# 1 Account.java

```java
package socialmedia;

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


/**
 * The account class is used to create objects that represent users that can post to the social media
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
public class Account implements Serializable{

    /**
     * A public {@link Integer} representing the number of accounts on the social media platform
     */
    public static int numberAccounts = 0;

    /**
     * A public {@link Integer} representing the current id for a new account
     */
    public static int currentId = 0;

    /**
     * A private {@link String} used to store the handle of an account this is unique to the account
     */
    private String handle;
    /**
     * A private constant {@link Integer} used to store the id of an account this is unique to the account
     */
    private final int ID;
    /**
     * A private {@link String} used to store the description of the account
     */
    private String description;
    /**
     * A private {@link Integer} used to store the total number of endorsements from the accounts posts
     */
    private int numberOfEndorsements = 0;
    /**
     * A private {@link ArrayList} storing elements of type {@link Post} that represents the posts the
         account has made
     */
    private ArrayList<Post> posts = new ArrayList<Post>();


    /**
     * This constructor makes an Account with a given handle
     * @param handle The handle which will be given to the account
     * @throws InvalidHandleException when trying to assign an invalid handle
     */
```

```
52      public Account(String handle) throws InvalidHandleException {
53          // A pre condition checks if the given handle is valid
54          isValidHandle(handle);
55
56          // The accounts handle, id and description are set
57          this.handle = handle;
58          this.ID = currentId;
59          this.description = "";
60
61          // Then the counters are incremented by 1
62          currentId++;
63          numberAccounts++;
64      }
65
66      /**
67       * This constructor makes an Account with a given handle and description
68       * @param handle The handle which will be given to the account
69       * @param description The description which will be given to the account
70       * @throws InvalidHandleException when trying to assign an invalid handle
71       */
72      public Account(String handle, String description) throws InvalidHandleException{
73          // A pre-condition checks if the given handle is valid
74          isValidHandle(handle);
75
76          // The accounts handle, id and description are set
77          this.handle = handle;
78          this.ID = currentId;
79          this.description = description;
80
81          // Then the counters are incremented by 1
82          currentId++;
83          numberAccounts++;
84      }
85
86      /**
87       * Gets the handle of an account
88       * @return the handle of an account
89       */
90      public String getHandle(){
91          return this.handle;
92      }
93
94      /**
95       * Sets the handle of an account
96       * @param handle the new handle of an account to be set
97       * @throws InvalidHandleException when trying to assign an invalid handle
98       */
99      public void setHandle(String handle) throws InvalidHandleException {
100         // This pre condition checks if the given handle is valid
101       isValidHandle(handle);
102         this.handle = handle;
103     }
104
105     /**
106      * Gets the id of an account
```

```
107         *  @return the id of an account
108         */
109        public int getId(){
110            return this.ID;
111        }
112
113        /**
114         * Gets the description of an account
115         * @return the description of an account
116         */
117        public String getDescription(){
118            return this.description;
119        }
120
121        /**
122         * Sets the description of an account
123         *  @param description the new description to be set
124         */
125        public void setDescription(String description) {
126            this.description = description;
127        }
128
129        /**
130         * Gets the number of endorsements an account has received on posts
131         * @return the number of endorsements an account has received on posts
132         */
133        public int getEndorsements(){
134            return this.numberOfEndorsements;
135        }
136
137        /**
138         * Increments the number of endorsements an account has received on posts
139         */
140        public void incrementEndorsements(){
141            this.numberOfEndorsements++;
142        }
143
144        /**
145         * Decrements the number of endorsements and account has received on posts
146         */
147        public void decrementEndorsements(){
148            this.numberOfEndorsements--;
149        }
150
151        /**
152         * Deletes an account
153         */
154        public void delete() {
155            // A new variable is created for asserting the account was deleted
156            int originalNumberOfAccounts = numberAccounts;
157
158            // The for loops goes through the accounts posts and calls their delete method
159            while(!posts.isEmpty()){
160                Post post = posts.get(0);
161                post.delete();
```

```
162         }
163 
164         // The number of accounts is decremented by 1
165         numberAccounts--;
166 
167         // Assertion checks that the post condition is met and if it is not met then it throws an exception
168         assert (originalNumberOfAccounts-1==numberAccounts):"Account not deleted successfully";
169     }
170 
171     /**
172      * Gets the list of posts that an account has posted
173      * @return the posts an account has posted
174      */
175     public ArrayList<Post> getPosts(){
176         return this.posts;
177     }
178 
179     /**
180      * Resets the counters for number of account and current id to their initial values
181      */
182     static public void resetCounters(){
183         numberAccounts = 0;
184         currentId = 0;
185     }
186 
187     /**
188      * Adds a given post to an accounts list of posts
189      * @param post the post to be added to an accounts list of posts
190      */
191     public void addPost(Post post){
192         posts.add(post);
193     }
194 
195     /**
196      * Removes a post from an accounts list of posts
197      * @param post the post to be removed from the list of posts
198      */
199     public void removePost(Post post){
200         // The index of the post to be removed is found using a for loop
201         int index = -1;
202         for(int i = 0; i < posts.size(); i++){
203             if(posts.get(i) == post){
204                 index = i;
205             }
206         }
207 
208         // Assertion checks that the post condition is met and if it is not met then it throws an exception
209         assert (index>=0):"Post not removed successfully";
210 
211         // The post is removed using this ID
212         posts.remove(index);
213     }
214     /**
215      * This method returns a string of information about the account
216      * @return the string of information about the account
```

```
     */
    @Override
    public String toString(){

        // A new StringBuilder is created and then information about the account is appended to it
        StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append("ID: ").append(ID).append("\n");
      stringBuilder.append("Handle: ").append(handle).append("\n");
      stringBuilder.append("Description: ").append(description).append("\n");
      stringBuilder.append("Post count: ").append(posts.size()).append("\n");
      stringBuilder.append("Endorse count: ").append(numberOfEndorsements);

        // The information is returned as a string
        return stringBuilder.toString();
    }
    /**
     * This method checks if the handle is valid
     * @param handle the handle being checked
     * @return true if the handle is valid
     * @throws InvalidHandleException when trying to assign an invalid handle
     */
    private boolean isValidHandle(String handle) throws InvalidHandleException{
        // This if statement check for is the new handle is empty or has more than 30 characters or contains
            whitespaces, if it is then the exception is thrown
      if(handle.equals("") || handle.length() > 30 || handle.contains(" ")){
         throw new InvalidHandleException("Handle must not be empty, be shorter than 30 characters and not
             have any whitespaces");
      }
        return true;
    }
}
```

# 2 ActionablePost.java

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

/**
 * The ActionablePost class provides an interface which allows actionable posts to be created easily
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
public abstract class ActionablePost extends Post {
    /**
     * A protected list {@link ArrayList} containing elements of type {@link Endorsement} used to store the
         endorsements of a post
     */
    protected ArrayList<Endorsement> endorsements = new ArrayList<Endorsement>();

     /**
     * A protected list {@link ArrayList} containing elements of type {@link Comment} used to store the
         comment of a post
     */
```

```
19      protected ArrayList<Comment> comments = new ArrayList<Comment>();
20
21      /**
22       * This method returns a string of information about the Individual post with a given indent
23       * @param indent the number of indents needed
24       * @return the string of information about the post
25       */
26      private String toString(int indent){
27          // A pre condition checks if the given indent is valid
28          if(indent < 0){
29              throw new IllegalArgumentException("Indent cannot be less than 0");
30          }
31
32          // A new StringBuilder is created and then information about the Individual post is appended to it
                with an indent
33          StringBuilder stringBuilder = new StringBuilder();
34        stringBuilder.append(" ".repeat((indent - 1) * 4) + "| > ").append("ID: ").append(ID).append("\n");
35        stringBuilder.append(" ".repeat(indent * 4)).append("Account:
              ").append(poster.getHandle()).append("\n");
36        stringBuilder.append(" ".repeat(indent * 4)).append("No. endorsements:
              ").append(getNumberOfEndorsements()).append(" | ");
37        stringBuilder.append("No. comments: ").append(comments.size()).append("\n");
38        stringBuilder.append(" ".repeat(indent * 4)).append(message);
39
40          // The information is returned as a string
41          return stringBuilder.toString();
42      }
43
44      /**
45       * This method recursively generates a string builder with details of the current posts and its children
            posts
46       * @param indent the indent to be applied to the details of the children
47       * @return A string builder of the children posts
48       */
49      public StringBuilder showChildren(int indent){
50          // A pre condition checks if the given indent is valid
51          if(indent < 0){
52              throw new IllegalArgumentException("Indent cannot be less than 0");
53          }
54
55          // A new string builder is instantiated
56          StringBuilder stringBuilder = new StringBuilder();
57
58          // If this is the current post then no indent is needed
59          if(indent == 0){
60              stringBuilder.append(toString());
61          }
62          // Otherwise an indent is applied to the details
63          else{
64              stringBuilder.append(toString(indent));
65          }
66          stringBuilder.append("\n");
67
68          // This for loop iterates through the comments of the post
69          for(int i = 0; i < comments.size(); i++){
```

