## 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
     */
    public Account(String handle) throws InvalidHandleException {
        // A pre condition checks if the given handle is valid
        isValidHandle(handle);

        // The accounts handle, id and description are set
        this.handle = handle;
        this.ID = currentId;
        this.description = "";

        // Then the counters are incremented by 1
        currentId++;
        numberAccounts++;
    }

    /**
     * This constructor makes an Account with a given handle and description
     * @param handle The handle which will be given to the account
     * @param description The description which will be given to the account
     * @throws InvalidHandleException when trying to assign an invalid handle
     */
    public Account(String handle, String description) throws InvalidHandleException{
        // A pre-condition checks if the given handle is valid
        isValidHandle(handle);

        // The accounts handle, id and description are set
        this.handle = handle;
        this.ID = currentId;
        this.description = description;

        // Then the counters are incremented by 1
        currentId++;
        numberAccounts++;
    }

    /**
     * Gets the handle of an account
     * @return the handle of an account
     */
    public String getHandle(){
        return this.handle;
    }

    /**
     * Sets the handle of an account
     * @param handle the new handle of an account to be set
     * @throws InvalidHandleException when trying to assign an invalid handle
     */
    public void setHandle(String handle) throws InvalidHandleException {
        // This pre condition checks if the given handle is valid
      isValidHandle(handle);
        this.handle = handle;
    }

    /**
     * Gets the id of an account
     *  @return the id of an account
     */
    public int getId(){
        return this.ID;
    }

    /**
     * Gets the description of an account
     * @return the description of an account
     */
    public String getDescription(){
        return this.description;
    }

    /**
     * Sets the description of an account
     *  @param description the new description to be set
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * Gets the number of endorsements an account has received on posts
     * @return the number of endorsements an account has received on posts
     */
    public int getEndorsements(){
        return this.numberOfEndorsements;
    }

    /**
     * Increments the number of endorsements an account has received on posts
     */
    public void incrementEndorsements(){
        this.numberOfEndorsements++;
    }

    /**
     * Decrements the number of endorsements and account has received on posts
     */
    public void decrementEndorsements(){
        this.numberOfEndorsements--;
    }

    /**
     * Deletes an account
     */
    public void delete() {
        // A new variable is created for asserting the account was deleted
        int originalNumberOfAccounts = numberAccounts;

        // The for loops goes through the accounts posts and calls their delete method
        while(!posts.isEmpty()){
            Post post = posts.get(0);
            post.delete();
```

```java
        }

        // The number of accounts is decremented by 1
        numberAccounts--;

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

    /**
     * Gets the list of posts that an account has posted
     * @return the posts an account has posted
     */
    public ArrayList<Post> getPosts(){
        return this.posts;
    }

    /**
     * Resets the counters for number of account and current id to their initial values
     */
    static public void resetCounters(){
        numberAccounts = 0;
        currentId = 0;
    }

    /**
     * Adds a given post to an accounts list of posts
     * @param post the post to be added to an accounts list of posts
     */
    public void addPost(Post post){
        posts.add(post);
    }

    /**
     * Removes a post from an accounts list of posts
     * @param post the post to be removed from the list of posts
     */
    public void removePost(Post post){
        // The index of the post to be removed is found using a for loop
        int index = -1;
        for(int i = 0; i < posts.size(); i++){
            if(posts.get(i) == post){
                index = i;
            }
        }

        // Assertion checks that the post condition is met and if it is not met then it throws an exception
        assert (index>=0):"Post not removed successfully";

