File - Post.java

```java
package socialmedia;

import java.io.Serializable;

/**
 * Parent class of all posts.
 *
 * @author 700037512, 700074221
 */
public class Post implements Serializable {

    private int postId;
    private static int newPostId = 0;
    private Account author;
    private String contents;
    private int endorsementCount;
    private PostType postType;

    /**
     * Constructor for the SocialMediaPlatform posts.
     * @param author Account that created the post.
     * @param contents Content of the post.
     *
     * @throws HandleNotRecognisedException if the handle does not match to any
     *                                      account in the system.
     */
    public Post(Account author, String contents) throws HandleNotRecognisedException {
        postId = ++Post.newPostId;
        isAuthorValid(author);
        this.author = author;
        this.contents = contents;
        this.endorsementCount = 0;
        postType = null;
    }

    /**
     * Post constructor that generates a dummy post, used so that it can be overridden in child objects.
     * @param id the id of the post.
     * @param contents the contents of the post.
     */
    public Post(int id, String contents) {
        postId = id;
        this.contents = contents;
        author = null;
    }

    /**
     * Overridden in Comment.java to remove a comment from a comment's ArrayList
     * @param comment the comment object to be removed
     */
    public void removeComment(Comment comment) {
    }

    /**
     * Constructor for the SocialMediaPlatform dummy post for deleted posts.
     * @param id ID assigned to a deleted post.
     */
    public Post(int id) {
        this.postId = id;
        this.author = null;
        this.contents = "The original content was removed from the system and is no longer available.";
        setPostType(PostType.DELETED);
    }

    /**
     * Method to add Comments to the parent post arraylist so they can be orphaned when the post is deleted.
     * @param comment The comment to be added to the arraylist.
     * @throws NotActionablePostException if the handle does not match to any
     *                                    account in the system.
     */
    public void addComment(Comment comment) throws NotActionablePostException {
        if (this.postType.equals(PostType.COMMENT) || this.postType.equals((PostType.ORIGINAL))){
            ((Comment) this).addComment(comment);
        } else {
```

File - Post.java

```
75              throw new NotActionablePostException("This post cannot be commented on.");
76          }
77      }
78
79      /**
80       * Setter method for the contents of the post.
81       * @param contents String contents of a post.
82       */
83      public void setContents(String contents) {
84          this.contents = contents;
85      }
86
87      /**
88       *Validation method checks if the author of the message is not null.
89       * @param author The account the comment has come from.
90       * @throws HandleNotRecognisedException if the handle does not match to any
91       *                                      account in the system.
92       */
93      public void isAuthorValid(Account author) throws HandleNotRecognisedException {
94          if (author == null) {
95              throw new HandleNotRecognisedException("Account is null.");
96          }
97      }
98
99      /**
100      * Adds an endorsement to the counter for a post.
101      * @throws NotActionablePostException if the ID refers to a endorsement post.
102      *                                      Endorsement posts are not endorsable.
103      *                                      Endorsements are not transitive. For
104      *                                      instance, if post A is endorsed by post
105      *                                      B, and an account wants to endorse B, in
106      *                                      fact, the endorsement must refers to A.
107      */
108     public void addEndorsement() throws NotActionablePostException{
109        isPostEndorsable(postType);
110        this.endorsementCount++;
111     }
112
113     /**
114      * Adds an endorsement to the current post's comment ArrayList.
115      * @param endorsement the endorsement object to be added to the comment.
116      * @param post the post the endorsement needs to be added too.
117      */
118     public void addEndorsementList(Endorsement endorsement, Post post) {
119         if (postType == postType.ORIGINAL) {
120             ((Original) post).addEndorsementArrayList(endorsement);
121         }else if (postType == postType.COMMENT){
122             ((Comment) post).addEndorsementArrayList(endorsement);
123         }
124     }
125
126     /**
127      * Method to remove an endorsement from the endorsement count when the endorsement is deleted.
128      */
129     public void removeEndorsement() {
130         this.endorsementCount--;
131     }
132
133     /**
134      * Setter method for NewPostId.
135      * @param newPostId Internal counter to set postIds in chronological order.
136      */
137     public static void setNewPostId(int newPostId) {
138         Post.newPostId = newPostId;
139     }
140
141     /**
142      * Returns the type of post.
143      * @return The type of post.
144      */
145     public PostType getPostType(){
146         return postType;
147     }
148
149     /**
```

File - Post.java

```java
     *Validation method to check if the post is endorsable.
     * @param postType The type of post.
     * @throws NotActionablePostException if the ID refers to a endorsement post.
     *                                    Endorsement posts are not endorsable.
     *                                    Endorsements are not transitive. For
     *                                    instance, if post A is endorsed by post
     *                                    B, and an account wants to endorse B, in
     *                                    fact, the endorsement must refers to A.
     */
    public static void isPostEndorsable(PostType postType) throws NotActionablePostException {
        if (postType.equals(PostType.ORIGINAL)){
            return;
        } else if  (postType.equals(PostType.COMMENT)){
            return;
        }else {
            throw new NotActionablePostException("Attempted to act upon an not-actionable post.");
        }
    }

    /**
     * Getter method for endorsementCount
     * @return Endorsement count
     */
    public int getEndorsementCount() {
        return endorsementCount;
    }

    /**
     * ToString method to show the details of the post, overridden in child objects.
     * @return Blank string
     */
    @Override
    public String toString() {
        return "";
    }

    /**
     * ToString method to show the details of the post, overridden in child objects.
     * @param indentationLevel the indentation required for the showPostChildrenDetails method.
     * @return Blank string
     */
    public String toString(int indentationLevel) {
        return "";
    }

    /**
     * Returns the post's author.
     * @return The Account object corresponding to the post's author
     */
    public Account getAuthor() {
        return author;
    }

    /**
     * Getter method for post id.
     * @return Id of the post.
     */
    public int getId() {
        return this.postId;
    }

    /**
     * Getter method for the Contents of the post.
     * @return Contents of the message.
     */
    public String getContents() {
        return contents;
    }

    /**
     * Set the post type for a post.
     * @param postType the postType to be set.
     */
    public void setPostType(PostType postType){
        this.postType = postType;
```