```
70              // An index and and a bar is added then the details about the comments are added to the string
                    builder
71              stringBuilder.append("  ".repeat(indent)+"|\n");
72              stringBuilder.append(comments.get(i).showChildren(indent + 1).toString());
73          }
74          // Finally the string builder is returned
75          return stringBuilder;
76      }
77
78      @Override
79      public String toString(){
80          // A new StringBuilder is created and then information about the Individual post is appended to it
81          StringBuilder stringBuilder = new StringBuilder();
82        stringBuilder.append("ID: ").append(ID).append("\n");
83        stringBuilder.append("Account: ").append(poster.getHandle()).append("\n");
84        stringBuilder.append("No. endorsements: ").append(getNumberOfEndorsements()).append(" | ");
85        stringBuilder.append("No. comments: ").append(comments.size()).append("\n");
86        stringBuilder.append(message);
87
88          // The information is returned as a string
89          return stringBuilder.toString();
90      }
91
92      /**
93       * This method gets the number of endorsements on the post
94       * @return the number of endorsements returned
95       */
96      public int getNumberOfEndorsements(){
97          return endorsements.size();
98      }
99
100     /**
101      * This method adds an endorsement to the endorsements list
102      * @param endorsement the endorsement to be added
103      */
104     public void addEndorsement(Endorsement endorsement){
105         endorsements.add(endorsement);
106         assert (endorsements.contains(endorsement)) : "Endorsement not added successfully";
107     }
108     /**
109      * This method adds a comment to the comments list
110      * @param comment the comment to be added
111      */
112     public void addComment(Comment comment){
113         comments.add(comment);
114         assert (comments.contains(comment)) : "Comment not added successfully";
115     }
116     /**
117      * This method removes an endorsement from the endorsements list
118      * @param endorsement the endorsement to be removed
119      */
120     public void removeEndorsement(Endorsement endorsement){
121         // this for loop iterates through each element in the endorsements list
122         int index = -1;
123         for(int i = 0; i < endorsements.size(); i++){
```

```
124             // this if statement gets the index of the endorsement by comparing the ith element in the
                    endorsement list
125             if(endorsements.get(i) == endorsement){
126                 index = i;
127             }
128         }
129         // Assertion makes sure the endorsement is removed successfully
130         assert (index >= 0) : "Endorsement not removed successfully";
131 
132         // this removes the endorsement from the endorsement list
133         endorsements.remove(index);
134 
135 
136     }
137     /**
138      * This method removes a comment from the comments list
139      * @param comment the comment to be removed
140      */
141     public void removeComment(EmptyPost comment){
142         // this for loop iterates through each element in the comments list
143         int index = -1;
144         for(int i = 0; i < comments.size(); i++){
145             // this if statement gets the index of the comment by comparing the ith element in the comments
                    list
146             if(comments.get(i) == comment){
147                 index = i;
148             }
149         }
150         // Assertion makes sure that the comment is removed successfully
151         assert (index >= 0) : "Comment not removed successfully";
152 
153         // this removes the comments from the comments list
154         comments.remove(index);
155     }
156 
157 
158     @Override
159     protected void delete(){
160         // A while loop iterates through the endorsements and deletes them
161         while(!endorsements.isEmpty()){
162             Endorsement endorsement = endorsements.get(0);
163             endorsement.delete();
164         }
165         // For each comment in the comment list the commented post is set to an empty post
166         for(Comment comment : comments){
167             comment.setCommentedPost(new EmptyPost());
168         }
169         // The comment list is cleared
170         comments.clear();
171 
172         // Assertion makes user that the deletion was successful
173         assert (endorsements.size() == 0 && comments.size() == 0) : "Post not deleted successfully";
174     }
175 
176     /**
```

```
177      * Get the list of endorsements for the post
178      * @return the list of endorsements
179      */
180     public ArrayList<Endorsement> getEndorsements() {
181         return endorsements;
182     }
183 
184 }
```

# 3 Comment.java

```
1  package socialmedia;
2  
3  
4  /**
5   * This comment class is used to create objects that represents comments on a post
6   *
7   * @author Jonathan Rutland and Daniel Stirling Barros
8   * @version 1.0
9   */
10 public class Comment extends ActionablePost{
11 
12     /**
13      * A public static field {@link Integer} that stores the number of comment posts on the social media
            platform
14      */
15     public static int numberComments = 0;
16 
17     /**
18      * A private field {@link EmptyPost} that stores a post that has been commented on
19      */
20     private EmptyPost commentedPost;
21 
22 
23     /**
24      * A constructor that creates a new comment object
25      * @param poster the account of the commenter
26      * @param commentedPost the post that the commenter is posting the comment in
27      * @param message the message the comment contains
28      * @throws InvalidPostException when trying to create an invalid post
29      * @throws NotActionablePostException when trying to act upon an non-actionable post
30      */
31     public Comment(Account poster, Post commentedPost, String message) throws InvalidPostException,
            NotActionablePostException{
32         // Use the constructor of the super class to set the id of the comment
33         super();
34 
35         // A pre condition checks if the commented post is actionable or if it has a valid message
36         isValidMessage(message);
37         isActionable(commentedPost);
38 
39         // The commented post, poster and message are set
40         this.commentedPost = commentedPost;
41         this.poster = poster;
```

```
42          this.message = message;
43
44          // The number of comments are incremented by 1
45          numberComments++;
46
47          // The comment is added to the original post list of comments and added to the posters list of posts
48          this.poster.addPost(this);
49          ((ActionablePost)commentedPost).addComment(this);
50      }
51
52      /**
53       * This method is used to set the original post
54       * @param commentedPost the post to be set
55       */
56      public void setCommentedPost(EmptyPost commentedPost){
57          this.commentedPost = commentedPost;
58      }
59
60
61      @Override
62      public void delete(){
63          // use the super method to handle the list of comments and endorsements
64          super.delete();
65
66          // A variable is set for the post condition
67          int originalNumberOfComments = numberComments;
68
69          // The comment is removed from the original post list of comments
70          if(commentedPost instanceof ActionablePost){
71              ((ActionablePost)commentedPost).removeComment(this);
72          }
73
74          // The comment is removed from the posters list of comments
75          poster.removePost(this);
76
77          // The number of comments is decremented by 1
78          numberComments--;
79
80          // Assertion checks that the post condition is met and if it is not met then it throws an exception
81          assert (originalNumberOfComments - 1 == numberComments):"Comment not successfully deleted";
82      }
83
84      /**
85       * Resets the counters for comment
86       */
87      static public void resetCounters(){
88          numberComments = 0;
89      }
90  }
```

# 4 EmptyPost.java

```
1   package socialmedia;
2
```

```java
import java.io.Serializable;

/**
 * The empty post class is used to create objects that represent user an empty post with a message
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
public class EmptyPost implements Serializable{

    /**
     * A protected attribute {@link String} that stores the message of the Post being created
     */
    protected String message;

    /**
     * This constructor creates an empty post
     */
    public EmptyPost(){
        this.message = "The original content was removed from the system and is no longer available.";
    }


    /**
     * Gets the message of the post
     * @return the message of the post
     */
    public String getMessage() {
        return message;
    }
}
```

# 5 Endorsement.java

```java
package socialmedia;


/**
 * This endorsement class is used to create endorsement objects that represent endorsements from posts
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
public class Endorsement extends Post{

    /**
     * A private field {@link ActionablePost} that stores the post that is being endorsed
     */
    private ActionablePost endorsedPost;

    /**
     * A public static {@link Integer} that stores the number of endorsements in the system
     */
    public static int numberEndorsements=0;
```

```
21
22
23      /**
24       * A constructor that creates a new endorsement object
25       * @param poster the account that is endorsing the post
26       * @param endorsedPost the post that is being endorsed
27       * @throws NotActionablePostException when trying to act upon an non-actionable post
28       */
29      public Endorsement(Account poster, Post endorsedPost) throws NotActionablePostException{
30          // Use the constructor of the super class sets the id of the post
31          super();
32
33          // A pre condition checks that if the post being endorsed is actionable
34          isActionable(endorsedPost);
35
36          // The post being endorsed is set along with the poster and message
37          this.endorsedPost = (ActionablePost)endorsedPost;
38          this.poster = poster;
39          this.message = endorsedPost.message;
40
41          // The post is added to the posters list of posts
42          this.poster.addPost(this);
43
44          // The endorsement gets added to the endorsed post endorsements list
45          ((ActionablePost)endorsedPost).addEndorsement(this);
46
47          // This gets the account that is being endorsed and increment the number of endorsements by 1
48          Account endorsedAccount = endorsedPost.getPoster();
49          endorsedAccount.incrementEndorsements();
50
51          // The number of endorsements is incremented by 1
52          numberEndorsements++;
53      }
54
55
56      @Override
57      public void delete(){
58          // A new variable is created for asserting the endorsement was deleted
59          int numberOfOriginalEndorsements = numberEndorsements;
60
61          // The account that was endorsed is retrieved and its number of endorsements is decremented by 1
62          Account endorsedAccount = endorsedPost.getPoster();
63          endorsedAccount.decrementEndorsements();
64
65
66          // The endorsement is removed from the posters and the endorsed post list of endorsements
67          poster.removePost(this);
68          endorsedPost.removeEndorsement(this);
69
70          // The number of endorsements is decremented by 1
71          numberEndorsements--;
72
73          // Assertion checks that the post condition is met and if it is not met then it throws an exception
74          assert (numberOfOriginalEndorsements -1 == numberEndorsements):"Endorsement not deleted
                successfully";
```