        // The post is removed using this ID
        posts.remove(index);
    }
    /**
     * This method returns a string of information about the account
     * @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

```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
     */
    protected ArrayList<Comment> comments = new ArrayList<Comment>();

    /**
     * This method returns a string of information about the Individual post with a given indent
     * @param indent the number of indents needed
     * @return the string of information about the post
     */
    private String toString(int indent){
        // A pre condition checks if the given indent is valid
        if(indent < 0){
            throw new IllegalArgumentException("Indent cannot be less than 0");
        }

        // A new StringBuilder is created and then information about the Individual post is appended to it
             with an indent
        StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append(" ".repeat((indent - 1) * 4) + "| > ").append("ID: ").append(ID).append("\n");
      stringBuilder.append(" ".repeat(indent * 4)).append("Account:
           ").append(poster.getHandle()).append("\n");
      stringBuilder.append(" ".repeat(indent * 4)).append("No. endorsements:
           ").append(getNumberOfEndorsements()).append(" | ");
      stringBuilder.append("No. comments: ").append(comments.size()).append("\n");
      stringBuilder.append(" ".repeat(indent * 4)).append(message);

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

    /**
     * This method recursively generates a string builder with details of the current posts and its children
         posts
     * @param indent the indent to be applied to the details of the children
     * @return A string builder of the children posts
     */
    public StringBuilder showChildren(int indent){
        // A pre condition checks if the given indent is valid
        if(indent < 0){
            throw new IllegalArgumentException("Indent cannot be less than 0");
        }

        // A new string builder is instantiated
        StringBuilder stringBuilder = new StringBuilder();

        // If this is the current post then no indent is needed
        if(indent == 0){
            stringBuilder.append(toString());
        }
        // Otherwise an indent is applied to the details
        else{
            stringBuilder.append(toString(indent));
        }
        stringBuilder.append("\n");

        // This for loop iterates through the comments of the post
        for(int i = 0; i < comments.size(); i++){
            // An index and and a bar is added then the details about the comments are added to the string
               builder
            stringBuilder.append("  ".repeat(indent)+"|\n");
            stringBuilder.append(comments.get(i).showChildren(indent + 1).toString());
        }
        // Finally the string builder is returned
        return stringBuilder;
    }

    @Override
    public String toString(){
        // A new StringBuilder is created and then information about the Individual 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(getNumberOfEndorsements()).append(" | ");
      stringBuilder.append("No. comments: ").append(comments.size()).append("\n");
      stringBuilder.append(message);

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

    /**
     * This method gets the number of endorsements on the post
     * @return the number of endorsements returned
     */
    public int getNumberOfEndorsements(){
        return endorsements.size();
    }

    /**
     * This method adds an endorsement to the endorsements list
     * @param endorsement the endorsement to be added
     */
    public void addEndorsement(Endorsement endorsement){
        endorsements.add(endorsement);
        assert (endorsements.contains(endorsement)) : "Endorsement not added successfully";
    }
    /**
     * This method adds a comment to the comments list
     * @param comment the comment to be added
     */
    public void addComment(Comment comment){
        comments.add(comment);
        assert (comments.contains(comment)) : "Comment not added successfully";
    }
    /**
     * This method removes an endorsement from the endorsements list
     * @param endorsement the endorsement to be removed
     */
    public void removeEndorsement(Endorsement endorsement){
        // this for loop iterates through each element in the endorsements list
        int index = -1;
        for(int i = 0; i < endorsements.size(); i++){
            // this if statement gets the index of the endorsement by comparing the ith element in the endorsement list
            if(endorsements.get(i) == endorsement){
                index = i;
            }
        }
        // Assertion makes sure the endorsement is removed successfully
        assert (index >= 0) : "Endorsement not removed successfully";

        // this removes the endorsement from the endorsement list
        endorsements.remove(index);


    }
    /**
     * This method removes a comment from the comments list
     * @param comment the comment to be removed
     */
    public void removeComment(EmptyPost comment){
        // this for loop iterates through each element in the comments list
        int index = -1;
        for(int i = 0; i < comments.size(); i++){
            // this if statement gets the index of the comment by comparing the ith element in the comments list
            if(comments.get(i) == comment){
                index = i;
            }
        }
        // Assertion makes sure that the comment is removed successfully
        assert (index >= 0) : "Comment not removed successfully";

        // this removes the comments from the comments list
        comments.remove(index);
    }


    @Override
    protected void delete(){
        // A while loop iterates through the endorsements and deletes them
        while(!endorsements.isEmpty()){
            Endorsement endorsement = endorsements.get(0);
            endorsement.delete();
        }
        // For each comment in the comment list the commented post is set to an empty post
        for(Comment comment : comments){
            comment.setCommentedPost(new EmptyPost());
        }
        // The comment list is cleared
        comments.clear();

        // Assertion makes user that the deletion was successful
        assert (endorsements.size() == 0 && comments.size() == 0) : "Post not deleted successfully";
    }

    /**
```

```java
     * Get the list of endorsements for the post
     * @return the list of endorsements
     */
    public ArrayList<Endorsement> getEndorsements() {
        return endorsements;
    }

}
```

## 3 Comment.java

```java
package socialmedia;


/**
 * This comment class is used to create objects that represents comments on a post
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
public class Comment extends ActionablePost{

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

    /**
     * A private field {@link EmptyPost} that stores a post that has been commented on
     */
    private EmptyPost commentedPost;


    /**
     * A constructor that creates a new comment object
     * @param poster the account of the commenter
     * @param commentedPost the post that the commenter is posting the comment in
     * @param message the message the comment contains
     * @throws InvalidPostException when trying to create an invalid post
     * @throws NotActionablePostException when trying to act upon an non-actionable post
     */
    public Comment(Account poster, Post commentedPost, String message) throws InvalidPostException,
         NotActionablePostException{
        // Use the constructor of the super class to set the id of the comment
        super();

        // A pre condition checks if the commented post is actionable or if it has a valid message
        isValidMessage(message);
        isActionable(commentedPost);

        // The commented post, poster and message are set
        this.commentedPost = commentedPost;
        this.poster = poster;
        this.message = message;

        // The number of comments are incremented by 1
        numberComments++;

        // The comment is added to the original post list of comments and added to the posters list of posts
        this.poster.addPost(this);
        ((ActionablePost)commentedPost).addComment(this);
    }

    /**
     * This method is used to set the original post
     * @param commentedPost the post to be set
     */
    public void setCommentedPost(EmptyPost commentedPost){
        this.commentedPost = commentedPost;
    }


    @Override
    public void delete(){
        // use the super method to handle the list of comments and endorsements
        super.delete();

        // A variable is set for the post condition
        int originalNumberOfComments = numberComments;

        // The comment is removed from the original post list of comments
        if(commentedPost instanceof ActionablePost){
            ((ActionablePost)commentedPost).removeComment(this);
        }

        // The comment is removed from the posters list of comments
        poster.removePost(this);

        // The number of comments is decremented by 1
        numberComments--;

        // Assertion checks that the post condition is met and if it is not met then it throws an exception
        assert (originalNumberOfComments - 1 == numberComments):"Comment not successfully deleted";
    }

    /**
     * Resets the counters for comment
     */
    static public void resetCounters(){
        numberComments = 0;
    }
}
```