File - Post.java

```
225     }
226
227     /**
228      * Overridden in Original.java, adds comments to an arraylist of deleted posts.
229      * @param comment the comment to be deleted.
230      */
231     public void addDeletedComment(Comment comment){
232     }
233 }
```

File - Account.java

```java
package socialmedia;

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

/**
 * Class for account objects.
 *
 * @author 700037512, 700074221
 */
public class Account implements Serializable {

    private String description;
    private static int newId = 0;
    private int id;
    private String handle;
    private int endorsementCount;
    private ArrayList<Post> userPosts;

    /**
     * Constructor for SocialMediaPlatform accounts.
     * @param handle the handle provided for the new account object.
     * @param description the description provided for the new account object.
     * @throws InvalidHandleException thrown if the new handle is empty,
     *                                has more than 30 characters,
     *                                or has white spaces.
     * @throws IllegalHandleException if the handle already exists in the platform.
     */
    public Account(String handle, String description) throws InvalidHandleException,
IllegalHandleException {
        isHandleValid(handle);
        this.handle = handle;
        this.description = description;
        id = ++Account.newId;
        endorsementCount = 0;
        this.userPosts = new ArrayList<>();
    }

    /**
     * Constructor for miniSocialMediaPlatform accounts.
     * @param handle the handle provided for the new account object.
     * @throws InvalidHandleException thrown if the new handle is empty,
     *                                has more than 30 characters,
     *                                or has white spaces.
     * @throws IllegalHandleException if the handle already exists in the platform.
     */
    public Account(String handle) throws InvalidHandleException, IllegalHandleException {
        isHandleValid(handle);
        this.handle = handle;
        description = "";
        id = ++Account.newId;
        userPosts = new ArrayList<>();
    }

    /**
     * Returns an account's ID
     * @return an account object's id.
     */
    public int getId() {
        return id;
    }

    /**
     * Returns an account's handle
     * @return an account object's handle.
     */
    public String getHandle() {
        return handle;
    }

    /**
     * Sets new handle for an account object.
     * @param handle the new handle.
     * @throws InvalidHandleException thrown if the new handle is empty,
     *                                has more than 30 characters,
```

File - Account.java

```
75       *                                or has white spaces.
76       * @throws IllegalHandleException if the handle already exists in the platform.
77       */
78      public void setHandle(String handle) throws InvalidHandleException, IllegalHandleException{
79          isHandleValid(handle);
80          this.handle = handle;
81      }
82
83      /**
84       * Validation method to check if provided handle can be used (not empty, no spaces, less than 30
    chars).
85       * @param handle the unique non-empty name of the account holder.
86       * @throws InvalidHandleException thrown if the new handle is empty,
87       *                                has more than 30 characters,
88       *                                or has white spaces.
89       */
90      public static void isHandleValid(String handle) throws InvalidHandleException{
91          if ("".equals(handle)) {
92              throw new InvalidHandleException("Empty handle.");
93          }
94          if (handle.contains(" ")){
95              throw new InvalidHandleException("Handle contains whitespace.");
96          }
97          if (handle.length()> 30){
98              throw new InvalidHandleException("Handle is too long.");
99          }
100     }
101
102     /**
103      * Sets a new description for an account.
104      * @param description the new description.
105      */
106     public void setDescription(String description) {
107         this.description = description;
108     }
109
110     /**
111      * Returns the description for a specified account.
112      * @return the description for the account.
113      */
114     public String getDescription(){
115         return description;
116     }
117
118     /**
119      * Increments endorsementCount, called when an account's post is endorsed.
120      */
121     public void addEndorsement(){
122         this.endorsementCount++;
123     }
124
125     /**
126      * Returns an account's endorsementCount
127      * @return an account object's endorsementCount.
128      */
129     public int getEndorsementCount(){
130         return endorsementCount;
131     }
132
133     /**
134      * Sets attributes for removed account.
135      */
136     public void clearAccount(){
137         id = -1;
138         handle = null;
139         description = null;
140         userPosts = new ArrayList<>();
141     }
142
143     /**
144      * Adds a post object to the user's post ArrayList.
145      * @param post The new post object to be added
146      */
147     public void addPost(Post post) {
148         userPosts.add(post);
```

File - Account.java

```
149     }
150
151     /**
152      * Sets new ID for an account object.
153      * @param newId the new ID.
154      */
155     public static void setNewId(int newId) {
156         Account.newId = newId;
157     }
158
159     /**
160      * Returns an account overview
161      * @return a string version of an account's overview
162      */
163     @Override
164     public String toString() {
165        return "\nID: " + id + "\nHandle: " + handle + "\nDescription: " + description +
166                "\nPost count: " + userPosts.size() + "\nEndorse count: " + endorsementCount + "\n";
167     }
168 }
```

File - Comment.java

```java
package socialmedia;

import java.util.ArrayList;

/**
 * Class for comment objects.
 *
 * @author 700037512, 700074221
 */
public class Comment extends Post{
    private boolean isOrphan;
    private Post parent;
    private ArrayList<Endorsement> endorsements;
    private ArrayList<Comment> comments;
    private Post originalParent;

    /**
     *
     * @param author The account that created the post.
     * @param contents The contents of the comment message.
     * @param parent The post that the comment refers too.
     * @param originalParent The post's original parent (eg. if a comment, not the comment parent but the original post
     *                        being commented on)
     * @throws InvalidPostException if the message is empty or has more than
     *                              100 characters.
     * @throws HandleNotRecognisedException if the handle does not match to any
     *                                      account in the system.
     * @throws NotActionablePostException if the ID refers to a endorsement post.
     *                                    Endorsement posts are not endorsable.
     *                                    Endorsements are not transitive. For
     *                                    instance, if post A is endorsed by post
     *                                    B, and an account wants to endorse B, in
     *                                    fact, the endorsement must refers to A.
     */

    public Comment(Account author, String contents, Post parent, Post originalParent)
            throws InvalidPostException, HandleNotRecognisedException, NotActionablePostException {
        super(author, contents);
        isContentsValid(contents);
        isOrphan = false;
        setPostType(PostType.COMMENT);
        this.parent = parent;
        endorsements = new ArrayList<Endorsement>();
        comments = new ArrayList<Comment>();
        this.originalParent = originalParent;
        parent.addComment(this);
    }

    /**
     * Constructor for a deleted comment taking the ID as a parameter.
     * @param id id of the post.
     */
    public Comment(int id) {
        super(id, "The original content was removed from the system and is no longer available.");
        comments = new ArrayList<>();
        setPostType(PostType.DELETED);
    }

    /**
     * Adds a comment to the current post's comment ArrayList.
     * @param comment the contents of the comment.
     */
    public void addComment(Comment comment){
        this.comments.add(comment);
    }

    /**
     *Validation method checks if the contents of the message is valid.
     * @param contents What the author has said.
     * @throws InvalidPostException  if the message is empty or has more than
     *                               100 characters.
     */
    public static void isContentsValid(String contents) throws InvalidPostException{
        if (contents.equals("")) {
```

