File - Account.java

```java
package socialmedia;
import java.io.Serializable;

/**
 * The class that represents accounts in this social media app.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class Account implements Serializable{
    private int id;
    private String handle;
    private String description;

    /**
     * Instantiates an account object.
     *
     * @param id unique id of the account
     * @param handle unique handle that the account can be referenced by
     * @param description description of the account
     */
    public Account(int id, String handle,String description) {
        this.id = id;
        this.handle = handle;
        this.description = description;
    }

    /**
     * Returns id associated with this account.
     * @return account id
     */
    public int getId() {
        return id;
    }

    /**
     * Returns the handle associated with this account
     * @return account handle
     */
    public String getHandle() {
        return handle;
    }

    /**
     * Sets the handle of this account.
     * @param handle new handle to be allocated to this account
     */
    public void setHandle(String handle) {
        this.handle = handle;
    }

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

    /**
     * toString override method which provides more detail about the account
     * object.
     * @return account information string
     */
    @Override
```

File - Account.java

```
67     public String toString() {
68         return "ID: " + id + "\n" +
69                 "Handle: " + handle + '\n' +
70                 "Description: " + description;
71 }
72 }
73
```

File - CommentPost.java

```java
package socialmedia;

import java.util.ArrayList;

/**
 * The class that represents comments in this social media app.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class CommentPost extends Post {
    private Post post;
    private ArrayList<EndorsementPost> endorsements;
    private ArrayList<CommentPost> comments;

    /**
     * This method is used to generate a comment post and create empty lists
     * for endorsements and comments which are directed at it. Also sets post
     * which stores a reference to the parent post of this comment.
     * @param id  id of comment
     * @param author Account details/ passed account object
     * @param message comment string
     * @param post original post or comment being replied to
     */
    public CommentPost(int id, Account author, String message, Post post) {
        super(id, author, message);
        this.post = post;
        endorsements = new ArrayList<EndorsementPost>();
        comments = new ArrayList<CommentPost>();
    }

    /**
     * Getter method for comments list.
     * @return ArrayList of comments
     */
    public ArrayList<CommentPost> getComments() {
        return comments;
    }

    /**
     * Getter method for endorsements list.
     * @return ArrayList of endorsements
     */
    public ArrayList<EndorsementPost> getEndorsements() {
        return endorsements;
    }

    /**
     * This method is used to add comments to the comments ArrayList in this
     * CommentPost.
     * @param commentPost for passing the commentPost object
     */
    public void addComment(CommentPost commentPost) {
        comments.add(commentPost);
    }

    /**
     * This method is used to remove comments from the comments ArrayList in
     * this CommentPost.
     * @param commentPost passes the commentPost object
     */
    public void removeComment(CommentPost commentPost) {
        comments.remove(commentPost);
    }

```

File - CommentPost.java

```
67      /**
68       * This method is used to add endorsements to the endorsement ArrayList in
69       * this CommentPost.
70       * @param endorsementPost passes the EndorsementPost object
71       */
72      public void addEndorsement(EndorsementPost endorsementPost) {
73          endorsements.add(endorsementPost);
74      }
75
76      /**
77       * This method is used to remove endorsements from the endorsement ArrayList
78       * in this CommentPost.
79       * @param endorsementPost passes the EndorsementPost object
80       */
81      public void removeEndorsement(EndorsementPost endorsementPost) {
82          endorsements.remove(endorsementPost);
83      }
84
85      /**
86       * This setter method is used to update post.
87       * @param post passes the post object
88       */
89      public void setPost(Post post) {
90          this.post = post;
91      }
92
93      /**
94       * This getter method gets the post.
95       * @return gets the post
96       */
97      public Post getPost() {
98          return post;
99      }
100
101     /**
102      * toString override method which provides more detail about the comment
103      * post object.
104      * @return comment post information string
105      */
106     @Override
107     public String toString() {
108         return "ID: " + super.getId() + "\nAccount: "
109                 + super.getAuthor().getHandle()
110                 + "\nNo. endorsements: " + endorsements.size()
111                 + " | No. comments: " + comments.size() + "\n"
112                 + super.getMessage();
113     }
114 }
115
```

File - EndorsementPost.java

```java
package socialmedia;

/**
 * The class that represents endorsements in this social media app.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class EndorsementPost extends Post{
    private Post post;

    /**
     * This method is used to generate an endorsement with a given post id,
     * author and message along with post which acts as a pointer
     * to the original or comment post being endorsed.
     * @param id id of the endorsement
     * @param author Account details/ passed account object
     * @param message copy of parent post's message
     * @param post original post or comment being endorsed
     */
    public EndorsementPost(int id, Account author, String message, Post post) {
        super(id,author,message);
        this.post=post;
    }

    /**
     * Returns the parent post that this endorsement points to.
     * @return endorsed post
     */
    public Post getPost() {
        return post;
    }
}

