) in posts array
        int count = 0;
        for (int i=0; i < posts.size(); i++) {
            if (!(posts.get(i) instanceof Comment) && !(posts.get(i) instanceof Endorsement)) {
                count += 1;
            }
        }
        return count;
    }

    @Override
    public int getTotalEndorsmentPosts() {
        // Count all posts of type Endorsement in posts array
        int count = 0;
        for (int i=0; i < posts.size(); i++) {
            if (posts.get(i) instanceof Endorsement) {
                count += 1;
            }
        }
        return count;
    }

    @Override
    public int getTotalCommentPosts() {
        // Count all posts of type Comment in posts array
        int count = 0;
        for (int i=0; i < posts.size(); i++) {
            if (posts.get(i) instanceof Comment) {
                count += 1;
            }
        }
        return count;
    }

    @Override
    public int getMostEndorsedPost() {
```

```
438         // Check each post in posts array and keep track of the most endorsed post and
    it's endorsement count
439         Post topPost = null;
440         int topCount = 0;
441         for (int i=0; i < posts.size(); i++) {
442             ArrayList<Endorsement> endorsements = posts.get(i).getEndorsements();
443             if (endorsements.size() >= topCount) {
444                 topPost = posts.get(i);
445                 topCount = endorsements.size();
446             }
447         }
448 
449         // Return 0 if no posts in system
450         if (topPost == null) {
451             return 0;
452         } else {
453             return topPost.getIdentifier();
454         }
455     }
456 
457     @Override
458     public int getMostEndorsedAccount() {
459         // Check each account and keep track of the most endorsed account and it's
    endorsement count
460         Account topAccount = null;
461         int topCount = 0;
462         for (int i=0; i < accounts.size(); i++) {
463             if (accounts.get(i).getTotalEndorsments() >= topCount) {
464                 topAccount = accounts.get(i);
465                 topCount = topAccount.getTotalEndorsments();
466             }
467         }
468 
469         // Return 0 if no accounts in system
470         if (topAccount == null) {
471             return 0;
472         } else {
473             return topAccount.getIdentifier();
474         }
475     }
476 
477     @Override
478     public void erasePlatform() {
479         // Reset counters for identifiers in account and post classes
480         Account.setCount(0);
481         Post.setCount(0);
482 
483         // Remove all stored posts and accounts from the system
484         accounts.clear();
485         posts.clear();
486     }
487 
488     @Override
489     public void savePlatform(String filename) throws IOException {
490         try {
491             // Create ObjectOutputStream for given filepath (creates file if one not
    found)
492             FileOutputStream file = new FileOutputStream(filename);
493             ObjectOutputStream out = new ObjectOutputStream (file);
494 
495             // Write accounts and posts arrays into file
```

```
            out.writeObject(accounts);
            out.writeObject(posts);

            // Properly close output streams once finished
            out.close();
            file.close();

        // Pass through exceptions if thrown during file write process
        } catch(IOException e) {
            throw e;
        }
    }

    @Override
    public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
        try {
            // Create ObjectInputStream for given filename
            FileInputStream file = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file);

            // Load accounts and posts array objects from serialised file
            accounts = (ArrayList<Account>)in.readObject();
            posts = (ArrayList<Post>)in.readObject();

            // Set the count in Account & Post classes to largest current identifier just
loaded
            Account lastAccount = accounts.get(accounts.size() - 1);
            Account.setCount(lastAccount.getIdentifier());

            Post lastPost = posts.get(posts.size() - 1);
            Post.setCount(lastPost.getIdentifier());

            // Properly close input streams once finished
            in.close();
            file.close();

        // Pass through exceptions if thrown during file read process
        } catch(IOException e) {
            throw e;
        } catch(ClassNotFoundException e) {
            throw e;
        }
    }

}

```

src\socialmedia\Post.java

```
package socialmedia;

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

/**
 * Post class representing a post within the system. Each post is associated with
 * an account that owns the post. Posts have incremental, unique identifiers, starting
 * from 1, and keep track of their comments and endorsements.
```

```
 *
 * @author Sam Townley and Charles Symonds
 * @version 1.0
 */
public class Post implements Serializable {
    /**
     * Static count to keep track of highest identifier (or total accounts created)
     */
    private static int count = 0;
    /**
     * Identifier representing the post
     * <p>
     * Takes the value of count+1 when new account created
     */
    protected int identifier;
    /**
     * The message content of the post
     */
    protected String message;
    /**
     * The account object representing the author of the post
     */
    protected Account author;
    /**
     * List of comment objects that the post has recieved
     */
    private ArrayList<Comment> comments = new ArrayList<>();
    /**
     * List of endorsement objects that the post has recieved
     */
    private ArrayList<Endorsement> endorsements = new ArrayList<>();