File - Comment.java

```
75              throw new InvalidPostException("Contents is empty.");
76          }
77          if (contents.length() > 100) {
78              throw new InvalidPostException("Contents is over 100 characters. It is too long.");
79          }
80      }
81
82      /**
83       * Getter method for the Parent of the Parent of the post after the immediate post has been deleted
    .
84       * @return the Original Grand-Parent of the post.
85       */
86      public Post getOriginalParent() {
87          return originalParent;
88      }
89
90      /**
91       * Makes a comment an orphan when the post it refers to is deleted.
92       */
93      public void makeOrphan(){
94          setPostType(PostType.DELETED);
95          this.isOrphan = true;
96          originalParent.addDeletedComment(this);
97      }
98
99      /**
100      * Adds an endorsement to the current post's comment ArrayList.
101      * @param endorsement the endorsement object to be added to the comment.
102      */
103     public void addEndorsementArrayList(Endorsement endorsement){
104         endorsements.add(endorsement);
105     }
106
107     /**
108      * Removes comment from an arraylist when a comment is deleted.
109      * @param comment Comment to be removed.
110      */
111     @Override
112     public void removeComment(Comment comment) {
113         this.comments.remove(comment);
114     }
115
116     /**
117      * Returns the ArrayList of Endorsement objects corresponding to the post.
118      * @return all endorsements for the current post
119      */
120     public ArrayList<Endorsement> getEndorsements() {
121         return endorsements;
122     }
123
124
125     /**
126      * Getter method for Comments.
127      * @return an array list of comments.
128      */
129     public ArrayList<Comment> getComments() {
130         return comments;
131     }
132
133     /**
134      * Getter method for the parent of the post.
135      * @return The parent of the comment.
136      */
137     public Post getParent() {
138         return parent;
139     }
140
141     /**
142      * Setter method for Parent post so it can be changed if the parent post is deleted.
143      * @param parent Post the comment refers to.
144      */
145     public void setParent(Post parent) {
146         this.parent = parent;
147     }
148
```

File - Comment.java

```java
    /**
     * The method generates a formatted string containing the details of a single
     * post. The format is as follows:
     *  ID: [post ID]
     *  Account: [account handle]
     *  No. endorsements: [number of endorsements received by the post] | No. comments: [number of comments received by the post]
     *  [post message]
     */
    @Override
    public String toString(){
        return "\nId: " + this.getId() + "\nAccount: " + this.getAuthor().getHandle() + "\nNo. endorsements: " +
                getEndorsementCount() + " | No. comments: " + comments.size() + "\n" + this.getContents() + "\n";
    }

    /**
     * toString which takes into account indentation
     * The method generates a formatted string containing the details of a single
     * post. The format is as follows:
     *  ID: [post ID]
     *  Account: [account handle]
     *  No. endorsements: [number of endorsements received by the post] | No. comments: [number of comments received by the post]
     *  [post message]
     */
    @Override
    public String toString(int indentationLevel){
        // Indent for all other lines
        String indent = "";
        for (int i = 0; i < indentationLevel; i++) {
            indent += "    ";
        }

        // Indent for first line
        String babyIndent = "";
        for (int i = 0; i < indentationLevel-1 ; i++) {
            babyIndent += "    ";
        }

        // Building the string
        String toString = babyIndent +"|\n" +
                babyIndent + "| > "+ "Id: " + this.getId() + "\n" +
                indent +"Account: " + this.getAuthor().getHandle() + "\n" +
                indent + "No. endorsements: " + getEndorsementCount() + " | No. comments: " + this.getComments().size() + "\n" +
                indent + this.getContents() + "\n";

        return toString;
    }
}
```

File - Original.java

```java
package socialmedia;

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

/**
 * Class for original objects.
 *
 * @author 700037512, 700074221
 */
public class Original extends Post implements Serializable {
    private ArrayList<Comment> comments;
    private ArrayList<Endorsement> endorsements;
    private ArrayList<Comment> deletedComments;

    /**
     * Constructor for Original post objects.
     * @param author The author object corresponding to who is creating the post.
     * @param contents The contents string of the post.
     * @throws InvalidPostException  if the message is empty or has more than
     *                               100 characters.
     * @throws HandleNotRecognisedException if the handle does not match to any
     *                                      account in the system.
     */
    public Original(Account author, String contents) throws InvalidPostException,
HandleNotRecognisedException {
        super(author, contents);
        isContentsValid(contents);
        this.comments = new ArrayList<>();
        this.endorsements = new ArrayList<>();
        this.deletedComments = new ArrayList<>();
        setPostType(PostType.ORIGINAL);
    }

    /**
     *Validation method checks if the contents of the message is valid.
     * @param contents What the author has said.
     * @throws InvalidPostException  if the message is empty or has more than
     *                               100 characters.
     */
    public static void isContentsValid(String contents) throws InvalidPostException{
        if (contents.equals("")) {
            throw new InvalidPostException("Contents is empty.");
        }
        if (contents.length() > 100) {
            throw new InvalidPostException("Contents is over 100 characters. It is too long.");
        }
    }

    /**
     * Adds comments to an arraylist of deleted posts.
     * @param comment the comment to be deleted.
     */
    @Override
    public void addDeletedComment(Comment comment){
        this.deletedComments.add(comment);
    }

    /**
     * Getter method for deleted comments.
     * @return An arraylist of deleted comments.
     */
    public ArrayList<Comment> getDeletedComments(){
        return deletedComments;
    }

    /**
     * Adds an endorsement to the current post's comment ArrayList.
     * @param endorsement the endorsement object to be added to the comment.
     */
    public void addEndorsementArrayList(Endorsement endorsement){
       endorsements.add(endorsement);
    }

    /**
```