```
    }

    /**
     * Resets the static counters in Endorsement
     */
    static public void resetCounters() {
        numberEndorsements = 0;
    }

    @Override
    public String toString(){
        // A new StringBuilder is created and then information about the endorsement post is appended to it
        StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append("ID: ").append(ID).append("\n");
      stringBuilder.append("Account: ").append(poster.getHandle()).append("\n");
      stringBuilder.append("No. endorsements: ").append(0).append(" | ");
      stringBuilder.append("No. comments: ").append(0).append("\n");
      stringBuilder.append(message);

        // The information is returned as a string
        return stringBuilder.toString();
    }


}
```

# 6 OriginalPost.java

```
package socialmedia;

/**
 * The original post class is used to create objects that represent users original posts on social media
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
public class OriginalPost extends ActionablePost{

    /**
     * A public static field {@link Integer} that stores the number of original posts on the social media
         platform
     */
    public static int numberOriginalPosts = 0;


    /**
     * A public constructor that creates a new post object
     * @param poster is the account that has posted the post
     * @param message is the message that the account has written in the post
     * @throws InvalidPostException if the message is empty or has more than 100 characters
     */
    public OriginalPost(Account poster, String message) throws InvalidPostException{
        // The super class constructor is used to set id of the post
        super();
```

```
26
27          // A pre condition checks if the message of the post is valid
28          isValidMessage(message);
29
30          // The poster and message of the post are set
31          this.poster = poster;
32          this.message = message;
33
34          // The number of posts is incremented
35          numberOriginalPosts++;
36
37          // Post is gets added to posters list of posts
38          poster.addPost(this);
39      }
40
41      @Override
42      public void delete(){
43          // A new variable is created for asserting the OriginalPost was deleted
44          int numberOfOriginalOriginalPosts = numberOriginalPosts;
45
46          // The super class delete is run ensuring that the comments and endorsements are handled accordingly
47          super.delete();
48
49          // This decreases the numberPosts counter by 1
50          numberOriginalPosts--;
51
52          // The post is removed from the posters list of posts
53          poster.removePost(this);
54
55          // Assertion checks that the post condition is met and if it is not met then it throws an exception
56          assert (numberOfOriginalOriginalPosts - 1 == numberOriginalPosts):"Original post not deleted
                successfully";
57      }
58
59      /**
60       * Resets the static counters for post
61       */
62      static public void resetCounters(){
63          numberOriginalPosts = 0;
64      }
65
66
67  }
```

# 7 Post.java

```
1   package socialmedia;
2
3   /**
4    * The post class is used to create objects that represent users posts on social media
5    *
6    * @author Jonathan Rutland and Daniel Stirling Barros
7    * @version 1.0
8    */
```

```java
public abstract class Post extends EmptyPost {
    /**
     * A protected constant attribute {@link Integer} used to store the id of a post this is unique to the
         post
     */
    protected final int ID;

    /**
     * A protected attribute {@link Account} used to store the account who posted the post
     */
    protected Account poster;

    /**
     * A public attribute {@link Integer} that stores the id of the next post to be created
     */
    public static int currentId=0;

    /**
     * A basic post constructor that sets ID and increments the current ID
     */
    public Post(){
        this.ID = currentId;
        currentId++;
    }

    /**
     * This method gets the id and then returns the id of the post
     * @return the id of the post
     */
    public int getId(){
        return this.ID;
    }

    /**
     * This method returns a string of information about the Individual post
     * @return the string of information about the post
     */
    @Override
    public abstract String toString();

    /**
     * This method gets the account that has posted the post
     * @return the account that has posted the post
     */
    public Account getPoster(){
        return poster;
    }

    /**
     * Deletes post
     */
    protected abstract void delete();

    /**
     * Resets the static counter used for the post
```

```
     */
    static public void resetCounters(){
        currentId=0;
    }

    /**
     * This method checks if the post message is valid
     * @param message the message being checked
     * @return true if it valid
     * @throws InvalidPostException when trying to create an invalid post
     */
    protected boolean isValidMessage(String message) throws InvalidPostException{
        // This if statement checks if the message is empty or contains more than 100 characters, if it does
            then the exception is thrown
      if(message.equals("") || message.length() > 100){
         throw new InvalidPostException("Message must not be empty and be shorter than 100 characters");
      }
        return true;
    }

    /**
     * This method checks if a post is actionable
     * @param post the post being checked
     * @return true if actionable
     * @throws NotActionablePostException when trying to act upon an non-actionable post
     */
    protected boolean isActionable(Post post) throws NotActionablePostException{
        if(!(post instanceof ActionablePost)){
         throw new NotActionablePostException("Post cannot be acted upon");
      }
        return true;
    }

}
```

# 8 SocialMedia.java

```
package socialmedia;

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

/**
 * The social media class provides an interface to create, modify, read and delete posts or accounts
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
```

```java
public class SocialMedia implements SocialMediaPlatform {
   /**
    * A {@link HashMap} is used here to store key value pairs with the keys of type {@link Integer} and values of type {@link Post}
    * This provides a way of getting Posts based of IDs
    */
   private HashMap<Integer, Post> posts = new HashMap<Integer, Post>();

   /**
    * A {@link HashMap} is used here to store key value pairs with the keys of type {@link String} and values of type {@link Account}
    * This provides a way of getting Accounts based of their handles
    */
   private HashMap<String, Account> accountsByHandle = new HashMap<String, Account>();

   /**
    * A {@link HashMap} is used here to store key value pairs with the keys of type {@link Integer} and values of type {@link Account}
    * This provides a way of getting Accounts based of their ids
    */
   private HashMap<Integer, Account> accountsById = new HashMap<Integer, Account>();

   @Override
   public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
      // A new variable is created for asserting the account was created
      int originalNumberOfAccounts = getNumberOfAccounts();

      // A pre-condition is used here to check if the handle is legal
      isLegalHandle(handle);

      // A new account object is created and instantiated
      Account newAccount = new Account(handle);

      // The id is retrieved from the account and used along with the account handle to add the account to the the hashmaps
      int id = newAccount.getId();
      accountsByHandle.put(handle, newAccount);
      accountsById.put(id, newAccount);

      // Assertion checks that the post condition is met and if it is not met then it throws an exception
      assert (originalNumberOfAccounts + 1 == getNumberOfAccounts()) : "Account not created successfully";

      // The id of new account is returned
      return id;
   }

   @Override
   public int createAccount(String handle, String description) throws IllegalHandleException, InvalidHandleException {
      // A new variable is created for asserting the account was created
      int originalNumberOfAccounts = getNumberOfAccounts();

      // A pre-condition is used here to check if the handle is legal
      isLegalHandle(handle);

```

```
68        // A new account is made with the given handle and description
69        Account newAccount = new Account(handle,description);
70
71        // The id is retrieved from the account and used along with the account handle to add the account to
              the the hashmaps
72        int id = newAccount.getId();
73        accountsByHandle.put(handle, newAccount);
74        accountsById.put(id, newAccount);
75
76        // Assertion checks that the post condition is met and if it is not met then it throws an exception
77        assert (originalNumberOfAccounts + 1 == getNumberOfAccounts() && description ==
              newAccount.getDescription()) : "Account not created successfully";
78
79        // The id of this new account is returned
80        return id;
81    }
82
83    @Override
84    public void removeAccount(int id) throws AccountIDNotRecognisedException {
85        // A new variable is created for asserting the account was removed
86        int originalNumberOfAccounts = getNumberOfAccounts();
87
88        // A pre-condition is used here to check if the account id is recognised
89        isRecognisedAccountID(id);
90
91        // The account to be deleted is retrieved and removed from the hashmaps using it's id and handle
92        Account deleteAccount = accountsById.get(id);
93        accountsById.remove(id);
94        String handle = deleteAccount.getHandle();
95        accountsByHandle.remove(handle);
96
97        // The posts that the account has posted are iterated through and removed from the hashmap containing
              posts in the system
98        ArrayList<Post> deletePosts = deleteAccount.getPosts();
99        for(Post post : deletePosts){
100           // This if statement looks for instances of ActionablePost in post if it is Actionable then it's
                  endorsements are also removed from the Post hashmap
101           if(post instanceof ActionablePost){
102               ArrayList<Endorsement> endorsements = ((ActionablePost)post).getEndorsements();
103               for(Endorsement endorsement : endorsements){
104                   posts.remove(endorsement.getId());
105               }
106           }
107           // The posts gets removed from the hashmap
108           posts.remove(post.getId());
109       }
110
111       // Finally the account is deleted
112       deleteAccount.delete();
113
114       // Assertion checks that the post condition is met and if it is not met then it throws an exception
115       assert (originalNumberOfAccounts - 1 == getNumberOfAccounts()) : "Account not deleted successfully";
116   }
117
118   @Override
```