```

File - GenericPost.java

```java
package socialmedia;

import java.util.ArrayList;

/**
 * The class that represents a generic post in this social media app that
 * comments can be redirected to if their parent posts have been deleted.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class GenericPost extends Post{
    private ArrayList<CommentPost> comments;

    /**
     * Instantiates a generic post object with an id of 0, a default message and
     * an ArrayList to hold comments whose original parent posts have been
     * deleted.
     */
    public GenericPost() {
        super(0, null, "The original content was removed"
                + " from the system and is no longer available.");
        comments = new ArrayList<CommentPost>();
    }

    /**
     * Returns the list of comments whose original parent posts have been
     * deleted.
     * @return list of comments which have no parent post
     */
    public ArrayList<CommentPost> getComments() {
        return comments;
    }
}

```

File - OriginalPost.java

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

/**
 * The class that represents original (top-level) posts in this social media
 * app.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class OriginalPost extends Post {
    private ArrayList<EndorsementPost> endorsements;
    private ArrayList<CommentPost> comments;

    /**
     * Instantiates an original post object.
     * @param id unique id of the post
     * @param author account that created the post
     * @param message message body of the post
     */
    public OriginalPost(int id, Account author, String message) {
        super(id, author, message);
        endorsements = new ArrayList<EndorsementPost>();
        comments = new ArrayList<CommentPost>();
    }

    /**
     * Returns the list of comments on this original post.
     * @return comments list
     */
    public ArrayList<CommentPost> getComments() {
        return comments;
    }

    /**
     * Returns the list of endorsements on this original post.
     * @return endorsements list
     */
    public ArrayList<EndorsementPost> getEndorsements() {
        return endorsements;
    }

    /**
     * Adds a comment to the list of comments on this original post.
     * @param commentPost comment to be added
     */
    public void addComment(CommentPost commentPost) {
        comments.add(commentPost);
    }

    /**
     * Removes a comment from the list of comments on this original post.
     * @param commentPost comment to be removed
     */
    public void removeComment(CommentPost commentPost) {
        comments.remove(commentPost);
    }

    /**
     * Adds an endorsement to the list of endorsements on this original post.
     * @param endorsementPost endorsement to be added
     */
    public void addEndorsement(EndorsementPost endorsementPost) {
        endorsements.add(endorsementPost);
    }
```

File - OriginalPost.java

```
67
68      /**
69       * Removes an endorsement from the list of endorsements on this original
70       * post.
71       * @param endorsementPost endorsement to be removed
72       */
73      public void removeEndorsement(EndorsementPost endorsementPost) {
74          endorsements.remove(endorsementPost);
75      }
76
77      /**
78       * toString override method which provides more detail about the original
79       * post object.
80       * @return original post information string
81       */
82      @Override
83      public String toString() {
84          return "ID: " + super.getId() + "\nAccount: "
85                  + super.getAuthor().getHandle()
86                  + "\nNo. endorsements: " + endorsements.size()
87                  + " | No. comments: " + comments.size() + "\n"
88                  + super.getMessage();
89      }
90  }
91
```

File - Post.java

```java
package socialmedia;
import java.io.Serializable;

/**
 * The class that represents all types of posts in this social media app.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public abstract class Post implements Serializable {
    // TODO - make private and use getters and setters
    private int id;
    private Account author;
    private String message;

    /**
     * Aids in instantiating a child of this class by assigning values for id,
     * author and message.
     * @param id id of the post
     * @param author author of the post
     * @param message message body of the post
     */
    Post(int id, Account author, String message) {
        this.id = id;
        this.author = author;
        this.message = message;
    }

    /**
     * Returns the id of the post object.
     * @return post id
     */
    public int getId() {
        return id;
    }

    /**
     * Returns the message body of the post object.
     * @return post message
     */
    public String getMessage() {
        return message;
    }

    /**
     * Returns the author of the post.
     * @return post author
     */
    public Account getAuthor() {
        return author;
    }
}

```

File - SocialMedia.java

```java
package socialmedia;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
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.LinkedHashMap;
import java.util.Map.Entry;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Main class for this social media app that implements most of the
 * SocialMediaPlatform interface's methods.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class SocialMedia implements SocialMediaPlatform {