File - Original.java

```java
     * Returns the ArrayList of Endorsement objects corresponding to the post.
     * @return all endorsements for the current post
     */
    public ArrayList<Endorsement> getEndorsements() {
        return endorsements;
    }

    /**
     * Returns the ArrayList of comments objects corresponding to the post.
     * @return all comments for the current post.
     */
    public ArrayList<Comment> getComments() {
        return comments;
    }

    /**
     * Adds a comment to the current post's comment ArrayList.
     * @param comment the contents of the comment.
     */
    public void addComment(Comment comment){
        this.comments.add(comment);
    }

   /**
    * The method generates a formatted string containing the details of a single
    * post. The format is as follows:
    *  ID: [post ID]
    *  Account: [account handle]
    *  No. endorsements: [number of endorsements received by the post] | No. comments: [number of comments received by the post]
    *  [post message]
    */
    @Override
    public String toString() {
        return "\nId: " + this.getId() + "\nAccount: " + this.getAuthor().getHandle() + "\nNo. endorsements: " +
                this.getEndorsementCount() + " | No. comments: " + comments.size() + "\n" + this.getContents() + "\n";
    }
}
```

File - PostType.java

```java
package socialmedia;

/**
 * PostType enum for setting/getting the type of post
 */
public enum PostType {
    ORIGINAL,
    COMMENT,
    ENDORSEMENT,
    DELETED
}

```

File - Endorsement.java

```java
package socialmedia;

import java.io.Serializable;

/**
 * Class for endorsement objects.
 *
 * @author 700037512, 700074221
 */
public class Endorsement extends Post implements Serializable {
    private Post parent;
    /**
     * Constructor for endorsement objects, adds an endorsement to the parent's endorsement counter.
     * @param author The author object corresponding to who is creating the comment.
     * @param contents The contents string of the comment.
     * @param parent The parent (Original/Comment) object that the comment is commenting on.
     * @throws HandleNotRecognisedException if the handle does not match to any
     *                                      account in the system.
     * @throws NotActionablePostException if the ID refers to a endorsement post.
     *                                    Endorsement posts are not endorsable.
     *                                    Endorsements are not transitive. For
     *                                    instance, if post A is endorsed by post
     *                                    B, and an account wants to endorse B, in
     *                                    fact, the endorsement must refers to A.
     */
    public Endorsement(Account author, String contents, Post parent) throws
            HandleNotRecognisedException, NotActionablePostException {
        super(author, contents);
        this.setContents(parent.getContents());
        this.parent = parent;
        parent.getAuthor().addEndorsement();
        setPostType(PostType.ENDORSEMENT);
        author.addEndorsement();
        parent.addEndorsement();
    }

    /**
     * Returns the parent (original/comment) post of the current endorsement post.
     * @return The parent of the current endorsement object.
     */
    public Post getParent() {
        return parent;
    }

    /**
     * Remove Endorsement from parent comment.
     */
    public void removeEndorsementFromParent(){
        this.getParent().removeEndorsement();
    }


    /**
     * Returns a string for the endorsement in the form "EP@" + [endorsed account handle] + ": " + [endorsed message]
     * @return The endorsement as a string
     */
    @Override
    public String toString(){
        String outputStr = "EP@" + this.getAuthor().getHandle() + ": " + this.getContents();
        return outputStr;
    }
}
```