```
119     public void removeAccount(String handle) throws HandleNotRecognisedException {
120        // A new variable is created for asserting the account was removed
121        int originalNumberOfAccounts = getNumberOfAccounts();
122
123        // A pre-condition is used here to check if the account handle is recognised
124        isRecognisedHandle(handle);
125
126        // The account to be deleted is retrieved and removed from the hashmaps using it's id and handle
127        Account deleteAccount = accountsByHandle.get(handle);
128        accountsByHandle.remove(handle);
129        int id = deleteAccount.getId();
130        accountsById.remove(id);
131
132        // The posts that the account has posted are iterated through and removed from the hashmap containing
               posts in the system
133        ArrayList<Post> deletePosts = deleteAccount.getPosts();
134        for(Post post : deletePosts){
135           // This if statement looks for instances of ActionablePost in post if it is Actionable then it's
                  endorsements are also removed from the Post hashmap
136              if(post instanceof ActionablePost){
137                  ArrayList<Endorsement> endorsements = ((ActionablePost)post).getEndorsements();
138                  for(Endorsement endorsement : endorsements){
139                      posts.remove(endorsement.getId());
140                  }
141              }
142           // The posts gets removed from the hashmap
143              posts.remove(post.getId());
144         }
145
146        // Finally the account is deleted
147        deleteAccount.delete();
148
149        // Assertion checks that the post condition is met and if it is not met then it throws an exception
150        assert (originalNumberOfAccounts - 1 == getNumberOfAccounts()) : "Account not deleted successfully";
151     }
152
153     @Override
154     public void changeAccountHandle(String oldHandle, String newHandle) throws HandleNotRecognisedException,
155           IllegalHandleException, InvalidHandleException{
156
157        // Two pre-conditions are used here to check if the old handle is recognised and if the new handle is
               legal
158        isRecognisedHandle(oldHandle);
159        isLegalHandle(newHandle);
160
161        // The account to be updated is retrieved from the hashmap
162        Account account = accountsByHandle.get(oldHandle);
163
164        // The handle of the account is updated
165        account.setHandle(newHandle);
166
167        // The old key value pair in the hashmap of accounts by handle is removed and updated with this new
               handle
168        accountsByHandle.remove(oldHandle);
169        accountsByHandle.put(newHandle, account);
```

```
170
171         // Assertion checks that the post condition is met and if it is not met then it throws an exception
172         assert (oldHandle != account.getHandle()) : "Handle not changed successfully";
173     }
174
175     @Override
176     public void updateAccountDescription(String handle, String description) throws
            HandleNotRecognisedException {
177
178         // The pre-condition checks here if the handle of the account to change is recognised
179         isRecognisedHandle(handle);
180
181         // The account to be updated is retrieved and its new description is set
182         Account account = accountsByHandle.get(handle);
183         account.setDescription(description);
184     }
185
186     @Override
187     public String showAccount(String handle) throws HandleNotRecognisedException {
188
189         // The pre-condition checks here if the handle of the account to be shown is recognised
190         isRecognisedHandle(handle);
191
192         // The account related to the handle is found
193         Account account = accountsByHandle.get(handle);
194
195         // The information about the account is returned
196         return account.toString();
197     }
198
199     @Override
200     public int createPost(String handle, String message) throws HandleNotRecognisedException,
            InvalidPostException {
201
202         // A new variable is created for asserting the post was created
203         int originalNumberOfOriginalPosts = getTotalOriginalPosts();
204
205         // The pre-condition checks if the handle of the account making the post is recognised in the system
206         isRecognisedHandle(handle);
207
208         // The account posting the post is retrieved from the hashmap
209         Account poster = accountsByHandle.get(handle);
210
211         // The new post is created with the poster account and message of the post
212         OriginalPost newPost = new OriginalPost(poster, message);
213
214         // The id of the post is retrieved and the post is added to the hashmaps of posts
215         int id = newPost.getId();
216         posts.put(id, newPost);
217
218         // Assertion checks that the post condition is met and if it is not met then it throws an exception
219         assert (originalNumberOfOriginalPosts + 1 == getTotalOriginalPosts()) : "Original post not created
                successfully";
220
221         // The id of the new post is returned
```

```
222         return id;
223     }
224
225     @Override
226     public int endorsePost(String handle, int id) throws HandleNotRecognisedException,
227           PostIDNotRecognisedException, NotActionablePostException {
228         // A new variable is created for asserting the post was endorsed
229         int originalNumberOfEndorsements = getTotalEndorsmentPosts();
230
231         // The two pre-condition checks if the handle or the post id is recognised in the system
232         isRecognisedHandle(handle);
233         isRecognisedPostID(id);
234
235         // The post getting endorsed is retrieved an then checked to see if it is an endorsement
236         // If it is an endorsement then the exception is thrown
237         Post post = posts.get(id);
238
239         // The account posting the endorsement is retrieved
240         Account account = accountsByHandle.get(handle);
241
242         // The endorsement is created with the account posting it and the post being endorsed
243         Endorsement endorsement = new Endorsement(account, post);
244
245         // The id of this endorsement is retrieved and the endorsement is added to the hashmaps of posts
246         int newId = endorsement.getId();
247         posts.put(newId, endorsement);
248
249         // Assertion checks that the post condition is met and if it is not met then it throws an exception
250         assert (originalNumberOfEndorsements + 1 == getTotalEndorsmentPosts()) : "Endorsement not created
                successfully";
251
252         // The id of the new endorsement is returned
253         return newId;
254     }
255
256
257
258     @Override
259     public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
260           PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
261         // A new variable is created for asserting the comment was created
262         int originalNumberOfComments = getTotalCommentPosts();
263
264         // The two pre-condition checks if the handle or the post id is recognised in the system
265         isRecognisedHandle(handle);
266         isRecognisedPostID(id);
267
268         // The post being commented on and account posting is retrieved
269         Post post = posts.get(id);
270         Account account = accountsByHandle.get(handle);
271
272         // The comment is created with the poster the post being commented and the message of the post
273         Comment comment = new Comment(account, post, message);
274
275         // The post id is retrieved then the post is added to the hashmap of posts
```

```
276         int newId = comment.getId();
277         posts.put(newId, comment);
278 
279         // Assertion checks that the post condition is met and if it is not met then it throws an exception
280         assert (originalNumberOfComments + 1 == getTotalCommentPosts());
281 
282         // The id of the new post is returned
283         return newId;
284     }
285 
286     @Override
287     public void deletePost(int id) throws PostIDNotRecognisedException {
288         // A new variable is created for asserting the post was deleted
289         int originalNumberOfPosts = posts.size();
290 
291         // The pre-condition checks if the id of post being deleted is recognised in the system
292         isRecognisedPostID(id);
293 
294         // The post being deleted is retrieved
295         Post delPost = posts.get(id);
296         // If the post is an actionable post then a for loop runs removing all of its endorsements from the
                system
297         if (delPost instanceof ActionablePost){
298             for(Endorsement endorsement : ((ActionablePost)delPost).getEndorsements()){
299                 int endorseId = endorsement.getId();
300                 posts.remove(endorseId);
301             }
302         }
303 
304         // The post is removed from the system and deleted
305         posts.remove(id);
306         delPost.delete();
307 
308         // Assertion checks that the post condition is met and if it is not met then it throws an exception
309         assert (originalNumberOfPosts - 1 == posts.size()) : "Post not deleted successfully";
310     }
311 
312     @Override
313     public String showIndividualPost(int id) throws PostIDNotRecognisedException {
314         // The pre-condition checks if the id of the post being shown is recognised in the system
315         isRecognisedPostID(id);
316 
317         // The IndividualPost related to the id is found
318         Post post = posts.get(id);
319 
320         // The information about the IndividualPost is returned
321         return post.toString();
322     }
323 
324     @Override
325     public StringBuilder showPostChildrenDetails(int id)
326             throws PostIDNotRecognisedException, NotActionablePostException {
327 
328         // The pre-condition checks of the id of the posts children being shown is recognised in the system
329         isRecognisedPostID(id);
```

```
330
331         // The post is retrieved from the system
332         Post post = posts.get(id);
333
334         // A method from Post should generate a string builder of its details and its children
335         StringBuilder postChildrenDetails = ((ActionablePost)post).showChildren(0);
336
337         // The string builder is returned
338         return postChildrenDetails;
339     }
340
341     @Override
342     public int getNumberOfAccounts() {
343         // The number of accounts is retrieved from the account class then returned
344         int numberOfAccounts = Account.numberAccounts;
345         return numberOfAccounts;
346     }
347
348     @Override
349     public int getTotalOriginalPosts() {
350         // The number of original posts is retrieved from the Post class and then returned
351         int numberOfPosts = OriginalPost.numberOriginalPosts;
352         return numberOfPosts;
353     }
354
355     @Override
356     public int getTotalEndorsmentPosts(){
357         // The number of endorsements is retrieved from the Endorsement class and then returned
358         int numberOfEndorsements = Endorsement.numberEndorsements;
359         return numberOfEndorsements;
360     }
361
362     @Override
363     public int getTotalCommentPosts() {
364         // The number of comments is retrieved from the Comment class and then returned
365         int numberOfComments = Comment.numberComments;
366         return numberOfComments;
367     }
368
369     @Override
370     public int getMostEndorsedPost() {
371         // The set of ids is retrieved from the hashmaps of posts
372         Set<Integer> ids = posts.keySet();
373
374         // These variables keep track of the most endorsed id the max number of endorsements
375         int mostEndorsedId = -1;
376         int maxEndorsements = 0;
377
378         // This for loop iterates through the post ids in the system
379         for(int id: ids){
380             // The ith post is retrieved and then compared to the max number of endorsements
381             if(!(posts.get(id) instanceof ActionablePost)){
382                 continue;
383             }
384             ActionablePost currentPost = (ActionablePost)posts.get(id);
```