## 4 EmptyPost.java

```java
package socialmedia;

```

```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;


    /**
     * A constructor that creates a new endorsement object
     * @param poster the account that is endorsing the post
     * @param endorsedPost the post that is being endorsed
     * @throws NotActionablePostException when trying to act upon an non-actionable post
     */
    public Endorsement(Account poster, Post endorsedPost) throws NotActionablePostException{
        // Use the constructor of the super class sets the id of the post
        super();

        // A pre condition checks that if the post being endorsed is actionable
        isActionable(endorsedPost);

        // The post being endorsed is set along with the poster and message
        this.endorsedPost = (ActionablePost)endorsedPost;
        this.poster = poster;
        this.message = endorsedPost.message;

        // The post is added to the posters list of posts
        this.poster.addPost(this);

        // The endorsement gets added to the endorsed post endorsements list
        ((ActionablePost)endorsedPost).addEndorsement(this);

        // This gets the account that is being endorsed and increment the number of endorsements by 1
        Account endorsedAccount = endorsedPost.getPoster();
        endorsedAccount.incrementEndorsements();

        // The number of endorsements is incremented by 1
        numberEndorsements++;
    }


    @Override
    public void delete(){
        // A new variable is created for asserting the endorsement was deleted
        int numberOfOriginalEndorsements = numberEndorsements;

        // The account that was endorsed is retrieved and its number of endorsements is decremented by 1
        Account endorsedAccount = endorsedPost.getPoster();
        endorsedAccount.decrementEndorsements();


        // The endorsement is removed from the posters and the endorsed post list of endorsements
        poster.removePost(this);
        endorsedPost.removeEndorsement(this);

        // The number of endorsements is decremented by 1
        numberEndorsements--;

        // Assertion checks that the post condition is met and if it is not met then it throws an exception
        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

```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();

        // A pre condition checks if the message of the post is valid
        isValidMessage(message);

        // The poster and message of the post are set
        this.poster = poster;
        this.message = message;

        // The number of posts is incremented
        numberOriginalPosts++;

        // Post is gets added to posters list of posts
        poster.addPost(this);
    }

    @Override
    public void delete(){
        // A new variable is created for asserting the OriginalPost was deleted
        int numberOfOriginalOriginalPosts = numberOriginalPosts;

        // The super class delete is run ensuring that the comments and endorsements are handled accordingly
        super.delete();

        // This decreases the numberPosts counter by 1
        numberOriginalPosts--;

        // The post is removed from the posters list of posts
        poster.removePost(this);

        // Assertion checks that the post condition is met and if it is not met then it throws an exception
        assert (numberOfOriginalOriginalPosts - 1 == numberOriginalPosts):"Original post not deleted
            successfully";
    }

    /**
     * Resets the static counters for post
     */
    static public void resetCounters(){
        numberOriginalPosts = 0;
    }


}
```

## 7 Post.java

```java
package socialmedia;

/**
 * The post class is used to create objects that represent users posts on social media
 *
 * @author Jonathan Rutland and Daniel Stirling Barros
 * @version 1.0
 */
```

```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);

```

```
        // A new account is made with the given handle and description
        Account newAccount = new Account(handle,description);

        // 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() && description ==
            newAccount.getDescription()) : "Account not created successfully";

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

    @Override
    public void removeAccount(int id) throws AccountIDNotRecognisedException {
        // A new variable is created for asserting the account was removed
        int originalNumberOfAccounts = getNumberOfAccounts();

        // A pre-condition is used here to check if the account id is recognised
        isRecognisedAccountID(id);

        // The account to be deleted is retrieved and removed from the hashmaps using it's id and handle
        Account deleteAccount = accountsById.get(id);
        accountsById.remove(id);
        String handle = deleteAccount.getHandle();
        accountsByHandle.remove(handle);

        // The posts that the account has posted are iterated through and removed from the hashmap containing
            posts in the system
        ArrayList<Post> deletePosts = deleteAccount.getPosts();
        for(Post post : deletePosts){
            // 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
            if(post instanceof ActionablePost){
                ArrayList<Endorsement> endorsements = ((ActionablePost)post).getEndorsements();
                for(Endorsement endorsement : endorsements){
                    posts.remove(endorsement.getId());
                }
            }
            // The posts gets removed from the hashmap
            posts.remove(post.getId());
          }

        // Finally the account is deleted
        deleteAccount.delete();

        // 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 deleted successfully";
    }

    @Override
```

```java
public void removeAccount(String handle) throws HandleNotRecognisedException {
  // A new variable is created for asserting the account was removed
  int originalNumberOfAccounts = getNumberOfAccounts();

  // A pre-condition is used here to check if the account handle is recognised
  isRecognisedHandle(handle);

  // The account to be deleted is retrieved and removed from the hashmaps using it's id and handle
  Account deleteAccount = accountsByHandle.get(handle);
  accountsByHandle.remove(handle);
  int id = deleteAccount.getId();
  accountsById.remove(id);

  // The posts that the account has posted are iterated through and removed from the hashmap containing
      posts in the system
  ArrayList<Post> deletePosts = deleteAccount.getPosts();
  for(Post post : deletePosts){
    // 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
      if(post instanceof ActionablePost){
        ArrayList<Endorsement> endorsements = ((ActionablePost)post).getEndorsements();
        for(Endorsement endorsement : endorsements){
          posts.remove(endorsement.getId());
        }
      }
    // The posts gets removed from the hashmap
      posts.remove(post.getId());
   }

  // Finally the account is deleted
  deleteAccount.delete();

  // 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 deleted successfully";
}

@Override
public void changeAccountHandle(String oldHandle, String newHandle) throws HandleNotRecognisedException,
    IllegalHandleException, InvalidHandleException{

  // Two pre-conditions are used here to check if the old handle is recognised and if the new handle is
      legal
  isRecognisedHandle(oldHandle);
  isLegalHandle(newHandle);

  // The account to be updated is retrieved from the hashmap
  Account account = accountsByHandle.get(oldHandle);

  // The handle of the account is updated
  account.setHandle(newHandle);

  // The old key value pair in the hashmap of accounts by handle is removed and updated with this new
      handle
  accountsByHandle.remove(oldHandle);
  accountsByHandle.put(newHandle, account);
```