File - SocialMedia.java

```java
package socialmedia;

import java.io.*;
import java.util.*;
import java.lang.*;

/**
 * Class for socialMedia objects.
 *
 * @author 700037512, 700074221
 */
public class SocialMedia implements SocialMediaPlatform, Serializable  {
    private HashMap<String, Account> accounts;
    private HashMap<Integer, Post> posts;

    /**
     * Constructor method for the social media platform to implement the interfaces.
     */
    public SocialMedia(){
        accounts = new HashMap<String, Account> ();
        posts = new HashMap<Integer, Post> ();
    }
    // Account/Post getter methods

    /**
     *Getter method returns a Hashmap of all the post with the Id as the key.
     * @return the Posts hashmap.
     */
    public HashMap<Integer, Post> getPosts() {
        return posts;
    }

    /**
     * Getter method returns a Hashmap of all the Accounts with handles as the key.
     * @return the Accounts hashmap.
     */
    public HashMap<String, Account> getAccounts() {
        return accounts;
    }

    /**
     * Creates a dummy post to set as a parent for comments who's parent has been deleted.
     * @return the dummy post found/newly created.
     */
    public Post dummyPost() {
        if (posts.containsKey(-1)) {
            return posts.get(-1);
        } else {
            Post dummyPost = new Post(-1);
            return dummyPost;
        }
    }

    /**
     * Finds an account object from accounts hashmap using the account's ID.
     * @param  ID the ID of the account being searched for.
     * @return the matching account object found if one is found.
     * @throws AccountIDNotRecognisedException if the ID does not match to any
     *                                         account in the system.
     */
    public Account getAccount(int ID) throws AccountIDNotRecognisedException {
        for (Account account : accounts.values()) {
            if (account.getId() == ID) {
                return account;
                }
            }
        throw new AccountIDNotRecognisedException("Account ID '" + ID +"' not recognised.");
    }

    /**
     * Finds an account object from accounts hashmap using the account's handle.
     * @param  handle the string handle of the account being searched for.
     * @return the matching account object found if one is found.
     * @throws HandleNotRecognisedException if the handle does not match to any
     *                                      account in the system.
```

```
76       */
77      public Account getAccount(String handle) throws HandleNotRecognisedException {
78          if (accounts.containsKey(handle)){
79               return accounts.get(handle);
80          } else{
81              throw new HandleNotRecognisedException("Handle '"+ handle +"' not recognised.");
82          }
83      }
84
85      /**
86       * Finds a post object from the posts hashmap using the post's ID.
87       * @param  ID The int ID of the post being searched for
88       * @return the matching post object if one is found
89       * @throws PostIDNotRecognisedException if the ID does not match to any
90       *                                      post on the system
91       */
92      public Post getPost(int ID) throws PostIDNotRecognisedException {
93          if (posts.containsKey(ID)){
94              return posts.get(ID);
95          } else{
96              throw new PostIDNotRecognisedException("Post ID '"+ ID +"' not recognised.");
97          }
98      }
99
100     /**
101      * Gets all posts by a specified author
102      * @param  author the author of the posts that are being searched for
103      * @return an ArrayList of posts that were found
104      */
105     public ArrayList<Post> getPostsByAuthor(Account author) {
106         ArrayList<Post> postsByAuthor = new ArrayList<>();
107         for (Post post : posts.values()) {
108             if ((post.getAuthor().getHandle()).equals(author.getHandle())) {
109                 postsByAuthor.add(post);
110             }
111         }
112         return postsByAuthor;
113     }
114
115     /**
116      * Remove comments and endorsements that were children of a removed post.
117      * @param comments the ArrayList of comments to be removed.
118      * @param endorsements the ArrayList of endorsements to be removed.
119      */
120     public void removeCommentsAndEndorsements(ArrayList<Comment> comments, ArrayList<Endorsement>
    endorsements) {
121         // Remove comments that were children the removed post
122         for (Comment comment : comments){
123             comment.makeOrphan();
124             comment.setParent(dummyPost());
125         }
126         // Remove endorsements that were children the removed post
127         for (Endorsement endorsement : endorsements){
128             this.posts.remove(endorsement.getId());
129         }
130     }
131
132
133     /**
134      * Removes posts in a provided ArrayList (helper method).
135      * @param posts the ArrayList of posts to be removed.
136      */
137     public void removePosts(ArrayList<Post> posts){
138         for (Post post : posts) {
139             if (post.getPostType().equals(PostType.COMMENT)) {
140                 ArrayList<Endorsement> endorsements = ((Comment) post).getEndorsements();
141                 ArrayList<Comment> comments = ((Comment) post).getComments();
142                 removeCommentsAndEndorsements(comments, endorsements);
143                 Post parent = ((Comment) post).getParent();
144                 parent.removeComment((Comment) post);
145                 post.setPostType(PostType.DELETED);
146                 this.posts.remove(post.getId());
147             }
148             else if (post.getPostType().equals(PostType.ORIGINAL)) {
149                 ArrayList<Comment> comments = ((Original) post).getComments();
```

```
150                 ArrayList<Endorsement> endorsements = ((Original) post).getEndorsements();
151                 removeCommentsAndEndorsements(comments, endorsements);
152                 this.posts.remove(post.getId());
153             }
154             else if (post.getPostType().equals(PostType.ENDORSEMENT)) {
155                 this.posts.remove(post.getId());
156                 ((Endorsement) post).removeEndorsementFromParent();
157             }
158         }
159     }
160 
161     // Main methods
162 
163     @Override
164     public int createAccount(String handle, String description) throws IllegalHandleException,
    InvalidHandleException {
165         if (accounts.containsKey(handle)){
166             throw new IllegalHandleException("Handle '"+ handle +"' has already been claimed.");
167         } else{
168             Account newAccount = new Account(handle, description);
169             int ID = newAccount.getId();
170             accounts.put(handle, newAccount);
171             return ID;
172         }
173     }
174 
175     @Override
176     public void removeAccount(String handle) throws HandleNotRecognisedException {
177         Account accountToBeRemoved = getAccount(handle);
178         ArrayList<Post> postsByAuthor = getPostsByAuthor(accountToBeRemoved);
179         accountToBeRemoved.clearAccount();
180         accounts.remove(handle);
181         removePosts(postsByAuthor);
182     }
183 
184     @Override
185     public void updateAccountDescription(String handle, String description) throws
    HandleNotRecognisedException {
186         Account accountToUpdate = getAccount(handle);
187         accountToUpdate.setDescription(description);
188     }
189 
190     @Override
191     public int getTotalOriginalPosts() {
192         int originalPostCount = 0;
193         for (Post post : posts.values()) {
194             if (post.getPostType().equals(PostType.ORIGINAL)) {
195                 originalPostCount++;
196             }
197         }
198         return originalPostCount;
199     }
200 
201     @Override
202     public int getTotalEndorsmentPosts() {
203         int endorsementPostCount = 0;
204         for (Post post : posts.values()) {
205             if (post.getPostType().equals(PostType.ENDORSEMENT)) {
206                 endorsementPostCount++;
207             }
208         }
209         return endorsementPostCount;
210     }
211 
212     @Override
213     public int getTotalCommentPosts() {
214         int commentPostCount = 0;
215         for (Post post : posts.values()) {
216             if (post.getPostType().equals(PostType.COMMENT)) {
217                 commentPostCount++;
218             }
219         }
220         return commentPostCount;
221     }
222 
```

File - SocialMedia.java

```
223     @Override
224     public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
225         if (accounts.containsKey(handle)){
226             throw new IllegalHandleException("Handle '"+ handle +"' has already been claimed.");
227         } else{
228             Account newAccount = new Account(handle);
229             int ID = newAccount.getId();
230             accounts.put(handle, newAccount);
231             return ID;
232         }
233     }
234 
235     @Override
236     public void removeAccount(int id) throws AccountIDNotRecognisedException {
237         Account accountToBeRemoved = getAccount(id);
238 
239         ArrayList<Post> postsByAuthor = getPostsByAuthor(accountToBeRemoved);
240         accountToBeRemoved.clearAccount();
241         accounts.remove(accountToBeRemoved.getHandle());
242 
243         // Dealing with posts made by this account
244         removePosts(postsByAuthor);
245     }
246 
247     @Override
248     public void changeAccountHandle(String oldHandle, String newHandle) throws
    HandleNotRecognisedException,
249             IllegalHandleException, InvalidHandleException {
250         Account accountToUpdate = getAccount(oldHandle);
251         accountToUpdate.setHandle(newHandle);
252         accounts.remove(oldHandle);
253         accounts.put(newHandle, accountToUpdate);
254     }
255 
256     @Override
257     public String showAccount(String handle) throws HandleNotRecognisedException {
258         Account accountToShow = getAccount(handle);
259         return accountToShow.toString();
260     }
261 
262     @Override
263     public int createPost(String handle, String message) throws HandleNotRecognisedException,
    InvalidPostException {
264         Account author = getAccount(handle);
265         Original newPost = new Original(author, message);
266         author.addPost(newPost);
267         posts.put(newPost.getId(), newPost);
268         return newPost.getId();
269     }
270 
271     @Override
272     public int endorsePost(String handle, int id) throws HandleNotRecognisedException,
    PostIDNotRecognisedException,
273             NotActionablePostException {
274         Account endorsingAccount = getAccount(handle);
275         Post post = posts.get(id);
276         Account postAuthorAccount = post.getAuthor();
277         Endorsement newEndorsement = new Endorsement(endorsingAccount, post.getContents(), post);
278         post.addEndorsementList(newEndorsement, post);
279         endorsingAccount.addPost(newEndorsement);
280         posts.put(newEndorsement.getId(), newEndorsement);
281         return newEndorsement.getId();
282     }
283 
284     @Override
285     public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
286             PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
287         Account commentingAccount = getAccount(handle);
288         Post post = getPost(id);
289         Post originalParent;
290         if (post.getPostType().equals(PostType.ORIGINAL)) {
291             originalParent = post;
292         } else {
293             originalParent = ((Comment)post).getOriginalParent();
294         }
```