```
385             if(currentPost.getNumberOfEndorsements() >= maxEndorsements){
386                 // If a new max number of endorsements is found the number is updated and the most endorsed id
                        is updated
387                 maxEndorsements = currentPost.getNumberOfEndorsements();
388                 mostEndorsedId = id;
389             }
390         }
391 
392         // Assertion checks that the post condition is met and if it is not met then it throws an exception
393         assert (mostEndorsedId > 0) : "Most endorsed post id not found successfully";
394 
395         // Finally the id of the post is the most endorsements is returned
396         return mostEndorsedId;
397     }
398 
399     @Override
400     public int getMostEndorsedAccount() {
401         // The set of account id is retrieved from the accounts by id hashmap
402         Set<Integer> ids = accountsById.keySet();
403 
404         // These variables keep track of the most endorsed id and the max number of endorsements
405         int mostEndorsedId = -1;
406         int maxEndorsements = 0;
407 
408         // This for loop iterates through the account ids in the system
409         for(int id : ids){
410             // The ith account is retrieved and then its endorsements is compared to the max number of
                    endorsements
411             Account currentAccount = accountsById.get(id);
412             if(currentAccount.getEndorsements() >= maxEndorsements){
413                 // If a new max number of endorsements is found the number is updated and the most endorsed id
                        is updated
414                 maxEndorsements = currentAccount.getEndorsements();
415                 mostEndorsedId = id;
416             }
417         }
418 
419         // Assertion checks that the post condition is met and if it is not met then it throws an exception
420         assert (mostEndorsedId > 0) : "Most endorsed account id not found successfully";
421 
422         // Finally the id of the most endorsed account is returned
423         return mostEndorsedId;
424     }
425 
426     @Override
427     public void erasePlatform() {
428         // The HashMaps of post and accounts are reset to empty
429         posts.clear();
430         accountsById.clear();
431         accountsByHandle.clear();
432         // The counters for accounts, posts, original posts, comments and endorsements are all reset to the
                original values
433         Account.resetCounters();
434         Post.resetCounters();
435         OriginalPost.resetCounters();
```

```
436         Comment.resetCounters();
437         Endorsement.resetCounters();
438     }
439 
440     @Override
441     public void savePlatform(String filename) throws IOException {
442         // The current id of accounts being created and the number of accounts is retrieved
443         int accountCurrentID = Account.currentId;
444         int accountNumber = Account.numberAccounts;
445         // The current id of posts being created and the number of original posts, comments and endorsements
                are retrieved
446         int postCurrentID = Post.currentId;
447         int originalPostNumber = OriginalPost.numberOriginalPosts;
448         int commentPostNumber = Comment.numberComments;
449         int endorsePostNumber = Endorsement.numberEndorsements;
450 
451         // An ObjectOutputStream is created for the desired file
452         try(ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))){
453             // First 2 lines contain the counter information for accounts
454             oos.writeObject(accountCurrentID);
455             oos.writeObject(accountNumber);
456 
457             // The next 4 lines contain the counter information for posts
458             oos.writeObject(postCurrentID);
459             oos.writeObject(originalPostNumber);
460             oos.writeObject(commentPostNumber);
461             oos.writeObject(endorsePostNumber);
462 
463             // The final 3 lines contain the 3 HashMaps in the system
464             oos.writeObject(posts);
465             oos.writeObject(accountsById);
466             oos.writeObject(accountsByHandle);
467         }
468     }
469 
470     @SuppressWarnings("unchecked")
471     @Override
472     public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
473 
474         // An ObjectInputStream is created for the desired file
475         try(ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))){
476             // The first 2 lines are read and casted to integers then they are used to set the account counters
477             Account.currentId = (int)in.readObject();
478             Account.numberAccounts = (int)in.readObject();
479 
480             // The next 4 lines are read and casted to integers then they are used to set the post counters
481             Post.currentId = (int)in.readObject();
482             OriginalPost.numberOriginalPosts = (int)in.readObject();
483             Comment.numberComments = (int)in.readObject();
484             Endorsement.numberEndorsements = (int)in.readObject();
485 
486             // The final 3 lines are read and casted to HashMaps and stores in the social media's post and
                    account hashmaps
487             posts = (HashMap<Integer, Post>)in.readObject();
488             accountsById = (HashMap<Integer,Account>)in.readObject();
```

```
489             accountsByHandle = (HashMap<String, Account>)in.readObject();
490         }
491     }
492 
493     /**
494      * This method checks if the handle is legal
495      * @param handle the handle being checked to see if it is legal
496      * @return true if legal
497      * @throws IllegalHandleException when trying to assign a handle that is already being used to an account
498      */
499     private boolean isLegalHandle(String handle) throws IllegalHandleException{
500         // This if statement checks if the handle already exists in social media an throws the exception if it
                does
501         if(accountsByHandle.get(handle) != null){
502             throw new IllegalHandleException("Handle already exists in the system");
503         }
504         return true;
505     }
506     /**
507      * This method checks if the PostID is in the system
508      * @param id the PostID being checked to see if it is in the system
509      * @return true if it is in the system
510      * @throws PostIDNotRecognisedException when trying to use a PostID that is not in the system
511      */
512     private boolean isRecognisedPostID(int id) throws PostIDNotRecognisedException{
513         // This if statement checks if there is a post matching to the id in the system, if not the the
                exception is thrown
514         if(posts.get(id) == null){
515             throw new PostIDNotRecognisedException("No Post in the system with given ID");
516         }
517         return true;
518     }
519     /**
520      * This method checks if the handle is in the system
521      * @param handle the handle being checked to see if it system
522      * @return true if it is in the system
523      * @throws HandleNotRecognisedException when trying to use a handle that is not in the system
524      */
525     private boolean isRecognisedHandle(String handle) throws HandleNotRecognisedException{
526         // This if statement checks if there is an account with the given id in the system if there isn't the
                exception is thrown
527         if(accountsByHandle.get(handle) == null){
528             throw new HandleNotRecognisedException("Account with given handle does not exist in the system");
529         }
530         return true;
531     }
532     /**
533      * This method checks if the AccountID is in the system
534      * @param id the AccountID being checked to see if it is in the system
535      * @return true if it is in the system
536      * @throws AccountIDNotRecognisedException when trying to use an AccountID that is not in the system
537      */
538     private boolean isRecognisedAccountID(int id) throws AccountIDNotRecognisedException{
539         // This if statement checks if there is an account with the given id in the system if there isn't the
                exception is thrown
```

```
540         if(accountsById.get(id) == null){
541            throw new AccountIDNotRecognisedException("Account with given id does not exist in the system");
542         }
543         return true;
544      }
545   
546   }
```

# 9 SocialMediaPlatformTestApp.java

```
1   import java.io.File;
2   import java.io.IOException;
3   
4   import socialmedia.AccountIDNotRecognisedException;
5   import socialmedia.HandleNotRecognisedException;
6   import socialmedia.IllegalHandleException;
7   import socialmedia.InvalidHandleException;
8   import socialmedia.InvalidPostException;
9   import socialmedia.NotActionablePostException;
10  import socialmedia.PostIDNotRecognisedException;
11  import socialmedia.SocialMedia;
12  import socialmedia.SocialMediaPlatform;
13  
14  /*
15   * To run with assertions run in this order
16   * javac -d bin .\src\socialmedia\*.java
17   * javac -cp bin -d bin .\src\SocialMediaPlatformTestApp.java
18   * java -cp bin. -ea SocialMediaPlatformTestApp
19   */
20  
21  /**
22   * A short program to illustrate an app testing some minimal functionality of a
23   * concrete implementation of the SocialMediaPlatform interface -- note you will
24   * want to increase these checks, and run it on your SocialMedia class (not the
25   * BadSocialMedia class).
26   *
27   *
28   * @author Diogo Pacheco
29   * @version 1.0
30   */
31  public class SocialMediaPlatformTestApp {
32  
33     /**
34      * Test method.
35      *
36      * @param args not used
37      */
38     public static void main(String[] args) throws Exception{
39        System.out.println("The system compiled and started the execution...");
40  
41        SocialMediaPlatform platform = new SocialMedia();
42  
43        assert (platform.getNumberOfAccounts() == 0) : "Initial SocialMediaPlatform not empty as required.";
44        assert (platform.getTotalOriginalPosts() == 0) : "Initial SocialMediaPlatform not empty as required.";
```