    /**
     * Constructs a new post on the system, associated with the given account.
     * <p>
     * Assumes the message is a valid string
     *
     * @param message the message to be used on the comment
     * @param account the author of the comment
     */
    public Post(Account account, String message) {
        this.identifier = ++count;
        this.message = message;
        this.author = account;
    }

    /**
     * The method creates a string showing clearly the state of the attributes of the post
class.
     *
     * @return a String detailing posts attributes.
     */
    @Override
    public String toString() {
        return "Post[id="+identifier+", Message="+message+", Author="+author.getHandle()+"]"
    }

    /**
     * The method sets the post class's counter to the given n value, meaning future
     * identifiers will be set as n+1, and so on
```

```
     * <p>
     * Using n=0 is the same as reseting the counter as if no posts exist yet
     * Therfore you must be careful not to duplicate identifiers when using this method!
     *
     * @param n the number to start setting new identifers from
     */
    public static void setCount(int n) {
        count = n;
    }

    /**
     * The method gets the value of the private int identifier for the class.
     *
     * @return a int identifier for the post
     */
    public int getIdentifier() {
        return identifier;
    }

    /**
     * The method gets the author of the post.
     *
     * @return an account object representing the author
     */
    public Account getAuthor() {
        return author;
    }

    /**
     * The method gets the array of all endorsements the post has.
     *
     * @return an arrayList of Endorsements
     */
    public ArrayList<Endorsement> getEndorsements() {
        return endorsements;
    }

    /**
     * The method gets the message content of the post.
     *
     * @return the posts message string
     */
    public String getMessage() {
        return message;
    }

    /**
     * The method deletes the post from the system, replacing it with a placeholder indicati
that
     * the content has been deleted.
     */
    public void delete() {
        // Make all comments orphans (remove the link to this post)
        for (int i = 0; i < comments.size(); i++) {
            comments.get(i).removeOriginalPost();
        }

        // Reset posts attributes and get author to remove post from it's array records
        endorsements.clear();
        author.deletePost(identifier);
```

```
128         author = null;
129         message = "The original content was removed from the system and is no longer
    available.";
130     }
131 
132     /**
133      * The method adds a new endorsement to the endorsements arraylist
134      *
135      * @param endorsement the endorsement object to add
136      */
137     public void addEndorsement(Endorsement endorsement) {
138         endorsements.add(endorsement);
139     }
140 
141     /**
142      * The method adds a new comment to the comments arraylist
143      *
144      * @param comment the comment object to add
145      */
146     public void addComment(Comment comment) {
147         comments.add(comment);
148     }
149 
150     /**
151      * The method returns the post information in a format ready to be displayed to the user
152      *
153      * @param indent the amount of indentation required infront of the information
154      *
155      * @return the StringBuilder object containing the formatted information
156      */
157     public StringBuilder getInfo(int indent) {
158         // Create Indentation String
159         String indentation = "";
160         for (int i=0; i<indent; i++) {
161             indentation += "        ";
162         }
163 
164         // Format post information into format provided by specification
165         StringBuilder info = new StringBuilder();
166         info.append("ID: ").append(identifier);
167         info.append("\n"+indentation).append("Account: ").append(author.getHandle());
168         info.append("\n"+indentation).append("No. endorsements: ").append(endorsements.size(
169         info.append(" | No. comments: ").append(comments.size());
170         info.append("\n"+indentation).append(message);
171         return info;
172     }
173 
174     /**
175      * The method returns the post information for all child posts (comments) for the currer
    post.
176      * <p>
177      * Designed recursively, so will keep calling itself over and over again until all comme
    have
178      * been added to the stringBuilder
179      *
180      * @param postInfo the current stringBuilder that's been constructed so far
181      * @param indent the amount of indentation required infront of the information
182      */
183     public void getChildInfo(StringBuilder postInfo, int indent) {
184         // Create Indentation String
185         String indentation = "";
```

```
        for (int i=0; i<indent; i++) {
            indentation += "        ";
        }

        // Add the post information for the comment, with the indentation added before hand
not the first post
        if (indent != 0) {

postInfo.append("\n").append(indentation).append("|").append("\n").append(indentation).apper
> ").append(getInfo(indent));
        } else {
            postInfo.append(getInfo(indent));
        }
        ++indent;

        // For each comment, recursively call getChildInfo for that comment with an indent c
higher than the current post
        if (comments.size() > 0) {
            for (int j=0; j < comments.size(); j++) {
                comments.get(j).getChildInfo(postInfo, indent);
            }
        }
        --indent;
    }
}

```

src\socialmedia\Endorsement.java

```
package socialmedia;