File - SocialMedia.java

```
295         Comment newComment = new Comment(commentingAccount, message, getPost(id), originalParent);
296         commentingAccount.addPost(newComment);
297         posts.put(newComment.getId(), newComment);
298         return newComment.getId();
299     }
300 
301     @Override
302     public void deletePost(int id) throws PostIDNotRecognisedException {
303         Post postToRemove = getPost(id);
304         ArrayList<Post> posts = new ArrayList();
305         posts.add(postToRemove);
306         removePosts(posts);
307     }
308 
309     @Override
310     public String showIndividualPost(int id) throws PostIDNotRecognisedException {
311         Post post = getPost(id);
312         String stringyPost = post.toString();
313         return stringyPost;
314     }
315 
316     @Override
317     public StringBuilder showPostChildrenDetails(int id) throws PostIDNotRecognisedException,
318             NotActionablePostException {
319         StringBuilder showPostChildren = new StringBuilder();
320         ArrayList<Comment> comments = new ArrayList<Comment>();
321         Post post = getPost(id);
322 
323         // Creating and adding parent comments to comments arraylist
324         comments = ((Original) post).getComments();
325 
326         int indentationLevel = 1;
327         showPostChildren.append(post.toString());
328         //System.out.println(post.toString();
329         if (post.getPostType().equals(PostType.COMMENT)) {
330             postHelper(comments, indentationLevel, showPostChildren);
331         } else {
332             postHelper(comments, indentationLevel, showPostChildren);
333         }
334         return showPostChildren;
335     }
336 
337     /**
338      * Helper method for showPostChildrenDetails, iterates trough provided comments ArrayList, sorts
    out indentation,
339      * recursively calls showPostChildrenDetails.
340      * @param comments The comment ArrayList provided for the post.
341      * @param indentationLevel The level of indentation on the post.
342      * @param showPostChildren Stringbuilder for the ShowPostChildrenDetails method.
343      * @return a StringBuilder that returns the post and it's children.
344      */
345     public StringBuilder postHelper(ArrayList<Comment> comments, int indentationLevel, StringBuilder
    showPostChildren) {
346         for (Comment comment: comments){
347             if (comment.getPostType().equals(PostType.DELETED)){
348                continue;
349             } else {
350                 String commentString = "";
351                 commentString += comment.toString(indentationLevel);
352                 showPostChildren.append(commentString);
353             }
354             if(!(comment.getComments().isEmpty())){
355                 postHelper(comment.getComments(), indentationLevel+1, showPostChildren);
356             }
357         }
358         return showPostChildren;
359     }
360 
361     @Override
362     public int getMostEndorsedPost() {
363         Post mostPopularPost = null;
364         for(Post post : posts.values()){
365             int popularity = -1;
366             if (post.getEndorsementCount() > popularity) {
367                 popularity = post.getEndorsementCount();
```

```
368                 mostPopularPost = post;
369             }
370         }
371         return mostPopularPost.getId();
372     }
373 
374     @Override
375     public int getMostEndorsedAccount() {
376         Account mostPopularAccount = null;
377         for (Account account : accounts.values()) {
378             int popularity = -1;
379             if (account.getEndorsementCount() > popularity) {
380                 popularity = account.getEndorsementCount();
381                 mostPopularAccount = account;
382             }
383         }
384         return mostPopularAccount.getId();
385     }
386 
387     @Override
388     public int getNumberOfAccounts(){
389         return accounts.size();
390     }
391 
392     @Override
393     public void savePlatform(String filename) throws IOException {
394         try{
395             FileOutputStream out = new FileOutputStream(filename);
396             ObjectOutputStream outStream = new ObjectOutputStream(out);
397             outStream.writeObject(this);
398             outStream.close();
399             out.close();
400             System.out.println("Platform saved in " + filename);
401         } catch (IOException e){
402             System.out.println("An error occurred.");
403             e.printStackTrace();
404         }
405 
406     }
407 
408     @Override
409     public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
410         try {
411                 ObjectInputStream  input = new ObjectInputStream(new FileInputStream(filename));
412                 Object object = input.readObject();
413                 if (object instanceof SocialMedia){
414                     this.accounts = ((SocialMedia) object).accounts;
415                     this.posts = ((SocialMedia)object).posts;
416                 }
417                 input.close();
418                 System.out.println("Platform loaded from " + filename);
419         } catch (ClassNotFoundException e) {
420             System.out.println("Class not found exception.");
421             e.printStackTrace();
422         } catch (IOException e) {
423             System.out.println("An error occurred.");
424             e.printStackTrace();
425         }
426     }
427 
428     @Override
429     public void erasePlatform(){
430         accounts = new HashMap<String, Account>();
431         posts = new HashMap<Integer, Post>();
432         Account.setNewId(0);
433         Post.setNewPostId(0);
434     }
435 }
436 
```