```
45        assert (platform.getTotalCommentPosts() == 0) : "Initial SocialMediaPlatform not empty as required.";
46        assert (platform.getTotalEndorsmentPosts() == 0) : "Initial SocialMediaPlatform not empty as
              required.";
47
48        Integer id;
49        try {
50           id = platform.createAccount("my_handle");
51           assert (platform.getNumberOfAccounts() == 1) : "number of accounts registered in the system does
                 not match";
52
53           platform.removeAccount(id);
54           assert (platform.getNumberOfAccounts() == 0) : "number of accounts registered in the system does
                 not match";
55
56        } catch (IllegalHandleException e) {
57           assert (false) : "IllegalHandleException thrown incorrectly";
58        } catch (InvalidHandleException e) {
59           assert (false) : "InvalidHandleException thrown incorrectly";
60        } catch (AccountIDNotRecognisedException e) {
61           assert (false) : "AccountIDNotRecognizedException thrown incorrectly";
62        }
63
64
65        // Testing typical case for create account
66        try{
67           platform.createAccount("ham");
68        }
69        catch (IllegalHandleException e){
70           assert (false) : "IllegalHandleException thrown incorrectly";
71        }
72        catch (InvalidHandleException e){
73           assert (false) : "InvalidHandleException thrown incorrectly";
74        }
75
76        // Testing erroneous case for create account for if handle contains whitespace
77        try{
78           platform.createAccount("smoked ham");
79           assert (false) : "Account incorrectly created";
80        }
81        catch (IllegalHandleException e){
82           assert (false) : "IllegalHandleException thrown incorrectly";
83        }
84        catch (InvalidHandleException e){
85           assert (true);
86        }
87
88        // Testing erroneous case for create account for if handle is empty
89        try{
90           platform.createAccount("");
91           assert (false) : "Account incorrectly created";
92        }
93        catch (IllegalHandleException e){
94           assert (false) : "IllegalHandleException thrown incorrectly";
95        }
96        catch (InvalidHandleException e){
```

```
97            assert (true);
98        }
99
100       // Testing erroneous case for create account for if handle is longer than 30 characters
101       try{
102           platform.createAccount("ham".repeat(11));
103           assert (false) : "Account incorrectly created";
104       }
105       catch (IllegalHandleException e){
106           assert (false) : "IllegalHandleException thrown incorrectly";
107       }
108       catch (InvalidHandleException e){
109           assert (true);
110       }
111
112       //Test erroneous case for illegal handle exception
113       try{
114           platform.createAccount("hamham");
115           platform.createAccount("hamham");
116           assert (false) : "Account incorrectly created";
117       }
118       catch (IllegalHandleException e){
119           assert (true);
120       }
121       catch (InvalidHandleException e){
122           assert (false) : "InvalidHandleException thrown incorrectly";
123       }
124
125
126
127
128       // Testing typical case for create account
129       try{
130           platform.createAccount("createham", "tasty");
131       }
132       catch (IllegalHandleException e){
133           assert (false) : "IllegalHandleException thrown incorrectly";
134       }
135       catch (InvalidHandleException e){
136           assert (false) : "InvalidHandleException thrown incorrectly";
137       }
138
139       // Testing erroneous case for create account for if handle contains whitespace
140       try{
141           platform.createAccount("smoked ham", "tasty");
142           assert (false) : "Account incorrectly created";
143       }
144       catch (IllegalHandleException e){
145           assert (false) : "IllegalHandleException thrown incorrectly";
146       }
147       catch (InvalidHandleException e){
148           assert (true);
149       }
150
151       // Testing erroneous case for create account for if handle is empty
```

```
152         try{
153            platform.createAccount("", "tasty");
154            assert (false) : "Account incorrectly created";
155         }
156         catch (IllegalHandleException e){
157            assert (false) : "IllegalHandleException thrown incorrectly";
158         }
159         catch (InvalidHandleException e){
160            assert (true);
161         }
162
163         // Testing erroneous case for create account for if handle is longer than 30 characters
164         try{
165            platform.createAccount("ham".repeat(11), "tasty");
166            assert (false) : "Account incorrectly created";
167         }
168         catch (IllegalHandleException e){
169            assert (false) : "IllegalHandleException thrown incorrectly";
170         }
171         catch (InvalidHandleException e){
172            assert (true);
173         }
174
175         //Test erroneous case for illegal handle exception
176         try{
177            platform.createAccount("illegalham", "tasty");
178            platform.createAccount("illegalham", "tasty");
179            assert (false) : "Account incorrectly created";
180         }
181         catch (IllegalHandleException e){
182            assert (true);
183         }
184         catch (InvalidHandleException e){
185            assert (false) : "InvalidHandleException thrown incorrectly";
186         }
187
188
189
190         // Test typical case for removing account
191         try{
192            platform.createAccount("removeham");
193            platform.removeAccount("removeham");
194         }
195         catch(HandleNotRecognisedException e){
196            assert (false) : "HandleNotRecognisedException thrown incorrectly";
197         }
198         catch(InvalidHandleException e){
199            assert (false) : "InvalidHandleException thrown incorrectly";
200         }
201         catch(IllegalHandleException e){
202            assert (false) : "IllegalHandleException thrown incorrectly";
203         }
204
205         // Testing erroneous case for removing account
206         try{
```

```
207             platform.removeAccount("grilledham");
208             assert (false) : "Account removed incorrectly";
209         }
210         catch(HandleNotRecognisedException e){
211             assert (true);
212         }
213 
214 
215 
216 
217         // Test typical case for removing account
218         try{
219             int newId = platform.createAccount("removeham1");
220             platform.removeAccount(newId);
221         }
222         catch(AccountIDNotRecognisedException e){
223             assert (false) : "AccountIDNotRecognisedException thrown incorrectly";
224         }
225         catch(InvalidHandleException e){
226             assert (false) : "InvalidHandleException thrown incorrectly";
227         }
228         catch(IllegalHandleException e){
229             assert (false) : "IllegalHandleException thrown incorrectly";
230         }
231 
232         // Testing erroneous case for removing account
233         try{
234             platform.removeAccount(-1);
235             assert (false) : "Account removed incorrectly";
236         }
237         catch(AccountIDNotRecognisedException e){
238             assert (true);
239         }
240 
241 
242 
243         // Testing typical case for changing account handle
244         try{
245             platform.createAccount("changeham");
246             platform.changeAccountHandle("changeham", "changedHam");
247         }
248         catch(InvalidHandleException e){
249             assert (false) : "InvalidHandleException thrown incorrectly";
250         }
251         catch(IllegalHandleException e){
252             assert (false) : "IllegalHandleException thrown incorrectly";
253         }
254         catch(HandleNotRecognisedException e){
255             assert (false) : "HandleNotRecognisedException thrown incorrectly";
256         }
257 
258         // Testing erroneous case for changing account handle when handle not recognised
259         try{
260             platform.changeAccountHandle("notrecognisedham", "changedHam");
261             assert (false) : "Account handle changed incorrectly";
```

```
262         }
263         catch(InvalidHandleException e){
264            assert (false) : "InvalidHandleException thrown incorrectly";
265         }
266         catch(IllegalHandleException e){
267            assert (false) : "IllegalHandleException thrown incorrectly";
268         }
269         catch(HandleNotRecognisedException e){
270            assert (true);
271         }
272
273         // Testing erroneous case for changing account handle when new handle contains a whitespace
274         try{
275            platform.createAccount("changeham1");
276            platform.changeAccountHandle("changeham1", "changed Ham");
277            assert (false) : "Account handle changed incorrectly";
278         }
279         catch(InvalidHandleException e){
280            assert (true);
281         }
282         catch(IllegalHandleException e){
283            assert (false) : "IllegalHandleException thrown incorrectly";
284         }
285         catch(HandleNotRecognisedException e){
286            assert (false) : "HandleNotRecognisedException thrown incorrectly";
287         }
288
289
290         // Testing erroneous case for changing account handle when new handle is empty
291         try{
292            platform.createAccount("changeham2");
293            platform.changeAccountHandle("changeham2", "");
294            assert (false) : "Account handle changed incorrectly";
295         }
296         catch(InvalidHandleException e){
297            assert (true);
298         }
299         catch(IllegalHandleException e){
300            assert (false) : "IllegalHandleException thrown incorrectly";
301         }
302         catch(HandleNotRecognisedException e){
303            assert (false) : "HandleNotRecognisedException thrown incorrectly";
304         }
305
306
307         // Testing erroneous case for changing account handle when new handle is longer than 30 characters
308         try{
309            platform.createAccount("changeham3");
310            platform.changeAccountHandle("changeham3", "ham".repeat(11));
311            assert (false) : "Account handle changed incorrectly";
312         }
313         catch(InvalidHandleException e){
314            assert (true);
315         }
316         catch(IllegalHandleException e){
```

```
317             assert (false) : "IllegalHandleException thrown incorrectly";
318         }
319         catch(HandleNotRecognisedException e){
320             assert (false) : "HandleNotRecognisedException thrown incorrectly";
321         }
322 
323         // Testing erroneous case for changing account handle when new handle is invalid
324         try{
325             platform.createAccount("changeham4");
326             platform.createAccount("invalidham");
327             platform.changeAccountHandle("changeham4", "invalidham");
328             assert (false) : "Account handle changed incorrectly";
329         }
330         catch(InvalidHandleException e){
331             assert (false) : "InvalidHandleException thrown incorrectly";
332         }
333         catch(IllegalHandleException e){
334             assert (true);
335         }
336         catch(HandleNotRecognisedException e){
337             assert (false) : "HandleNotRecognisedException thrown incorrectly";
338         }
339 
340 
341 
342         // Testing typical case for updating account description
343         try{
344             platform.createAccount("changedescriptionham", "tasty");
345             platform.updateAccountDescription("changedescriptionham", "meaty");
346         }
347         catch(HandleNotRecognisedException e){
348             assert (false) : "HandleNotRecognisedException thrown incorrectly";
349         }
350         catch(IllegalHandleException e){
351             assert (false) : "IllegalHandleException thrown incorrectly";
352         }
353         catch(InvalidHandleException e){
354             assert (false) : "InvalidHandleException thrown incorrectly";
355         }
356 
357         // Testing erroneous case for updating account description when handle not recognised
358         try{
359             platform.updateAccountDescription("notrecognisedham", "meaty");
360             assert (false) : "Description changed incorrectly";
361         }
362         catch(HandleNotRecognisedException e){
363             assert (true);
364         }
365 
366 
367 
368 
369         // Testing typical case for showing an account
370         try{
371             int newId = platform.createAccount("showaccount");
```