    ArrayList<Account> accounts;
    ArrayList<Post> posts;
    // Generic post for comments of deleted posts to be linked to
    GenericPost genPost;

    /**
     * Instantiates a social media platform object that stores accounts, posts
     * and a generic post which holds all comments that originally pointed to
     * posts that have since been deleted.
     */
    public SocialMedia() {
        accounts = new ArrayList<>();
        posts = new ArrayList<>();
        genPost = new GenericPost();
    }

    /**
     * Generates an id which is 1 higher than the existing highest one.
     * @param option option to choose either accounts (a) or posts (p)
     * @return the new id
     */
    public int idGenerator(String option) {
        int maxId = 0;
        switch (option) {
            case "a":
                for (Account acc : accounts) {
                    if (maxId < acc.getId()) {
                        maxId = acc.getId();
                    }
                }
                break;
            case "p":
                for (Post post : posts) {
                    if (maxId < post.getId()) {
                        maxId = post.getId();
                    }
                }
                break;
        }
        return ++maxId;
    }
```

File - SocialMedia.java

```
 67
 68     @Override
 69     public int createAccount(String handle)
 70             throws IllegalHandleException,InvalidHandleException {
 71         checkInvalidHandle(handle);
 72         checkIllegalHandle(handle);
 73         int id;
 74         id=idGenerator("a");
 75         accounts.add(new Account(id, handle, ""));
 76         return id;
 77     }
 78
 79     @Override
 80     public int createAccount(String handle, String description)
 81             throws IllegalHandleException, InvalidHandleException {
 82         checkInvalidHandle(handle);
 83         checkIllegalHandle(handle);
 84         int id;
 85         id=idGenerator("a");
 86         accounts.add(new Account(id, handle, description));
 87         return id;
 88     }
 89
 90     @Override
 91     public void removeAccount(int id) throws AccountIDNotRecognisedException {
 92         checkAccountIdExists(id);
 93         removeAccount(getAccountById(id));
 94     }
 95
 96     @Override
 97     public void removeAccount(String handle)
 98             throws HandleNotRecognisedException {
 99         checkHandleExists(handle);
100         removeAccount(getAccountByHandle(handle));
101     }
102
103     /**
104      * Removes an account from the system along with any of it's posts.
105      * @param account account to remove
106      */
107     public void removeAccount(Account account) {
108         for (Post post : posts) {
109             if (post.getAuthor().equals(account)) {
110                 try {
111                     deletePost(post.getId());
112                 } catch (PostIDNotRecognisedException e) {
113                     // IGNORE
114                 }
115             }
116         }
117         accounts.remove(account);
118     }
119
120     /**
121      * Checks if an account exists with the passed id and throws an exception
122      * if not.
123      * @param id id to be searched for in the system
124      * @throws AccountIDNotRecognisedException thrown if no account could be
125      *                                         found with the passed id
126      */
127     public void checkAccountIdExists(int id)
128             throws AccountIDNotRecognisedException {
129         if(getAccountById(id) == null){
130             throw new AccountIDNotRecognisedException("No account exists with"
131                     + " specified id.");
132         }
```

```
133     }
134
135     /**
136      * Checks if an account exists with the passed handle and throws an
137      * exception if not.
138      * @param handle handle to be searched for in the system
139      * @throws HandleNotRecognisedException thrown if no account could be found
140      *                                      found with the passed handle
141      */
142     public void checkHandleExists(String handle)
143             throws HandleNotRecognisedException {
144         if(getAccountByHandle(handle)==null){
145             throw new HandleNotRecognisedException("No account exists with"
146                     + " specified handle.");
147         }
148     }
149
150     /**
151      * Checks if the passed handle already belongs to an account in the system
152      * and throws an exception if it does.
153      * @param handle handle to be searched for in the system
154      * @throws IllegalHandleException thrown if an account already has the
155      *                                passed handle
156      */
157     public void checkIllegalHandle(String handle) throws IllegalHandleException{
158         if(getAccountByHandle(handle) != null){
159             throw new IllegalHandleException("Handle already exists!");
160         }
161     }
162
163     /**
164      * Checks if the passed handle conforms to the following rules for handles
165      * in this social media system: must not be empty or have more than 30
166      * characters and must not have white spaces.
167      * @param handle handle to be checked
168      * @throws InvalidHandleException thrown if the handle does not conform to
169      *                                the specified rules
170      */
171     public void checkInvalidHandle(String handle) throws InvalidHandleException{
172         if(handle.length()>30 || handle.length()<1 || handle.contains(" ")){
173             throw new InvalidHandleException("Invalid handle! Handle must"
174                     + " contain between 1 and 30 characters and not have white"
175                     + " spaces!");
176         }
177     }
178
179     @Override
180     public void changeAccountHandle(String oldHandle, String newHandle)
181             throws HandleNotRecognisedException,
182             IllegalHandleException, InvalidHandleException {
183         checkHandleExists(oldHandle);
184         checkInvalidHandle(newHandle);
185         checkIllegalHandle(newHandle);
186
187         getAccountByHandle(oldHandle).setHandle(newHandle);
188     }
189
190     @Override
191     public void updateAccountDescription(String handle, String description)
192             throws HandleNotRecognisedException {
193         checkHandleExists(handle);
194         getAccountByHandle(handle).setDescription(description);
195     }
196
197     @Override
198     public String showAccount(String handle)
```

File - SocialMedia.java