File - SocialMediaPlatformTestApp.java

```java
import socialmedia.*;

import java.io.IOException;
import java.util.HashMap;

/**
 * Testing for all SocialMedia.java functionality
 *
 * @author  700037512, 700074221
 */
public class SocialMediaPlatformTestApp {

    /**
     * Print helper method
     * @param data the string to be printed to console
     */
    public static void print(String data){System.out.println(data);  }

    /**
     * Print helper method
     * @param data the int to be printed to console
     */
    public static void print(int data){System.out.println(data);  }

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

        SocialMediaPlatform platform = new SocialMedia();

        assert (platform.getNumberOfAccounts() == 0) : "Innitial SocialMediaPlatform not empty as required.";
        assert (platform.getTotalOriginalPosts() == 0) : "Innitial SocialMediaPlatform not empty as required.";
        assert (platform.getTotalCommentPosts() == 0) : "Innitial SocialMediaPlatform not empty as required.";
        assert (platform.getTotalEndorsmentPosts() == 0) : "Innitial SocialMediaPlatform not empty as required.";

        /**
         * Tests for account methods
         * createAccount □, showAccount □, updateAccountDescription □, changeAccountHandle □, getNumberOfAccounts □,
         * getMostEndorsedAccount □ removeAccount □
         */
        try {
            /**
             * createAccount
             * - Handle cant be same as another (IllegalHandleException) □
             * - Handle can't be empty, have more than 30 characters, or have a space in it ( InvalidHandleException) □
             * Should return correct account IDs - 0, 1 □
             */

            int my_handle = platform.createAccount("my_handle");
            assert (my_handle == 1) : "Incorrect account ID returned for createAccount.";
            int test_1 = platform.createAccount("test_1", "testing");
            assert (test_1 == 2) : "Incorrect account ID returned ";

            // IllegalHandleException
            try {
                platform.createAccount("test_1", "testing"); // Should throw IllegalHandleException
            } catch (IllegalHandleException e) {
                assert (true) : "IllegalHandleException not thrown for repeated account handles";
            }

            // InvalidHandleException
            try {
                platform.createAccount("", "testing"); // Should throw InvalidHandleException
            } catch (InvalidHandleException e) {
                assert (true) : "InvalidHandleException not thrown for empty account handles";
```

File - SocialMediaPlatformTestApp.java

```
70              }
71
72              /**
73               * showAccount
74               * - Provided handle must exist (HandleNotRecognisedException) 
75               * - Account info should be returned as a string in the correct form 
76               */
77
78              // HandleNotRecognisedException
79              try {
80                  platform.showAccount("FakeHandle"); // Should throw HandleNotRecognisedException
81              } catch (HandleNotRecognisedException e) {
82                  assert (true) : "HandleNotRecognisedException not thrown for non-existent accounts";
83              }
84
85              String accountInfo = platform.showAccount("my_handle");
86              //print(accountInfo);
87              String accountInfo1 = platform.showAccount("test_1");
88              //print(accountInfo1);
89
90              /**
91               * updateAccountDescription
92               * - Provided handle must exist (HandleNotRecognisedException)
93               * - Using showAccount after changing account description should return a string in the
    correct form 
94               */
95              // HandleNotRecognisedException
96              try {
97                  platform.updateAccountDescription("FakeHandle", "test description"); // Should throw
98                  // HandleNotRecognisedException
99              } catch (HandleNotRecognisedException e) {
100                 assert (true) : "HandleNotRecognisedException not thrown for non-existent accounts";
101             }
102             platform.updateAccountDescription("my_handle", " test description");
103             accountInfo = platform.showAccount("my_handle");
104             //print(accountInfo);
105
106             /**
107              * changeAccountHandle
108              * - Provided handle must exist (HandleNotRecognisedException)
109              * - Handle cant be same as another (IllegalHandleException)
110              * - Handle can't be empty, have more than 30 characters, or have a space in it (
    InvalidHandleException)
111              * - Using showAccount after changing account description should return the correct string
112              */
113             platform.changeAccountHandle("my_handle", "new_Handle");
114             accountInfo = platform.showAccount("new_Handle");
115             //print(accountInfo);
116
117             /**
118              * getNumberOfAccounts
119              * - Using getNumberOfAccounts should return int 2 as two accounts currently exist on the
    system
120              */
121             assert (platform.getNumberOfAccounts() == 2 ) : "getNumberOfAccounts calculated incorrect
    account count";
122
123             /**
124              * getMostEndorsedAccount
125              * - Using getMostEndorsedAccount should return int 1 as it is the account ID with the most
     endorsements
126              * currently 
127              */
128             int postID = platform.createPost("test_1", "testPost");
129             platform.endorsePost("new_Handle", postID);
130             assert (platform.getMostEndorsedAccount() == 1 ) : "getMostEndorsedAccount returned
    incorrect account ID";
131
132             /**
133              * removeAccount
134              * - Account ID must exist on the system (AccountIDNotRecognisedException) 
135              * - Using getNumberOfAccounts after removing an account should return int 2 as one of the
    3 accounts has been
136              * removed 
137              */
```

File - SocialMediaPlatformTestApp.java

```
138             platform.removeAccount(1);
139             assert (platform.getNumberOfAccounts() == 2) : "number of accounts registered in the system
     does not match";
140 
141 
142         } catch (IllegalHandleException e) {
143             e.printStackTrace();
144         } catch (InvalidHandleException e) {
145             e.printStackTrace();
146         } catch (AccountIDNotRecognisedException e) {
147             e.printStackTrace();
148         } catch (HandleNotRecognisedException e) {
149             e.printStackTrace();
150         } catch (InvalidPostException e) {
151             e.printStackTrace();
152         } catch (PostIDNotRecognisedException e) {
153             e.printStackTrace();
154         } catch (NotActionablePostException e) {
155             e.printStackTrace();
156         }
157 
158         /**
159          * Tests for post methods
160          * createPost ☐, endorsePost ☐, commentPost ☐, deletePost ☐, showIndividualPost ☐,
161          * showPostChildrenDetails (+ postHelper) ☐, getTotalOriginalPosts , getTotalEndorsmentPosts ☐,
162          * getTotalCommentPosts ☐, getMostEndorsedPost ☐
163          */
164         try {
165             platform.createAccount("test_2");
166             /**
167              * createPost
168              * - Account handle must exist on system (HandleNotRecognisedException) ☐
169              * - Post contents must be valid (InvalidPostException) ☐
170              */
171             // createPost, showIndividualPost
172             int post = platform.createPost("test_1", "Main test post");
173 
174             /**
175              * showIndividualPost
176              * - Post ID should exist on system (PostIDNotRecognisedException) ☐
177              * - Using showIndividualPost for the post previously created should return the correct
    string ☐
178              */
179             //print(platform.showIndividualPost(post));
180 
181             /**
182              * showAccount
183              * - Account handle should exist on the system (HandleNotRecognisedException) ☐
184              * - Using showAccount("test_1") should return the correct string ☐
185              */
186             String test_1 = platform.showAccount("test_1");
187             //print(test_1);
188 
189             /**
190              * endorsePost
191              * - Account handle should exist on the system (HandleNotRecognisedException) ☐
192              * - Post ID should exist on system (PostIDNotRecognisedException) ☐
193              * - Endorsement posts should not be endorsable (NotActionablePostException) ☐
194              * - Using showIndividualPost for the previously created post should return the correct
    string ☐
195              */
196             //print(platform.showAccount("test_1"));
197 
198             int endorse = platform.endorsePost("test_2", 3);
199             //print(platform.showIndividualPost(endorse));
200             //print(platform.showIndividualPost(3));
201             //print(platform.showAccount("test_1"));
202 
203             /**
204              * commentPost
205              * - Account handle should exist on the system (HandleNotRecognisedException) ☐
206              * - Post ID should exist on system (PostIDNotRecognisedException) ☐
207              * - Endorsement posts should not be commentable (NotActionablePostException) ☐
208              * - Using showIndividualPost should return the correct string ☐
209              */
```