/**
 * Endorsements class enherits from the Post class as they share similar properties. Unlike
a post though,
 * the comments and endorsements arrayList will always be empty. Similar to a comment, an
 * endorsement keeps track of the identifier of the original post associated with the
endorsement.
 *
 * @author Sam Townley and Charles Symonds
 * @version 1.0
 */
public class Endorsement extends Post {
    /**
     * The identifier of the parent post thats being endorsed
     */
    private int originalPostID;

    /**
     * Constructs a new endorsement on a post/comment within the system.
     * <p>
     * Assumes the id points to an existing post/comment that allows an endorsement
     * Assumes the message is a valid string
     *
     * @param account the author of the comment
     * @param id the int identifier of the post being commented on
     * @param message the message to be used on the comment
     */
    public Endorsement(Account account, int id, String message) {
```

```
28          super(account, message);
29          this.originalPostID = id;
30      }
31
32      /**
33       * The method creates a string showing clearly the state of the attributes of the
endorsement class.
34       *
35       * @return a String detailing endorsement's attributes.
36       */
37      @Override
38      public String toString() {
39          return "Endorsement[id="+identifier+", Message="+message+",
Author="+author.getHandle()+", Original_Post_ID="+originalPostID+"]";
40      }
41  }
42
```

src\socialmedia\Comment.java

```
1   package socialmedia;
2
3   /**
4    * Comment class inherts attributes from the Post class, as comments and posts share
similar
5    * properties. However, a comment also keeps track of the identifier of the original post
6    * associated with the comment.
7    *
8    * @author Sam Townley and Charles Symonds
9    * @version 1.0
10   */
11  public class Comment extends Post {
12      /**
13       * The identifier of the parent post that this comment belongs to
14       */
15      private int originalPostID;
16
17      /**
18       * Constructs a new comment on a post within the system.
19       * <p>
20       * Assumes the id points to an existing post that allows comments
21       * Assumes the message is a valid string
22       *
23       * @param id the int identifier of the post being commented on
24       * @param message the message to be used on the comment
25       * @param account the author of the comment
26       */
27      public Comment(int id, String message, Account account) {
28          super(account, message);
29          this.originalPostID = id;
30      }
31
32      /**
33       * The method creates a string showing clearly the state of the attributes of the
comment class.
34       *
35       * @return a String detailing comment's attributes.
36       */
37      @Override
```

```
    public String toString() {
        return "Comment[id="+identifier+", Message="+message+",
Author="+author.getHandle()+", Original_Post_ID="+originalPostID+"]";
    }

    /**
     * The method makes the comment an orphan, by removing its link with the parent post.
     * <p>
     * Used when the post being commented on has been removed from the system.
     */
    public void removeOriginalPost() {
        // -1 indicates that the comment is an orphan
        originalPostID = -1;
    }
}

```

src\socialmedia\Account.java

```
package socialmedia;

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


/**
 * Account class representing an account on the SocialMediaPlatform.
 * Has two constructors, one for supplying both a description and a handle
 * and the other for just a handle. The identifier is linearly generated as to be
 * unique.
 * Also provides functionality for managing account's posts within the platform.
 *
 * @author Sam Townley and Charles Symonds
 * @version 1.0
 */
public class Account implements Serializable {
    /**
     * Static count to keep track of highest identifier (or total accounts created)
     */
    private static int count = 0;
    /**
     * Identifer for the account
     * <p>
     * Takes the value of count+1 when new account created
     */
    private int identifier;
    /**
     * Unique handle chosen by the account holder to represent their account
     */
    private String handle;
    /**
     * Description chosen by account holder to be displayed with the account.
     * <p>
     * Defaults to an empty string
     */
    private String description = "";
    /**
     * ArrayList keeping track of all Post objects created by the account
```

```
40       */
41      private ArrayList<Post> posts = new ArrayList<Post>();
42
43      /**
44       * The method creates a string showing clearly the state of the attributes of the
    account class.
45       *
46       * @return a String detailing account attributes.
47       */
48      @Override
49      public String toString() {
50          return "Account[id="+identifier+", handle="+handle+",
    description="+description+"]";
51      }
52
53      /**
54       * The method sets the account class's counter to the given n value, meaning future
55       * identifiers will be set as n+1, and so on
56       * <p>
57       * Using n=0 is the same as reseting the counter as if no accounts exist
58       * Therefore you must be careful not to duplicate identifiers when using this method!
59       *
60       * @param n the number to start setting new identifers from
61       */
62      public static void setCount(int n) {
63          count = n;
64      }
65
66      /**
67       * The method gets the value of the private int identifier for the class.
68       *
69       * @return a int identifier for the account
70       */
71      public int getIdentifier() {
72          return identifier;
73      }
74
75      /**
76       * The method gets the accounts handle.
77       *
78       * @return a String handle used to identify the account
79       */
80      public String getHandle() {
81          return handle;
82      }
83
84      /**
85       * The method gets the accounts description. Will return "" if no description
86       * has been set.
87       *
88       * @return a String description for the account
89       */
90      public String getDescription() {
91          return description;
92      }
93
94      /**
95       * The method gets the value of the private int identifier for the class.
96       *
97       * @return a int identifier for the account
```