```
372             assert (platform.showAccount("showaccount").equals("ID: " + newId + "\n"+ "Handle:
                    showaccount\nDescription: \nPost count: 0\nEndorse count: 0")) : "Show account string returned
                    incorrectly";
373         }
374         catch(IllegalHandleException e){
375             assert (false) : "IllegalHandleException thrown incorrectly";
376         }
377         catch(InvalidHandleException e){
378             assert (false) : "InvalidHandleException thrown incorrectly";
379         }
380         catch(HandleNotRecognisedException e){
381             assert (false) : "HandleNotRecognisedException thrown incorrectly";
382         }
383 
384         // Testing erroneous case for showing account when handle not recognised
385         try{
386             platform.showAccount("notrecognisedham");
387             assert (false) : "Account shown incorrectly";
388         }
389         catch(HandleNotRecognisedException e){
390             assert (true);
391         }
392 
393 
394         // Testing typical case for creating a post
395         try{
396             platform.createAccount("postcreator");
397             platform.createPost("postcreator", "look at this piece of ham");
398         }
399         catch(IllegalHandleException e){
400             assert (false) : "IllegalHandleException thrown incorrectly";
401         }
402         catch(InvalidHandleException e){
403             assert (false) : "InvalidHandleException thrown incorrectly";
404         }
405         catch(HandleNotRecognisedException e){
406             assert (false) : "HandleNotRecognisedException thrown incorrectly";
407         }
408         catch(InvalidPostException e){
409             assert (false) : "InvalidPostException thrown incorrectly";
410         }
411 
412         // Testing erroneous case for when handle is not recognised
413         try{
414             platform.createPost("notrecognisedhandle", "look at this piece of ham");
415             assert (false) : "Post created incorrectly";
416         }
417         catch(HandleNotRecognisedException e){
418             assert (true);
419         }
420         catch(InvalidPostException e){
421             assert (false) : "InvalidPostException thrown incorrectly";
422         }
423 
424         // Testing erroneous case when post message is empty
```

```
        try{
            platform.createAccount("postcreator1");
            platform.createPost("postcreator1", "");
            assert (false) : "Post created incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (true);
        }

        // Testing erroneous case when post message is longer than 100 characters
        try{
            platform.createAccount("postcreator2");
            platform.createPost("postcreator2", "ham".repeat(35));
            assert (false) : "Post created incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (true);
        }



        // Testing typical case when making an endorsement
        try{
            platform.createAccount("postcreator3");
            platform.createAccount("endorser");
            int postId = platform.createPost("postcreator3", "Endorse this if you agree ham is good");
            platform.endorsePost("endorser", postId);
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
```

```
480         }
481         catch(InvalidPostException e){
482            assert (false) : "InvalidPostException thrown incorrectly";
483         }
484         catch(PostIDNotRecognisedException e){
485            assert (false) : "PostIDNotRecognisedException thrown incorrectly";
486         }
487         catch(NotActionablePostException e){
488            assert (false) : "NotActionablePostException thrown incorrectly";
489         }
490
491         // Testing erroneous case when making an endorsement and post id doesn't exist in the system
492         try{
493            platform.createAccount("endorser1");
494            platform.endorsePost("endorser1", -1);
495            assert (false) : "Endorsement incorrectly created";
496         }
497         catch(IllegalHandleException e){
498            assert (false) : "IllegalHandleException thrown incorrectly";
499         }
500         catch(InvalidHandleException e){
501            assert (false) : "InvalidHandleException thrown incorrectly";
502         }
503         catch(HandleNotRecognisedException e){
504            assert (false) : "HandleNotRecognisedException thrown incorrectly";
505         }
506         catch(PostIDNotRecognisedException e){
507            assert (true);
508         }
509         catch(NotActionablePostException e){
510            assert (false) : "NotActionablePostException thrown incorrectly";
511         }
512
513         // Testing erroneous case when making an endorsement on an endorsement
514         try{
515            platform.createAccount("postcreator4");
516            platform.createAccount("endorser2");
517            platform.createAccount("endorser3");
518            int postId = platform.createPost("postcreator4", "Endorse this if you agree ham is good");
519            int endorseId = platform.endorsePost("endorser3", postId);
520            platform.endorsePost("endorser2", endorseId);
521            assert (false) : "Endorsement incorrectly created";
522         }
523         catch(IllegalHandleException e){
524            assert (false) : "IllegalHandleException thrown incorrectly";
525         }
526         catch(InvalidHandleException e){
527            assert (false) : "InvalidHandleException thrown incorrectly";
528         }
529         catch(HandleNotRecognisedException e){
530            assert (false) : "HandleNotRecognisedException thrown incorrectly";
531         }
532         catch(InvalidPostException e){
533            assert (false) : "InvalidPostException thrown incorrectly";
534         }
```

```
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException thrown incorrectly";
        }
        catch(NotActionablePostException e){
            assert (true);
        }


        // Testing typical case for creating a comment
        try{
            platform.createAccount("postcreator5");
            platform.createAccount("commenter");
            int postId = platform.createPost("postcreator5", "look at this piece of ham");
            platform.commentPost("commenter", postId, "wow that is some ham");
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (false) : "InvalidPostException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException thrown incorrectly";
        }
        catch(NotActionablePostException e){
            assert (false) : "NotActionablePostException thrown incorrectly";
        }

        // Testing erroneous case when making a comment and post id doesn't exist in the system
        try{
            platform.createAccount("commenter1");
            platform.commentPost("commenter", -1, "wow some tasty ham");
            assert (false) : "Comment incorrectly created";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
            assert (true);
        }
        catch(NotActionablePostException e){
            assert (false) : "NotActionablePostException thrown incorrectly";
        }
```

```
590         catch (InvalidPostException e){
591             assert (false) : "InvalidPostException thrown incorrectly";
592         }
593 
594         // Testing erroneous case when making an comment on an endorsement
595         try{
596             platform.createAccount("postcreator6");
597             platform.createAccount("endorser4");
598             platform.createAccount("commenter2");
599             int postId = platform.createPost("postcreator6", "Endorse this if you agree ham is good");
600             int endorseId = platform.endorsePost("endorser4", postId);
601             platform.commentPost("commenter2", endorseId, "This comment is not allowed!");
602             assert (false) : "Comment incorrectly created";
603         }
604         catch(IllegalHandleException e){
605             assert (false) : "IllegalHandleException thrown incorrectly";
606         }
607         catch(InvalidHandleException e){
608             assert (false) : "InvalidHandleException thrown incorrectly";
609         }
610         catch(HandleNotRecognisedException e){
611             assert (false) : "HandleNotRecognisedException thrown incorrectly";
612         }
613         catch(InvalidPostException e){
614             assert (false) : "InvalidPostException thrown incorrectly";
615         }
616         catch(PostIDNotRecognisedException e){
617             assert (false) : "PostIDNotRecognisedException thrown incorrectly";
618         }
619         catch(NotActionablePostException e){
620             assert (true);
621         }
622 
623         // Testing erroneous case when comment message is empty
624         try{
625             platform.createAccount("postcreator7");
626             platform.createAccount("commenter3");
627             int postId = platform.createPost("postcreator7", "Comment if you think ham is good");
628             platform.commentPost("commenter3", postId, "");
629             assert (false) : "Comment created incorrectly";
630         }
631         catch(IllegalHandleException e){
632             assert (false) : "IllegalHandleException thrown incorrectly";
633         }
634         catch(InvalidHandleException e){
635             assert (false) : "InvalidHandleException thrown incorrectly";
636         }
637         catch(HandleNotRecognisedException e){
638             assert (false) : "HandleNotRecognisedException thrown incorrectly";
639         }
640         catch(InvalidPostException e){
641             assert (true);
642         }
643         catch(PostIDNotRecognisedException e){
644             assert (false) : "PostIDNotRecognisedException thrown incorrectly";
```

```
        }
        catch(NotActionablePostException e){
            assert (false) : "NotActionablePostException thrown incorrectly";
        }

        // Testing erroneous case when comment message is longer than 100 characters
        try{
            platform.createAccount("postcreator8");
            platform.createAccount("commenter4");
            int postId = platform.createPost("postcreator8", "Comment if you think ham is good");
            platform.commentPost("commenter4", postId, "ham".repeat(34));
            assert (false) : "Comment created incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (true);
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException thrown incorrectly";
        }
        catch(NotActionablePostException e){
            assert (false) : "NotActionablePostException thrown incorrectly";
        }