```
199             throws HandleNotRecognisedException {
200         checkHandleExists(handle);
201         Account showAcc = getAccountByHandle(handle);
202 
203         // Find post count and endorsement count
204         int postCount = 0;
205         int endorseCount = 0;
206         for (Post pst : posts) {
207             if (pst.getAuthor().equals(showAcc)) {
208                 postCount++;
209                 if (pst instanceof OriginalPost) {
210                     OriginalPost op = (OriginalPost) pst;
211                     endorseCount += op.getEndorsements().size();
212                 } else if (pst instanceof CommentPost) {
213                     CommentPost cp = (CommentPost) pst;
214                     endorseCount += cp.getEndorsements().size();
215                 }
216             }
217         }
218 
219         // Make strAppend
220         String strAppend = "\nPost count: " + postCount + "\nEndorse count: "
221                 + endorseCount;
222 
223         // Append strAppend
224         return getAccountByHandle(handle).toString() + strAppend;
225     }
226 
227     /**
228      * Returns the account object that matches the specified handle or null if
229      * none match it.
230      * @param handle handle of account to be found
231      * @return found account object or null if none exist with the passed handle
232      */
233     public Account getAccountByHandle(String handle) {
234         Account account = null;
235         for (Account acc : accounts) {
236             if (acc.getHandle().equals(handle)) {
237                 account = acc;
238                 break;
239             }
240 
241         }
242         return account;
243     }
244 
245     /**
246      * Returns the account object that matches the specified id or null if none
247      * match it.
248      * @param id id of account to be found
249      * @return found account object or null if none exist with the passed id
250      */
251     public Account getAccountById(int id) {
252         Account account = null;
253         for (Account acc : accounts) {
254             if (acc.getId() == id) {
255                 account = acc;
256                 break;
257             }
258 
259         }
260         return account;
261     }
262 
263     /**
264      * Checks if the passed post message conforms to the following rules for
```

File - SocialMedia.java

```
265      * posts in this social media system: must not be empty or contain more than
266      * 100 characters.
267      * @param message post message to be checked
268      * @throws InvalidPostException thrown if the post message does not conform
269      *                              to the specified rules
270      */
271     public void checkPostIsValid(String message) throws InvalidPostException {
272         if (message.length() < 1 || message.length() > 100) {
273             throw new InvalidPostException("Post message must be between 1 and"
274                     + " 100 characters!");
275         }
276     }
277 
278     @Override
279     public int createPost(String handle, String message)
280             throws HandleNotRecognisedException, InvalidPostException {
281         checkHandleExists(handle);
282         checkPostIsValid(message);
283         int id;
284         id = idGenerator("p");
285         posts.add(new OriginalPost(id, getAccountByHandle(handle), message));
286         return id;
287     }
288 
289     /**
290      * Returns the post object associated with the passed id or null if none
291      * match it.
292      * @param id id of post to be found
293      * @return found post object or null if none exist with the passed id
294      */
295     public Post postFinder(int id){
296         Post post = null;
297         for (Post pst:posts ){
298             if(pst.getId()==id){
299                 post=pst;
300             }
301         }
302         return post;
303     }
304 
305     /**
306      * Counts how many posts there are which are instances of the passed post
307      * class in the system.
308      * @param input post class to be counted
309      * @return amount of posts of the specified class in the system
310      */
311     public int postCount(Class<?> input) {
312         int c = 0;
313         for (Post pst : posts) {
314             if (input.isInstance(pst)) {
315                 c++;
316             }
317         }
318         return c;
319     }
320 
321 
322     @Override
323     public int endorsePost(String handle, int id)
324             throws HandleNotRecognisedException,
325             PostIDNotRecognisedException, NotActionablePostException {
326         checkHandleExists(handle);
327 
328         Post post = postFinder(id);
329         checkPostExists(post);
330         checkPostIsActionable(post);
```