```java

      // Assertion checks that the post condition is met and if it is not met then it throws an exception
      assert (oldHandle != account.getHandle()) : "Handle not changed successfully";
   }

   @Override
   public void updateAccountDescription(String handle, String description) throws
       HandleNotRecognisedException {

      // The pre-condition checks here if the handle of the account to change is recognised
      isRecognisedHandle(handle);

      // The account to be updated is retrieved and its new description is set
      Account account = accountsByHandle.get(handle);
      account.setDescription(description);
   }

   @Override
   public String showAccount(String handle) throws HandleNotRecognisedException {

      // The pre-condition checks here if the handle of the account to be shown is recognised
      isRecognisedHandle(handle);

      // The account related to the handle is found
      Account account = accountsByHandle.get(handle);

      // The information about the account is returned
      return account.toString();
   }

   @Override
   public int createPost(String handle, String message) throws HandleNotRecognisedException,
       InvalidPostException {

      // A new variable is created for asserting the post was created
      int originalNumberOfOriginalPosts = getTotalOriginalPosts();

      // The pre-condition checks if the handle of the account making the post is recognised in the system
      isRecognisedHandle(handle);

      // The account posting the post is retrieved from the hashmap
      Account poster = accountsByHandle.get(handle);

      // The new post is created with the poster account and message of the post
      OriginalPost newPost = new OriginalPost(poster, message);

      // The id of the post is retrieved and the post is added to the hashmaps of posts
      int id = newPost.getId();
      posts.put(id, newPost);

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

      // The id of the new post is returned
```

```java
        return id;
    }

    @Override
    public int endorsePost(String handle, int id) throws HandleNotRecognisedException,
            PostIDNotRecognisedException, NotActionablePostException {
        // A new variable is created for asserting the post was endorsed
        int originalNumberOfEndorsements = getTotalEndorsmentPosts();

        // The two pre-condition checks if the handle or the post id is recognised in the system
        isRecognisedHandle(handle);
        isRecognisedPostID(id);

        // The post getting endorsed is retrieved an then checked to see if it is an endorsement
        // If it is an endorsement then the exception is thrown
        Post post = posts.get(id);

        // The account posting the endorsement is retrieved
        Account account = accountsByHandle.get(handle);

        // The endorsement is created with the account posting it and the post being endorsed
        Endorsement endorsement = new Endorsement(account, post);

        // The id of this endorsement is retrieved and the endorsement is added to the hashmaps of posts
        int newId = endorsement.getId();
        posts.put(newId, endorsement);

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

        // The id of the new endorsement is returned
        return newId;
    }



    @Override
    public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
            PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
        // A new variable is created for asserting the comment was created
        int originalNumberOfComments = getTotalCommentPosts();

        // The two pre-condition checks if the handle or the post id is recognised in the system
        isRecognisedHandle(handle);
        isRecognisedPostID(id);

        // The post being commented on and account posting is retrieved
        Post post = posts.get(id);
        Account account = accountsByHandle.get(handle);

        // The comment is created with the poster the post being commented and the message of the post
        Comment comment = new Comment(account, post, message);

        // The post id is retrieved then the post is added to the hashmap of posts
        int newId = comment.getId();
        posts.put(newId, comment);

        // Assertion checks that the post condition is met and if it is not met then it throws an exception
        assert (originalNumberOfComments + 1 == getTotalCommentPosts());

        // The id of the new post is returned
        return newId;
    }

    @Override
    public void deletePost(int id) throws PostIDNotRecognisedException {
        // A new variable is created for asserting the post was deleted
        int originalNumberOfPosts = posts.size();

        // The pre-condition checks if the id of post being deleted is recognised in the system
        isRecognisedPostID(id);

        // The post being deleted is retrieved
        Post delPost = posts.get(id);
        // If the post is an actionable post then a for loop runs removing all of its endorsements from the
            system
        if (delPost instanceof ActionablePost){
            for(Endorsement endorsement : ((ActionablePost)delPost).getEndorsements()){
                int endorseId = endorsement.getId();
                posts.remove(endorseId);
            }
        }

        // The post is removed from the system and deleted
        posts.remove(id);
        delPost.delete();

        // Assertion checks that the post condition is met and if it is not met then it throws an exception
        assert (originalNumberOfPosts - 1 == posts.size()) : "Post not deleted successfully";
    }

    @Override
    public String showIndividualPost(int id) throws PostIDNotRecognisedException {
        // The pre-condition checks if the id of the post being shown is recognised in the system
        isRecognisedPostID(id);

        // The IndividualPost related to the id is found
        Post post = posts.get(id);

        // The information about the IndividualPost is returned
        return post.toString();
    }

    @Override
    public StringBuilder showPostChildrenDetails(int id)
            throws PostIDNotRecognisedException, NotActionablePostException {

        // The pre-condition checks of the id of the posts children being shown is recognised in the system
        isRecognisedPostID(id);

        // The post is retrieved from the system
        Post post = posts.get(id);

        // A method from Post should generate a string builder of its details and its children
        StringBuilder postChildrenDetails = ((ActionablePost)post).showChildren(0);

        // The string builder is returned
        return postChildrenDetails;
    }

    @Override
    public int getNumberOfAccounts() {
        // The number of accounts is retrieved from the account class then returned
        int numberOfAccounts = Account.numberAccounts;
        return numberOfAccounts;
    }

    @Override
    public int getTotalOriginalPosts() {
        // The number of original posts is retrieved from the Post class and then returned
        int numberOfPosts = OriginalPost.numberOriginalPosts;
        return numberOfPosts;
    }