File - SocialMediaPlatformTestApp.java

```
210             int firstComment = platform.commentPost("test_2", 3, "Comment under main test post 1");
211             int secondComment = platform.commentPost("test_1", firstComment, "Comment under first
    comment");
212             platform.commentPost("test_1", secondComment, "Comment under second comment");
213             platform.commentPost("test_2", 3, "Comment under main test post 1");
214 
215             //print(platform.showIndividualPost(firstComment));
216             //print(platform.showIndividualPost(3));
217 
218             /**
219              * showPostChildrenDetails
220              * - Post ID should exist on system (PostIDNotRecognisedException) 
221              * - Endorsement posts should not be commentable (NotActionablePostException) 
222              * - Using showPostChildrenDetails should return the correct string 
223              */
224             platform.showPostChildrenDetails(3);
225             System.out.println(platform.showPostChildrenDetails(3));
226 
227             /**
228              * getTotalCommentPosts
229              * - Using getTotalCommentPosts should return int 4 
230              */
231             int totalComment = platform.getTotalCommentPosts();
232             assert (totalComment == 4) : "getTotalCommentPosts comment count not calculated correctly";
233 
234             /**
235              * getTotalEndorsmentPosts
236              * - Using getTotalEndorsmentPosts should return int 1 
237              */
238             int totalEndorse = platform.getTotalEndorsmentPosts();
239             assert (totalEndorse == 1) : "getTotalEndorsmentPosts endorsement count not calculated
    correctly";
240 
241             /**
242              * getTotalOriginalPosts
243              * - Using getTotalOriginalPosts should return int 2 
244              */
245             int totalOriginal = platform.getTotalOriginalPosts();
246             assert (totalOriginal == 2) : "getTotalOriginalPosts original count not calculated
    correctly";
247 
248             /**
249              * getMostEndorsedPost
250              * Using getMostEndorsedPost should return the correct string 
251              */
252             int postID = platform.getMostEndorsedPost();
253             assert(postID == 8) : "incorrect post returned for getMostEndorsedPost";
254 
255             /**
256              * deletePost
257              * - Post ID should exist on system (PostIDNotRecognisedException)
258              * - Using getTotalOriginalPosts should return int 1 
259              */
260             platform.deletePost(1);
261             try {
262                 ((SocialMedia) platform).getPost(1);
263             } catch (PostIDNotRecognisedException e) {
264                 assert(true) : "deletePost did not remove post as expected";
265             }
266         } catch(PostIDNotRecognisedException e) {
267             e.printStackTrace();
268         }  catch(InvalidPostException e) {
269             e.printStackTrace();
270         } catch (HandleNotRecognisedException e) {
271             e.printStackTrace();
272         }catch (NotActionablePostException e) {
273             e.printStackTrace();
274         } catch (IllegalHandleException e){
275             e.printStackTrace();
276         } catch (InvalidHandleException e) {
277             e.printStackTrace();
278         }
279 
280         /**
281          * Tests for platform methods
```

File - SocialMediaPlatformTestApp.java

```
282          * savePlatform , loadPlatform , erasePlatform 
283          */
284         try {
285             /**
286              * savePlatform
287              * - IOException if file issues 
288              * - Should create a file called "testSave" 
289              */
290             platform.savePlatform("testSave");
291             // Saving hashmaps locally so that they can be used to see if loadPlatform() works
292             HashMap<Integer, Post> localPosts =  ((SocialMedia) platform).getPosts();
293             HashMap<String, Account> localAccounts = ((SocialMedia) platform).getAccounts();
294 
295             /**
296              * erasePlatform
297              * - Should clear all arraylists, hashmaps and counters 
298              */
299             platform.erasePlatform();
300             assert (platform.getNumberOfAccounts() == 0) : "Accounts hashmap has not been reset";
301             assert (platform.getTotalOriginalPosts() == 0) : "Posts hashmap has not been reset";
302 
303             /**
304              * loadPlatform
305              * - Hashmaps imported should be the same as before the platform was erased
306              */
307             platform.loadPlatform("testSave");
308             String test_2 = ((SocialMedia) platform).getAccount(3).getHandle();
309             assert(test_2.equals("test_2")) : "Loaded accounts hashmap does not match original hashmap"
    ;
310 
311             String main_test_post = ((SocialMedia) platform).getPost(3).getContents();
312             assert(main_test_post.equals("Main test post")) : "Loaded posts hashmap does not match
    original hashmap";
313 
314         } catch (IOException e) {
315             e.printStackTrace();
316         } catch (ClassNotFoundException e) {
317             e.printStackTrace();
318         } catch (AccountIDNotRecognisedException e) {
319             e.printStackTrace();
320         } catch (PostIDNotRecognisedException e) {
321             e.printStackTrace();
322         }
323     }
324 }
325 
```