```
331
332
333
334         int endoId = idGenerator("p");
335         String message = "EP@" + handle + ": " + post.getMessage();
336         Account account = getAccountByHandle(handle);
337         EndorsementPost newEndorsement = new EndorsementPost(endoId,
338                 account, message , post);
339
340         if (post instanceof OriginalPost) {
341             OriginalPost op = (OriginalPost) post;
342             for (EndorsementPost ep: op.getEndorsements()) {
343                 if(ep.getAuthor().equals(account)) {
344                     return ep.getId();
345                 }
346             }
347             op.addEndorsement(newEndorsement);
348         } else if (post instanceof CommentPost) {
349             CommentPost cp = (CommentPost) post;
350             for (EndorsementPost ep: cp.getEndorsements()) {
351                 if (ep.getAuthor().equals(account)) {
352                     return ep.getId();
353                 }
354             }
355             cp.addEndorsement(newEndorsement);
356         }
357
358         posts.add(newEndorsement);
359
360         return endoId;
361     }
362
363     /**
364      * Checks if the post object passes as input is able to be endorsed and
365      * commented on. This is not true if it is an endorsement post.
366      * @param post post object
367      * @throws NotActionablePostException thrown if the passed post object is an
368      *                                    endorsement post
369      */
370     public void checkPostIsActionable(Post post)
371             throws NotActionablePostException {
372         if (post instanceof EndorsementPost) {
373             throw new NotActionablePostException("Endorsement posts cannot be"
374                     + " endorsed or commented on.");
375         }
376     }
377
378     /**
379      * The method checks if the post object passed as input contains info and is
380      * not null.
381      * @param post - passes post Object.
382      * @throws PostIDNotRecognisedException - checks if post is null, if true
383      *                                        throws exception
384      */
385     public void checkPostExists(Post post) throws PostIDNotRecognisedException {
386         if (post == null) {
387             throw new PostIDNotRecognisedException("No post exists with"
388                     + " specified ID.");
389         }
390     }
391
392
393     @Override
394     public int commentPost(String handle, int id, String message)
395             throws HandleNotRecognisedException,
396             PostIDNotRecognisedException,
```

```
397             NotActionablePostException, InvalidPostException {
398         checkHandleExists(handle);
399 
400         Post post = postFinder(id);
401 
402         checkPostExists(post);
403         checkPostIsActionable(post);
404         checkPostIsValid(message);
405 
406         int commId = idGenerator("p");
407         CommentPost newComment = new CommentPost(commId,
408                 getAccountByHandle(handle), message, post);
409 
410         posts.add(newComment);
411 
412         if (post instanceof OriginalPost) {
413             OriginalPost op = (OriginalPost) post;
414             op.addComment(newComment);
415         } else if (post instanceof CommentPost) {
416             CommentPost cp = (CommentPost) post;
417             cp.addComment(newComment);
418         }
419         return commId;
420     }
421 
422     @Override
423     public void deletePost(int id) throws PostIDNotRecognisedException {
424         Post post = postFinder(id);
425         checkPostExists(post);
426         if (post instanceof OriginalPost) {
427             OriginalPost originalPost = (OriginalPost) post;
428             ArrayList<EndorsementPost> endorsements
429                     = originalPost.getEndorsements();
430             ArrayList<CommentPost> comments = originalPost.getComments();
431             for ( CommentPost o: comments){
432                 o.setPost(genPost);
433             }
434             ArrayList<CommentPost> genPostComments = genPost.getComments();
435             genPostComments.addAll(comments);
436             posts.removeAll(endorsements);
437 
438         } else if (post instanceof CommentPost) {
439             CommentPost commentPost = (CommentPost) post;
440             ArrayList<EndorsementPost> endorsements
441                     = commentPost.getEndorsements();
442             ArrayList<CommentPost> comments = commentPost.getComments();
443             for (CommentPost c : comments) {
444                 c.setPost(genPost);
445             }
446             ArrayList<CommentPost> genPostComments = genPost.getComments();
447             genPostComments.addAll(comments);
448             posts.removeAll(endorsements);
449 
450             removeCommentFromParent(commentPost);
451         } else if (post instanceof EndorsementPost) {
452             EndorsementPost endorsementPost = (EndorsementPost) post;
453             removeEndorsementFromParent(endorsementPost);
454         }
455         posts.remove(post);
456     }
457 
458     /**
459      * This method uses the input of comment post and removes it from it's
460      * parent post's comment ArrayList.
461      * @param commentPost comment post object
462      */
```