    @Override
    public int getTotalEndorsmentPosts(){
        // The number of endorsements is retrieved from the Endorsement class and then returned
        int numberOfEndorsements = Endorsement.numberEndorsements;
        return numberOfEndorsements;
    }

    @Override
    public int getTotalCommentPosts() {
        // The number of comments is retrieved from the Comment class and then returned
        int numberOfComments = Comment.numberComments;
        return numberOfComments;
    }

    @Override
    public int getMostEndorsedPost() {
        // The set of ids is retrieved from the hashmaps of posts
        Set<Integer> ids = posts.keySet();

        // These variables keep track of the most endorsed id the max number of endorsements
        int mostEndorsedId = -1;
        int maxEndorsements = 0;

        // This for loop iterates through the post ids in the system
        for(int id: ids){
            // The ith post is retrieved and then compared to the max number of endorsements
            if(!(posts.get(id) instanceof ActionablePost)){
                continue;
            }
            ActionablePost currentPost = (ActionablePost)posts.get(id);
        if(currentPost.getNumberOfEndorsements() >= maxEndorsements){
          // If a new max number of endorsements is found the number is updated and the most endorsed id
              is updated
          maxEndorsements = currentPost.getNumberOfEndorsements();
          mostEndorsedId = id;
        }
      }

      // Assertion checks that the post condition is met and if it is not met then it throws an exception
      assert (mostEndorsedId > 0) : "Most endorsed post id not found successfully";

      // Finally the id of the post is the most endorsements is returned
      return mostEndorsedId;
    }

    @Override
    public int getMostEndorsedAccount() {
      // The set of account id is retrieved from the accounts by id hashmap
      Set<Integer> ids = accountsById.keySet();

      // These variables keep track of the most endorsed id and the max number of endorsements
      int mostEndorsedId = -1;
      int maxEndorsements = 0;

      // This for loop iterates through the account ids in the system
      for(int id : ids){
        // The ith account is retrieved and then its endorsements is compared to the max number of
            endorsements
        Account currentAccount = accountsById.get(id);
        if(currentAccount.getEndorsements() >= maxEndorsements){
          // If a new max number of endorsements is found the number is updated and the most endorsed id
              is updated
          maxEndorsements = currentAccount.getEndorsements();
          mostEndorsedId = id;
        }
      }

      // Assertion checks that the post condition is met and if it is not met then it throws an exception
      assert (mostEndorsedId > 0) : "Most endorsed account id not found successfully";

      // Finally the id of the most endorsed account is returned
      return mostEndorsedId;
    }

    @Override
    public void erasePlatform() {
      // The HashMaps of post and accounts are reset to empty
      posts.clear();
      accountsById.clear();
      accountsByHandle.clear();
      // The counters for accounts, posts, original posts, comments and endorsements are all reset to the
          original values
      Account.resetCounters();
      Post.resetCounters();
      OriginalPost.resetCounters();
```

```java
        Comment.resetCounters();
        Endorsement.resetCounters();
    }

    @Override
    public void savePlatform(String filename) throws IOException {
        // The current id of accounts being created and the number of accounts is retrieved
        int accountCurrentID = Account.currentId;
        int accountNumber = Account.numberAccounts;
        // The current id of posts being created and the number of original posts, comments and endorsements
            are retrieved
        int postCurrentID = Post.currentId;
        int originalPostNumber = OriginalPost.numberOriginalPosts;
        int commentPostNumber = Comment.numberComments;
        int endorsePostNumber = Endorsement.numberEndorsements;

        // An ObjectOutputStream is created for the desired file
        try(ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))){
            // First 2 lines contain the counter information for accounts
            oos.writeObject(accountCurrentID);
            oos.writeObject(accountNumber);

            // The next 4 lines contain the counter information for posts
            oos.writeObject(postCurrentID);
            oos.writeObject(originalPostNumber);
            oos.writeObject(commentPostNumber);
            oos.writeObject(endorsePostNumber);

            // The final 3 lines contain the 3 HashMaps in the system
            oos.writeObject(posts);
            oos.writeObject(accountsById);
            oos.writeObject(accountsByHandle);
        }
    }

    @SuppressWarnings("unchecked")
    @Override
    public void loadPlatform(String filename) throws IOException, ClassNotFoundException {

        // An ObjectInputStream is created for the desired file
        try(ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))){
            // The first 2 lines are read and casted to integers then they are used to set the account counters
            Account.currentId = (int)in.readObject();
            Account.numberAccounts = (int)in.readObject();

            // The next 4 lines are read and casted to integers then they are used to set the post counters
            Post.currentId = (int)in.readObject();
            OriginalPost.numberOriginalPosts = (int)in.readObject();
            Comment.numberComments = (int)in.readObject();
            Endorsement.numberEndorsements = (int)in.readObject();

            // The final 3 lines are read and casted to HashMaps and stores in the social media's post and
                account hashmaps
            posts = (HashMap<Integer, Post>)in.readObject();
            accountsById = (HashMap<Integer,Account>)in.readObject();
            accountsByHandle = (HashMap<String, Account>)in.readObject();
        }
    }