```
98       */
99      public ArrayList<Post> getPosts() {
100         return posts;
101     }
102
103     /**
104      * The method sets the account handle to the new String specified.
105      * <p>
106      * Assumes the given handle is already unique, and has been checked
107      * beforehand by the SocialMedia class.
108      *
109      * @param h the new handle for the account
110      */
111     public void setHandle(String h) {
112         handle = h;
113     }
114
115     /**
116      * The method sets the account description to the new String specified.
117      *
118      * @param d the new description for the account
119      */
120     public void setDescription(String d) {
121         description = d;
122     }
123
124     /**
125      * The method removes the account's post that matches the given ID
126      *
127      * @param id the int identifier for the post that needs to be removed
128      */
129     public void deletePost(int id) {
130         // Search through posts until a post with matching id is found, then delete the
    post
131         for (int i=0; i < posts.size(); i++) {
132             if (posts.get(i).getIdentifier() == id) {
133                 posts.remove(i);
134                 break;
135             }
136         }
137     }
138
139     /**
140      * The method creates a new post associated with this account, given a message
141      * <p>
142      * Assumes the given message is a valid String, that has already been checked
143      * against the requirements for a post
144      *
145      * @param message the String message to be used on the post
146      *
147      * @return the Post object that has just been created
148      */
149     public Post createPost(String message) {
150         Post newPost = new Post(this, message);
151         posts.add(newPost);
152         return newPost;
153     }
154
155     /**
```

```
156      * The method creates a new comment associated with this account, given a message and
         original post to comment on
157      * <p>
158      * Assumes the given message is a valid String, that has already been checked
159      * against the requirements for a post and that the given post is one which allows
         comments
160      *
161      * @param post a Post object representing the post that is being commented on
162      * @param message the String message to be used on the comment
163      *
164      * @return the Comment object that has just been created
165      */
166     public Comment createComment(Post post, String message) {
167         Comment newComment = new Comment(post.getIdentifier(), message, this);
168         post.addComment(newComment);
169         return newComment;
170     }
171
172     /**
173      * The method creates a new endorsement associated with this account, given a post to
         endorse
174      * <p>
175      * Assumes that the given post is valid and one which allows endorsements
176      *
177      * @param post a Post object representing the post that is being endorsed
178      *
179      * @return the Endorsement object that has just been created
180      */
181     public Endorsement createEndorsement(Post post) {
182         Endorsement newEndorsement = new Endorsement(this, post.getIdentifier(),
         post.getMessage());
183         post.addEndorsement(newEndorsement);
184         return newEndorsement;
185     }
186
187     /**
188      * The method counts the total number of endorsements accociated with the account
189      *
190      * @return the int count of endorsements
191      */
192     public int getTotalEndorsments() {
193         // For each post in account, get number of endorsements, and add to total count
194         int total = 0;
195         for (int i=0; i<posts.size(); i++) {
196             ArrayList<Endorsement> endorsements = posts.get(i).getEndorsements();
197             total += endorsements.size();
198         }
199         return total;
200     }
201
202     /**
203      * The method returns the account information in a format ready to be displayed to the
         user
204      *
205      * @return the StringBuilder object containing the formatted information
206      */
207     public StringBuilder getInfo() {
208         int endorsementCount = getTotalEndorsments();
209         // Format account information into format provided by specification
210         StringBuilder info = new StringBuilder();
211         info.append("ID: ").append(identifier);
```

```
212         info.append("\nHandle: ").append(handle);
213         info.append("\nDescription: ").append(description);
214         info.append("\nPost count: ").append(posts.size());
215         info.append("\nEndorse count: ").append(endorsementCount);
216         return info;
217     }
218 
219     /**
220      * Constructs a new account instance with a pre-defined handle and account description
221      * <p>
222      * Assumes the given handle has already been verified as unique by SocialMedia
223      * Assumes the description provided has already been validated
224      *
225      * @param handle the handle to be used for the account
226      * @param description the String to set as the account's description
227      */
228     public Account(String handle, String description) {
229         this.handle = handle;
230         this.description = description;
231         this.identifier = ++count;
232     }
233 
234     /**
235      * Constructs a new account instance with a pre-defined handle only
236      * <p>
237      * Assumes the given handle has already been verified as unique by SocialMedia
238      * Account created will have a blank description string
239      *
240      * @param handle the handle to be used for the account
241      */
242     public Account(String handle) {
243         this.handle = handle;
244         this.identifier = ++count;
245     }
246 }
247 
```