```
463     public void removeCommentFromParent(CommentPost commentPost) {
464         Post parent = commentPost.getPost();
465         if (parent instanceof OriginalPost) {
466             OriginalPost op = (OriginalPost) parent;
467             op.removeComment(commentPost);
468         } else if (parent instanceof CommentPost) {
469             CommentPost cp = (CommentPost) parent;
470             cp.removeComment(commentPost);
471         }
472     }
473 
474     /**
475      * This method uses the input of endorsement post and removes it from it's
476      * parent post's endorsement ArrayList.
477      * @param endorsementPost endorsement post object
478      */
479     public void removeEndorsementFromParent(EndorsementPost endorsementPost) {
480         Post parent = endorsementPost.getPost();
481         if (parent instanceof OriginalPost) {
482             OriginalPost op = (OriginalPost) parent;
483             op.removeEndorsement(endorsementPost);
484         } else if (parent instanceof CommentPost) {
485             CommentPost cp = (CommentPost) parent;
486             cp.removeEndorsement(endorsementPost);
487         }
488     }
489 
490     @Override
491     public String showIndividualPost(int id)
492             throws PostIDNotRecognisedException {
493         Post post = postFinder(id);
494         checkPostExists(post);
495         return post.toString();
496     }
497 
498     @Override
499     public StringBuilder showPostChildrenDetails(int id)
500             throws PostIDNotRecognisedException, NotActionablePostException {
501         Post post = postFinder(id);
502         checkPostExists(post);
503         checkPostIsActionable(post);
504 
505         StringBuilder init = new StringBuilder();
506         appendChildrenRecursively(post,init,0,false);
507         return init;
508     }
509 
510     /**
511      * This method recursively generates a StringBuilder containing a thread of
512      * comments on a post or comment with distinct indentation for each level by
513      * taking the comment/original post as input and going through the comments
514      * array, appending each comment and it's children to the StringBuilder.
515      * @param post passes post object
516      * @param sb StringBuilder that the thread is built within
517      * @param depth used to calculate spacing and format comments on posts
518      *              and comments on comments recursively
519      */
520     public void appendChildrenRecursively(Post post, StringBuilder sb,
521                                           int depth, boolean firstComment) {
522         if (post instanceof OriginalPost) {
523             OriginalPost originalPost = (OriginalPost) post;
524             ArrayList<CommentPost> comments = originalPost.getComments();
525 
526             // Append current post string
527             sb.append(post);
528 
```

File - SocialMedia.java

```
529             boolean first = true;
530             for (CommentPost o : comments) {
531                 appendChildrenRecursively(o,sb,depth + 1, first);
532                 first = false;
533             }
534 
535         } else if (post instanceof CommentPost) {
536             // Get indent
537             String indent = " ".repeat(depth*4);
538             String firstIndent = " ".repeat((depth*4) - 4);
539 
540             if (sb.charAt(sb.length() - 1) == '\n') {
541                 sb.append(indent);
542             }
543 
544             CommentPost commentPost = (CommentPost) post;
545             ArrayList<CommentPost> comments = commentPost.getComments();
546 
547             // Append current post string
548             String currentInfo = post.toString();
549 
550             // Create Regex Pattern
551             Pattern pattern = Pattern.compile("\n");
552             // Get matcher object from pattern
553             Matcher matcher = pattern.matcher(currentInfo);
554             // Replace newline with indented newline
555             currentInfo = matcher.replaceAll("\n" + indent);
556 
557             if (firstComment) {
558                 currentInfo = "\n" + firstIndent + "|\n"
559                         + firstIndent + "| > " + currentInfo;
560             } else {
561                 currentInfo = "\n\n" + firstIndent + "| > " + currentInfo;
562             }
563 
564             sb.append(currentInfo);
565 
566             boolean first = true;
567             for (CommentPost c : comments) {
568                 appendChildrenRecursively(c,sb,depth+1,first);
569                 first = false;
570             }
571         }
572     }
573 
574     @Override
575     public int getNumberOfAccounts() {
576         return accounts.size();
577     }
578 
579 
580     @Override
581     public int getTotalOriginalPosts() {
582         return postCount(OriginalPost.class);
583     }
584 
585 
586     @Override
587     public int getTotalEndorsmentPosts() {
588         return postCount(EndorsementPost.class);
589     }
590 
591     @Override
592     public int getTotalCommentPosts() {
593         return postCount(CommentPost.class);
594     }
```

File - SocialMedia.java