    /**
     * This method checks if the handle is legal
     * @param handle the handle being checked to see if it is legal
     * @return true if legal
     * @throws IllegalHandleException when trying to assign a handle that is already being used to an account
     */
    private boolean isLegalHandle(String handle) throws IllegalHandleException{
        // This if statement checks if the handle already exists in social media an throws the exception if it
            does
        if(accountsByHandle.get(handle) != null){
            throw new IllegalHandleException("Handle already exists in the system");
        }
        return true;
    }
    /**
     * This method checks if the PostID is in the system
     * @param id the PostID being checked to see if it is in the system
     * @return true if it is in the system
     * @throws PostIDNotRecognisedException when trying to use a PostID that is not in the system
     */
    private boolean isRecognisedPostID(int id) throws PostIDNotRecognisedException{
        // This if statement checks if there is a post matching to the id in the system, if not the the
            exception is thrown
        if(posts.get(id) == null){
            throw new PostIDNotRecognisedException("No Post in the system with given ID");
        }
        return true;
    }
    /**
     * This method checks if the handle is in the system
     * @param handle the handle being checked to see if it system
     * @return true if it is in the system
     * @throws HandleNotRecognisedException when trying to use a handle that is not in the system
     */
    private boolean isRecognisedHandle(String handle) throws HandleNotRecognisedException{
        // This if statement checks if there is an account with the given id in the system if there isn't the
            exception is thrown
        if(accountsByHandle.get(handle) == null){
            throw new HandleNotRecognisedException("Account with given handle does not exist in the system");
        }
        return true;
    }
    /**
     * This method checks if the AccountID is in the system
     * @param id the AccountID being checked to see if it is in the system
     * @return true if it is in the system
     * @throws AccountIDNotRecognisedException when trying to use an AccountID that is not in the system
     */
    private boolean isRecognisedAccountID(int id) throws AccountIDNotRecognisedException{
        // This if statement checks if there is an account with the given id in the system if there isn't the
            exception is thrown
```

```java
      if(accountsById.get(id) == null){
         throw new AccountIDNotRecognisedException("Account with given id does not exist in the system");
      }
      return true;
   }

}
```

## 9 SocialMediaPlatformTestApp.java

```java
import java.io.File;
import java.io.IOException;

import socialmedia.AccountIDNotRecognisedException;
import socialmedia.HandleNotRecognisedException;
import socialmedia.IllegalHandleException;
import socialmedia.InvalidHandleException;
import socialmedia.InvalidPostException;
import socialmedia.NotActionablePostException;
import socialmedia.PostIDNotRecognisedException;
import socialmedia.SocialMedia;
import socialmedia.SocialMediaPlatform;

/*
 * To run with assertions run in this order
 * javac -d bin .\src\socialmedia\*.java
 * javac -cp bin -d bin .\src\SocialMediaPlatformTestApp.java
 * java -cp bin. -ea SocialMediaPlatformTestApp
 */

/**
 * A short program to illustrate an app testing some minimal functionality of a
 * concrete implementation of the SocialMediaPlatform interface -- note you will
 * want to increase these checks, and run it on your SocialMedia class (not the
 * BadSocialMedia class).
 *
 *
 * @author Diogo Pacheco
 * @version 1.0
 */
public class SocialMediaPlatformTestApp {

   /**
    * Test method.
    *
    * @param args not used
    */
   public static void main(String[] args) throws Exception{
      System.out.println("The system compiled and started the execution...");

      SocialMediaPlatform platform = new SocialMedia();

      assert (platform.getNumberOfAccounts() == 0) : "Initial SocialMediaPlatform not empty as required.";
      assert (platform.getTotalOriginalPosts() == 0) : "Initial SocialMediaPlatform not empty as required.";
```

```
        assert (platform.getTotalCommentPosts() == 0) : "Initial SocialMediaPlatform not empty as required.";
        assert (platform.getTotalEndorsmentPosts() == 0) : "Initial SocialMediaPlatform not empty as
            required.";

        Integer id;
        try {
           id = platform.createAccount("my_handle");
           assert (platform.getNumberOfAccounts() == 1) : "number of accounts registered in the system does
               not match";

           platform.removeAccount(id);
           assert (platform.getNumberOfAccounts() == 0) : "number of accounts registered in the system does
               not match";

        } catch (IllegalHandleException e) {
           assert (false) : "IllegalHandleException thrown incorrectly";
        } catch (InvalidHandleException e) {
           assert (false) : "InvalidHandleException thrown incorrectly";
        } catch (AccountIDNotRecognisedException e) {
           assert (false) : "AccountIDNotRecognizedException thrown incorrectly";
        }


        // Testing typical case for create account
        try{
           platform.createAccount("ham");
        }
        catch (IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch (InvalidHandleException e){
           assert (false) : "InvalidHandleException thrown incorrectly";
        }

        // Testing erroneous case for create account for if handle contains whitespace
        try{
           platform.createAccount("smoked ham");
           assert (false) : "Account incorrectly created";
        }
        catch (IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch (InvalidHandleException e){
           assert (true);
        }

        // Testing erroneous case for create account for if handle is empty
        try{
           platform.createAccount("");
           assert (false) : "Account incorrectly created";
        }
        catch (IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch (InvalidHandleException e){
```

```
        assert (true);
    }

    // Testing erroneous case for create account for if handle is longer than 30 characters
    try{
        platform.createAccount("ham".repeat(11));
        assert (false) : "Account incorrectly created";
    }
    catch (IllegalHandleException e){
        assert (false) : "IllegalHandleException thrown incorrectly";
    }
    catch (InvalidHandleException e){
        assert (true);
    }

    //Test erroneous case for illegal handle exception
    try{
        platform.createAccount("hamham");
        platform.createAccount("hamham");
        assert (false) : "Account incorrectly created";
    }
    catch (IllegalHandleException e){
        assert (true);
    }
    catch (InvalidHandleException e){
        assert (false) : "InvalidHandleException thrown incorrectly";
    }




    // Testing typical case for create account
    try{
        platform.createAccount("createham", "tasty");
    }
    catch (IllegalHandleException e){
        assert (false) : "IllegalHandleException thrown incorrectly";
    }
    catch (InvalidHandleException e){
        assert (false) : "InvalidHandleException thrown incorrectly";
    }