        // Typical case for deleting a post
        try{
            platform.createAccount("postcreator9");
            int postId = platform.createPost("postcreator9", "Ham is terrible");
            platform.deletePost(postId);
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (false) : "InvalidPostException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException thrown incorrectly";
        }
```

```
700
701         // Erroneous case for deleting a post when post id doesn't exist in the system
702         try{
703            platform.deletePost(-1);
704            assert (false) : "Post incorrectly deleted";
705         }
706         catch(PostIDNotRecognisedException e){
707            assert (true);
708         }
709
710         // typical test for the getter of numberOfAccounts
711         int numberAccounts = platform.getNumberOfAccounts();
712         int id1 = -1;
713
714         try{
715            id1 = platform.createAccount("numberAccounts");
716         }
717         catch(IllegalHandleException e){
718            assert (false) : "IllegalHandleException thrown incorrectly";
719         }
720         catch(InvalidHandleException e){
721            assert (false) : "InvalidHandleException thrown incorrectly";
722         }
723         assert (numberAccounts + 1 == platform.getNumberOfAccounts()):"Account number is invalid";
724         try{
725            platform.removeAccount(id1);
726         }
727         catch(AccountIDNotRecognisedException e){
728            assert (false) : "AccountIDNotRecognisedException thrown incorrectly";
729         }
730
731         assert (numberAccounts == platform.getNumberOfAccounts()):"Account number is invalid";
732
733         // typical test for the getter of TotalOriginalPosts
734         int numberPosts = platform.getTotalOriginalPosts();
735         int postId1 = -1;
736         try{
737            platform.createAccount("numberPosts");
738            postId1 = platform.createPost("numberPosts", "post");
739         }
740         catch(IllegalHandleException e){
741            assert (false) : "IllegalHandleException thrown incorrectly";
742         }
743         catch(InvalidHandleException e){
744            assert (false) : "InvalidHandleException thrown incorrectly";
745         }
746         catch(HandleNotRecognisedException e){
747            assert (false) : "HandleNotRecognisedException thrown incorrectly";
748         }
749         catch(InvalidPostException e){
750            assert (false) : "InvalidPostException thrown incorrectly";
751         }
752         assert (numberPosts + 1 == platform.getTotalOriginalPosts()):"Post number is invalid";
753
754         try{
```

```
755             platform.deletePost(postId1);
756         }
757         catch(PostIDNotRecognisedException e){
758             assert (false) : "PostIDNotRecognisedException";
759         }
760         assert (numberPosts == platform.getTotalOriginalPosts()):"Post number is invalid";
761 
762         // typical test for the getter of TotalEndorsementPost
763         int numberEndorsements = platform.getTotalEndorsmentPosts();
764         int postId2;
765         int EndorsementId1 = -1;
766         try{
767             platform.createAccount("numberEndorsements");
768             postId2 = platform.createPost("numberEndorsements", "post");
769             EndorsementId1 = platform.endorsePost("numberEndorsements", postId2);
770         }
771         catch(IllegalHandleException e){
772             assert (false) : "IllegalHandleException thrown incorrectly";
773         }
774         catch(InvalidHandleException e){
775             assert (false) : "InvalidHandleException thrown incorrectly";
776         }
777         catch(HandleNotRecognisedException e){
778             assert (false) : "HandleNotRecognisedException thrown incorrectly";
779         }
780         catch(InvalidPostException e){
781             assert (false) : "InvalidPostException thrown incorrectly";
782         }
783         catch(PostIDNotRecognisedException e){
784             assert (false) : "PostIDNotRecognisedException";
785         }
786         catch(NotActionablePostException e){
787             assert (false) : "PostIDNotRecognisedException";
788         }
789         assert (numberEndorsements + 1 == platform.getTotalEndorsmentPosts()):"Endorsement number is invalid";
790 
791         try{
792             platform.deletePost(EndorsementId1);
793         }
794         catch(PostIDNotRecognisedException e){
795             assert (false) : "PostIDNotRecognisedException";
796         }
797         assert (numberEndorsements == platform.getTotalEndorsmentPosts()):"Endorsement number is invalid";
798 
799         // typical test for the getter of TotalCommentPosts
800         int numberComments = platform.getTotalCommentPosts();
801         int postId3;
802         int commentId1 = -1;
803         try{
804             platform.createAccount("numberComments");
805             postId3 = platform.createPost("numberComments", "post");
806             commentId1 = platform.commentPost("numberComments", postId3, "fine");
807         }
808         catch(IllegalHandleException e){
809             assert (false) : "IllegalHandleException thrown incorrectly";
```

```
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (false) : "InvalidPostException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException";
        }
        catch(NotActionablePostException e){
            assert (false) : "NotActionablePostException thrown incorrectly";
        }
        assert (numberComments + 1 == platform.getTotalCommentPosts()):"Endorsement number is invalid";

        try{
            platform.deletePost(commentId1);
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException";
        }
        assert (numberComments == platform.getTotalCommentPosts()):"Endorsement number is invalid";

        // typical test for the getter of MostEndorsedPost
        int postId4 = -1;
        try{
            platform.createAccount("MostEndorsedPost");
            postId4 = platform.createPost("MostEndorsedPost", "so fine");
            platform.createPost("MostEndorsedPost", "damn so fine");
            platform.endorsePost("MostEndorsedPost", postId4);
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
            assert (false) : "InvalidPostException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException";
        }
        catch(NotActionablePostException e){
            assert (false) : "PostIDNotRecognisedException";
        }
        assert (postId4 == platform.getMostEndorsedPost()) : "Incorrectly got mostEndorsedPost";

        // typical test for the getter of MostEndorsedAccount
```

```
        int id3 = -1;
        int postId6 = -1;
        try{
           id3 = platform.createAccount("MostEndorsedAccount1");
           platform.createAccount("MostEndorsedAccount2");
           postId6 = platform.createPost("MostEndorsedAccount1", "damn it's so fine");
           platform.createPost("MostEndorsedAccount2", "damn it is so fine");
           platform.endorsePost("MostEndorsedAccount1", postId6);
        }
        catch(IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
           assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
           assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
           assert (false) : "InvalidPostException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
           assert (false) : "PostIDNotRecognisedException";
        }
        catch(NotActionablePostException e){
           assert (false) : "PostIDNotRecognisedException";
        }
        assert (id3 == platform.getMostEndorsedAccount()) : "Incorrectly got mostEndorsedAccount";

        // typical test for ErasePlatform
        int postId8 = -1;
        try{
           platform.createAccount("erasePlatform");
           postId8 = platform.createPost("erasePlatform", "hmm");
           platform.endorsePost("erasePlatform", postId8);
           platform.commentPost("erasePlatform", postId8, "damn hmm");
        }
        catch(IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
           assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
           assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidPostException e){
           assert (false) : "InvalidPostException thrown incorrectly";
        }
        catch(PostIDNotRecognisedException e){
           assert (false) : "PostIDNotRecognisedException";
        }
        catch(NotActionablePostException e){
           assert (false) : "PostIDNotRecognisedException";
        }
```

```
920         platform.erasePlatform();
921         assert (platform.getNumberOfAccounts() == 0 && platform.getTotalCommentPosts() == 0 &&
                platform.getTotalEndorsmentPosts() == 0 && platform.getTotalOriginalPosts() == 0) : "Incorrectly
                erased platform";
922
923         // typical test for save and load platform
924         int postId9 = -1;
925         try{
926            platform.createAccount("savePlatform");
927            postId9 = platform.createPost("savePlatform", "damn oh so fine");
928            platform.endorsePost("savePlatform", postId9);
929            platform.commentPost("savePlatform", postId9, "oh");
930         }
931         catch(IllegalHandleException e){
932            assert (false) : "IllegalHandleException thrown incorrectly";
933         }
934         catch(InvalidHandleException e){
935            assert (false) : "InvalidHandleException thrown incorrectly";
936         }
937         catch(HandleNotRecognisedException e){
938            assert (false) : "HandleNotRecognisedException thrown incorrectly";
939         }
940         catch(InvalidPostException e){
941            assert (false) : "InvalidPostException thrown incorrectly";
942         }
943         catch(PostIDNotRecognisedException e){
944            assert (false) : "PostIDNotRecognisedException";
945         }
946         catch(NotActionablePostException e){
947            assert (false) : "PostIDNotRecognisedException";
948         }
949         String filename = "test_save_platform.ser";
950         try{
951            platform.savePlatform(filename);
952         }
953         catch(IOException e){
954            assert (false) : "IOException";
955         }
956         SocialMediaPlatform loadedPlatform = new SocialMedia();
957         try{
958            loadedPlatform.loadPlatform(filename);
959         }
960         catch(IOException e){
961            assert (false) : "IOException";
962         }
963         catch(ClassNotFoundException e){
964            assert (false) : "ClassNotFoundException";
965         }
966         assert (platform.getNumberOfAccounts() == loadedPlatform.getNumberOfAccounts() &&
                platform.getTotalOriginalPosts() == loadedPlatform.getTotalOriginalPosts() &&
                platform.getTotalCommentPosts() == loadedPlatform.getTotalCommentPosts() &&
                platform.getTotalEndorsmentPosts() == loadedPlatform.getTotalEndorsmentPosts());
967         File savedFile = new File(filename);
968         savedFile.delete();
969     }
```

```
970 }
```