```
595
596     @Override
597     public int getMostEndorsedPost() {
598         int id = -1;
599         int cMax=-1;
600         for (Post pst : posts){
601             if (pst instanceof OriginalPost){
602                 OriginalPost op = (OriginalPost) pst;
603                 if(cMax < op.getEndorsements().size()){
604                     cMax=op.getEndorsements().size();
605                     id = op.getId() ;
606                 }
607             } else if (pst instanceof CommentPost){
608                 CommentPost cp = (CommentPost) pst;
609                 if(cMax < cp.getEndorsements().size()){
610                     cMax=cp.getEndorsements().size();
611                     id = cp.getId() ;
612                 }
613             }
614         }
615         return id;
616     }
617
618     @Override
619     public int getMostEndorsedAccount() {
620         LinkedHashMap<Account, Integer> endorsedAccounts
621                 = new LinkedHashMap<>();
622
623         for (Post pst : posts) {
624             if (pst instanceof EndorsementPost) {
625                 endorsedAccounts.merge(pst.getAuthor(), 1, Integer::sum);
626             }
627         }
628
629         int id= -1;
630         int maxEndorseAcc = -1;
631         for (Entry<Account, Integer> entry : endorsedAccounts.entrySet()) {
632             if (maxEndorseAcc < entry.getValue()) {
633                 maxEndorseAcc = entry.getValue();
634                 id = entry.getKey().getId();
635             }
636         }
637         return id;
638     }
639
640     @Override
641     public void erasePlatform() {
642         accounts.clear();
643         posts.clear();
644         genPost.getComments().clear();
645     }
646
647     @Override
648     public void savePlatform(String filename) throws IOException {
649         File file = new File(filename);
650
651         try (ObjectOutputStream out
652                      = new ObjectOutputStream(new BufferedOutputStream(
653                 new FileOutputStream(file)))) {
654             // Serialise accounts list
655             out.writeObject(accounts);
656             // Serialise posts list
657             out.writeObject(posts);
658             // Serialise genPost
659             out.writeObject(genPost);
660         }
```

```
661     }
662
663     @Override
664     public void loadPlatform(String filename)
665             throws IOException, ClassNotFoundException {
666         File file = new File(filename);
667         if (!file.exists()) {
668             throw new IOException("No file exists with the given pathname!");
669         }
670
671         ArrayList<Post> posts;
672         ArrayList<Account> accounts;
673         GenericPost genPost;
674
675         try (ObjectInputStream in
676                     = new ObjectInputStream(new BufferedInputStream(
677                 new FileInputStream(file)))) {
678             // Deserialise accounts list
679             Object object = in.readObject();
680             if (object instanceof ArrayList) {
681                 accounts = (ArrayList<Account>) object;
682             } else {
683                 throw new ClassNotFoundException("The loaded object is not an"
684                         + " ArrayList!");
685             }
686
687             // Deserialise posts list
688             object = in.readObject();
689             if (object instanceof ArrayList) {
690                 posts = (ArrayList<Post>) object;
691             } else {
692                 throw new ClassNotFoundException("The loaded object is not an"
693                         + " ArrayList!");
694             }
695
696             // Deserialise genPost
697             object = in.readObject();
698             if (object instanceof GenericPost) {
699                 genPost = (GenericPost) object;
700             } else {
701                 throw new ClassNotFoundException("The loaded object is not a"
702                         + " GenericPost!");
703             }
704
705             // Load platform
706             this.accounts = accounts;
707             this.posts = posts;
708             this.genPost = genPost;
709         } catch (ClassCastException e) {
710             throw new ClassNotFoundException("The input file is not valid!", e);
711         }
712     }
713
714 }
715
```

File - SocialMediaPlatformTestApp.java

```java
import socialmedia.*;

import java.io.IOException;

/**
 * A program to test the functionality of this social media app.
 *
 * @author Hayden
 * @author Jorel
 * @version 1.0
 */
public class SocialMediaPlatformTestApp {

    /**
     * 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();
        initialise(platform);
    }

    public static void initialise(SocialMediaPlatform platform) {
        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.";

        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";
        }

        Integer postID;
        Integer accountId;
        Integer account2Id;
        Integer endoID;
        Integer commentID;
        Integer comment2ID;
        try {
            accountId = platform.createAccount("neel");
            postID =  platform.createPost("neel","haha!");
            assert (platform.getTotalOriginalPosts() == 1) : "number of posts registered in the system does not match";

            platform.changeAccountHandle("neel","neil");
```

File - SocialMediaPlatformTestApp.java