    // Testing erroneous case for create account for if handle contains whitespace
    try{
        platform.createAccount("smoked ham", "tasty");
        assert (false) : "Account incorrectly created";
    }
    catch (IllegalHandleException e){
        assert (false) : "IllegalHandleException thrown incorrectly";
    }
    catch (InvalidHandleException e){
        assert (true);
    }

    // Testing erroneous case for create account for if handle is empty
```

```
        try{
           platform.createAccount("", "tasty");
           assert (false) : "Account incorrectly created";
        }
        catch (IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch (InvalidHandleException e){
           assert (true);
        }

        // Testing erroneous case for create account for if handle is longer than 30 characters
        try{
           platform.createAccount("ham".repeat(11), "tasty");
           assert (false) : "Account incorrectly created";
        }
        catch (IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch (InvalidHandleException e){
           assert (true);
        }

        //Test erroneous case for illegal handle exception
        try{
           platform.createAccount("illegalham", "tasty");
           platform.createAccount("illegalham", "tasty");
           assert (false) : "Account incorrectly created";
        }
        catch (IllegalHandleException e){
           assert (true);
        }
        catch (InvalidHandleException e){
           assert (false) : "InvalidHandleException thrown incorrectly";
        }



        // Test typical case for removing account
        try{
           platform.createAccount("removeham");
           platform.removeAccount("removeham");
        }
        catch(HandleNotRecognisedException e){
           assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(InvalidHandleException e){
           assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }

        // Testing erroneous case for removing account
        try{
```

```java
            platform.removeAccount("grilledham");
            assert (false) : "Account removed incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (true);
        }




        // Test typical case for removing account
        try{
            int newId = platform.createAccount("removeham1");
            platform.removeAccount(newId);
        }
        catch(AccountIDNotRecognisedException e){
            assert (false) : "AccountIDNotRecognisedException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }

        // Testing erroneous case for removing account
        try{
            platform.removeAccount(-1);
            assert (false) : "Account removed incorrectly";
        }
        catch(AccountIDNotRecognisedException e){
            assert (true);
        }



        // Testing typical case for changing account handle
        try{
            platform.createAccount("changeham");
            platform.changeAccountHandle("changeham", "changedHam");
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }

        // Testing erroneous case for changing account handle when handle not recognised
        try{
            platform.changeAccountHandle("notrecognisedham", "changedHam");
            assert (false) : "Account handle changed incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (true);
        }

        // Testing erroneous case for changing account handle when new handle contains a whitespace
        try{
            platform.createAccount("changeham1");
            platform.changeAccountHandle("changeham1", "changed Ham");
            assert (false) : "Account handle changed incorrectly";
        }
        catch(InvalidHandleException e){
            assert (true);
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }


        // Testing erroneous case for changing account handle when new handle is empty
        try{
            platform.createAccount("changeham2");
            platform.changeAccountHandle("changeham2", "");
            assert (false) : "Account handle changed incorrectly";
        }
        catch(InvalidHandleException e){
            assert (true);
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }


        // Testing erroneous case for changing account handle when new handle is longer than 30 characters
        try{
            platform.createAccount("changeham3");
            platform.changeAccountHandle("changeham3", "ham".repeat(11));
            assert (false) : "Account handle changed incorrectly";
        }
        catch(InvalidHandleException e){
            assert (true);
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }

        // Testing erroneous case for changing account handle when new handle is invalid
        try{
            platform.createAccount("changeham4");
            platform.createAccount("invalidham");
            platform.changeAccountHandle("changeham4", "invalidham");
            assert (false) : "Account handle changed incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }
        catch(IllegalHandleException e){
            assert (true);
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }



        // Testing typical case for updating account description
        try{
            platform.createAccount("changedescriptionham", "tasty");
            platform.updateAccountDescription("changedescriptionham", "meaty");
        }
        catch(HandleNotRecognisedException e){
            assert (false) : "HandleNotRecognisedException thrown incorrectly";
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
            assert (false) : "InvalidHandleException thrown incorrectly";
        }

        // Testing erroneous case for updating account description when handle not recognised
        try{
            platform.updateAccountDescription("notrecognisedham", "meaty");
            assert (false) : "Description changed incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (true);
        }




        // Testing typical case for showing an account
        try{
            int newId = platform.createAccount("showaccount");
            assert (platform.showAccount("showaccount").equals("ID: " + newId + "\n"+ "Handle:
                showaccount\nDescription: \nPost count: 0\nEndorse count: 0")) : "Show account string returned
                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";
        }

        // Testing erroneous case for showing account when handle not recognised
        try{
            platform.showAccount("notrecognisedham");
            assert (false) : "Account shown incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (true);
        }


        // Testing typical case for creating a post
        try{
            platform.createAccount("postcreator");
            platform.createPost("postcreator", "look at this piece of 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";
        }

        // Testing erroneous case for when handle is not recognised
        try{
            platform.createPost("notrecognisedhandle", "look at this piece of ham");
            assert (false) : "Post created incorrectly";
        }
        catch(HandleNotRecognisedException e){
            assert (true);
        }
        catch(InvalidPostException e){
            assert (false) : "InvalidPostException thrown incorrectly";
        }

        // 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";
```

```java
        }
        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 an endorsement and post id doesn't exist in the system
        try{
            platform.createAccount("endorser1");
            platform.endorsePost("endorser1", -1);
            assert (false) : "Endorsement 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";
        }