```
60              // if throws HandleNotRecognisedException then update to handle not
    registered correctly
61              platform.showAccount("neil");
62
63              endoID= platform.endorsePost("neil",postID);
64              assert(platform.getTotalEndorsmentPosts()==1): "number of endorsement posts
    registered in the system does not match";
65
66              commentID=platform.commentPost("neil",postID," dont laugh : ( ");
67              assert(platform.getTotalCommentPosts()==1): "number of comment posts
    registered in the system does not match";
68
69              platform.deletePost(commentID);
70              assert(platform.getTotalCommentPosts()==0): "deletion hasn't registered
    properly on the system";
71
72              String correctIndividualPostString = "ID: 1\n"
73                      + "Account: neil\n"
74                      + "No. endorsements: 1 | No. comments: 0\n"
75                      + "haha!";
76              assert (platform.showIndividualPost(postID).equals(
    correctIndividualPostString)) : "showIndividualPost returns incorrect string";
77
78              account2Id = platform.createAccount("sarah", "bread");
79              platform.updateAccountDescription("sarah","i like bread!");
80
81              String descriptionCheck = "ID: 2\n"
82                      + "Handle: sarah\n"
83                      + "Description: i like bread!\n"
84                      + "Post count: 0\n"
85                      + "Endorse count: 0";
86              assert(platform.showAccount("sarah").equals(descriptionCheck)):" Error
    updating account description";
87
88              comment2ID = platform.commentPost("sarah",postID,"french toast is yummy");
89              platform.commentPost("neil",comment2ID,"I agree!");
90              platform.commentPost("neil",postID,"why am I laughing");
91
92              String correctChildrenPostString = "ID: 1\n"
93                      + "Account: neil\n"
94                      + "No. endorsements: 1 | No. comments: 2\n"
95                      + "haha!\n"
96                      + "|\n"
97                      + "| > ID: 3\n"
98                      + "    Account: sarah\n"
99                      + "    No. endorsements: 0 | No. comments: 1\n"
100                     + "    french toast is yummy\n"
101                     + "    |\n"
102                     + "    | > ID: 4\n"
103                     + "        Account: neil\n"
104                     + "        No. endorsements: 0 | No. comments: 0\n"
105                     + "        I agree!\n"
106                     + "\n"
107                     + "| > ID: 5\n"
108                     + "    Account: neil\n"
109                     + "    No. endorsements: 0 | No. comments: 0\n"
110                     + "    why am I laughing";
111             assert(platform.showPostChildrenDetails(postID).toString().equals(
    correctChildrenPostString)):"showChildrenPostDetails returns incorrect string";
112
113             assert(platform.getMostEndorsedPost() == postID):"most endorsed post is not
    registered correctly";
114
115             assert(platform.getMostEndorsedAccount() == accountId):"most endorsed
    account is not registered correctly";
116
```

File - SocialMediaPlatformTestApp.java

```
117             platform.savePlatform("test_001.ser");
118             platform.erasePlatform();
119 
120             assert (platform.getNumberOfAccounts() == 0) : "SocialMediaPlatform not
    erased";
121             assert (platform.getTotalOriginalPosts() == 0) : "SocialMediaPlatform not
    erased";
122             assert (platform.getTotalCommentPosts() == 0) : "SocialMediaPlatform not
    erased";
123             assert (platform.getTotalEndorsmentPosts() == 0) : "SocialMediaPlatform not
    erased";
124 
125             platform.loadPlatform("test_001.ser");
126             assert(platform.getNumberOfAccounts() == 2):"SocialMediaPlatform has not
    loaded properly";
127             System.out.println("\n\nTests completed successfully!");
128         } catch (IllegalHandleException e) {
129             assert (false) : "IllegalHandleException thrown incorrectly";
130         } catch (InvalidHandleException e) {
131             assert (false) : "InvalidHandleException thrown incorrectly";
132         }  catch (HandleNotRecognisedException e) {
133             assert (false) : "HandleNotRecognisedException thrown incorrectly";
134         } catch (InvalidPostException e) {
135             assert (false) : "InvalidPostException thrown incorrectly";
136         } catch (NotActionablePostException e) {
137             assert (false) : "NotActionablePostException  thrown incorrectly";
138         } catch (PostIDNotRecognisedException e) {
139             assert (false) : "PostIDNotRecognisedException thrown incorrectly";
140         } catch (IOException | ClassNotFoundException e) {
141             assert(false) : "input format error";
142         }
143 
144     }
145     }
146 
147 
148 
149 
150 
```