        // Testing erroneous case when making an endorsement on an endorsement
        try{
            platform.createAccount("postcreator4");
            platform.createAccount("endorser2");
            platform.createAccount("endorser3");
            int postId = platform.createPost("postcreator4", "Endorse this if you agree ham is good");
            int endorseId = platform.endorsePost("endorser3", postId);
            platform.endorsePost("endorser2", endorseId);
            assert (false) : "Endorsement 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(InvalidPostException e){
            assert (false) : "InvalidPostException thrown incorrectly";
        }
        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";
        }
```

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

      // Testing erroneous case when making an comment on an endorsement
      try{
         platform.createAccount("postcreator6");
         platform.createAccount("endorser4");
         platform.createAccount("commenter2");
         int postId = platform.createPost("postcreator6", "Endorse this if you agree ham is good");
         int endorseId = platform.endorsePost("endorser4", postId);
         platform.commentPost("commenter2", endorseId, "This comment is not allowed!");
         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(InvalidPostException e){
         assert (false) : "InvalidPostException thrown incorrectly";
      }
      catch(PostIDNotRecognisedException e){
         assert (false) : "PostIDNotRecognisedException thrown incorrectly";
      }
      catch(NotActionablePostException e){
         assert (true);
      }

      // Testing erroneous case when comment message is empty
      try{
         platform.createAccount("postcreator7");
         platform.createAccount("commenter3");
         int postId = platform.createPost("postcreator7", "Comment if you think ham is good");
         platform.commentPost("commenter3", postId, "");
         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";
        }

        // 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";
        }

        // Erroneous case for deleting a post when post id doesn't exist in the system
        try{
           platform.deletePost(-1);
           assert (false) : "Post incorrectly deleted";
        }
        catch(PostIDNotRecognisedException e){
           assert (true);
        }

        // typical test for the getter of numberOfAccounts
        int numberAccounts = platform.getNumberOfAccounts();
        int id1 = -1;

        try{
           id1 = platform.createAccount("numberAccounts");
        }
        catch(IllegalHandleException e){
           assert (false) : "IllegalHandleException thrown incorrectly";
        }
        catch(InvalidHandleException e){
           assert (false) : "InvalidHandleException thrown incorrectly";
        }
        assert (numberAccounts + 1 == platform.getNumberOfAccounts()):"Account number is invalid";
        try{
           platform.removeAccount(id1);
        }
        catch(AccountIDNotRecognisedException e){
           assert (false) : "AccountIDNotRecognisedException thrown incorrectly";
        }

        assert (numberAccounts == platform.getNumberOfAccounts()):"Account number is invalid";

        // typical test for the getter of TotalOriginalPosts
        int numberPosts = platform.getTotalOriginalPosts();
        int postId1 = -1;
        try{
           platform.createAccount("numberPosts");
           postId1 = platform.createPost("numberPosts", "post");
        }
        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";
        }
        assert (numberPosts + 1 == platform.getTotalOriginalPosts()):"Post number is invalid";

        try{
            platform.deletePost(postId1);
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException";
        }
        assert (numberPosts == platform.getTotalOriginalPosts()):"Post number is invalid";

        // typical test for the getter of TotalEndorsementPost
        int numberEndorsements = platform.getTotalEndorsmentPosts();
        int postId2;
        int EndorsementId1 = -1;
        try{
            platform.createAccount("numberEndorsements");
            postId2 = platform.createPost("numberEndorsements", "post");
            EndorsementId1 = platform.endorsePost("numberEndorsements", postId2);
        }
        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 (numberEndorsements + 1 == platform.getTotalEndorsmentPosts()):"Endorsement number is invalid";

        try{
            platform.deletePost(EndorsementId1);
        }
        catch(PostIDNotRecognisedException e){
            assert (false) : "PostIDNotRecognisedException";
        }
        assert (numberEndorsements == platform.getTotalEndorsmentPosts()):"Endorsement number is invalid";

        // typical test for the getter of TotalCommentPosts
        int numberComments = platform.getTotalCommentPosts();
        int postId3;
        int commentId1 = -1;
        try{
            platform.createAccount("numberComments");
            postId3 = platform.createPost("numberComments", "post");
            commentId1 = platform.commentPost("numberComments", postId3, "fine");
        }
        catch(IllegalHandleException e){
            assert (false) : "IllegalHandleException thrown incorrectly";
```

```java
        }
        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";
        }
```

```java
      platform.erasePlatform();
      assert (platform.getNumberOfAccounts() == 0 && platform.getTotalCommentPosts() == 0 &&
          platform.getTotalEndorsmentPosts() == 0 && platform.getTotalOriginalPosts() == 0) : "Incorrectly
          erased platform";

      // typical test for save and load platform
      int postId9 = -1;
      try{
         platform.createAccount("savePlatform");
         postId9 = platform.createPost("savePlatform", "damn oh so fine");
         platform.endorsePost("savePlatform", postId9);
         platform.commentPost("savePlatform", postId9, "oh");
      }
      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";
      }
      String filename = "test_save_platform.ser";
      try{
         platform.savePlatform(filename);
      }
      catch(IOException e){
         assert (false) : "IOException";
      }
      SocialMediaPlatform loadedPlatform = new SocialMedia();
      try{
         loadedPlatform.loadPlatform(filename);
      }
      catch(IOException e){
         assert (false) : "IOException";
      }
      catch(ClassNotFoundException e){
         assert (false) : "ClassNotFoundException";
      }
      assert (platform.getNumberOfAccounts() == loadedPlatform.getNumberOfAccounts() &&
          platform.getTotalOriginalPosts() == loadedPlatform.getTotalOriginalPosts() &&
          platform.getTotalCommentPosts() == loadedPlatform.getTotalCommentPosts() &&
          platform.getTotalEndorsmentPosts() == loadedPlatform.getTotalEndorsmentPosts());
      File savedFile = new File(filename);
      savedFile.delete();
   }
```

```
}
```