1 Accounts/interfaces/MiniUser.java

```java
package socialmedia.Accounts.interfaces;

import socialmedia.Posts.Comment;
import socialmedia.Posts.Endorsement;
import socialmedia.Posts.OriginalMessage;

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

/**
 * A User class to keep all data about the user, including
 * functions that change and modify this data. Each user is
 * given their own unique id and handle, as well as an
 * optional description
 *
 * @author Wiktor Wiejak
 * @version 2.9
 */
public interface MiniUser extends Serializable {
    /**
     * get the id of the user
     *
     * @return the id of the user
     */
    public int getId();

    /**
     * get the handle of the user
     *
     * @return the handle of the user
     */
    public String getHandle();

    /**
     * get the description of the user
     *
     * @return the description of the user
     */
    public String getDescription();

    /**
     * set the id of the user
     *
```

```
     * @param id the id to update the user with
     */
    public void setId(int id);

    /**
     *  set the handle of the user
     *
     * @param handle the handle to update the user with
     */
    public void setHandle(String handle);

    /**
     * set the description of the user
     *
     * @param description the description to update the user with
     */
    public void setDescription(String description);

    /**
     * add a message to the user, this would be called when this user
     * has posted a message
     *
     * @param message added message instance
     */
    public void addMessage(OriginalMessage message);

    /**
     * get all messages that the user has posted
     *
     * @return the messages that the user has posted
     */
    public ArrayList<OriginalMessage> getMessages();

    /**
     * add a comment to the user, this would be called when this user
     * has posted a comment
     *
     * @param comment the comment that the user has posted
     */
    public void addComment(Comment comment);

    /**
     * get all comments that the user has posted
     *
     * @return all the comments that the user has posted
     */
```

```java
public ArrayList<Comment> getComments();

/**
 *  add an endorsement to the user, this would be called when this user
 *  has posted an endorsement
 *
 * @param endorsement the endorsement that the user has posted
 */
public void addEndorsement(Endorsement endorsement);

/**
 * returns all endorsements that the user has posted
 *
 * @return an arraylist of all endorsements the user has posted
 */
public ArrayList<Endorsement> getEndorsements();

/**
 * returns the number of posts that the user has posted in total
 *
 * @return the total number of messages, comments and endorsements
 */
public int getPostCount();

/**
 * get the total number of endorsements that the users posts
 * that can be endorsed contains, this is both messages and
 * comments
 *
 * @return the number of endorsements the users posts contain
 */
public static int getEndorsementCount() {
    return 0;
}

/**
 * clear the messages of the user
 */
public void clearMessages();

/**
 * clear the users comments
 */
public void clearComments();

/**
```

```
     * clear the users endoresements
     */
    public void clearEndorsements();
}
```

2 Accounts/User.java

```
package socialmedia.Accounts;

import socialmedia.Accounts.interfaces.MiniUser;
import socialmedia.Posts.Comment;
import socialmedia.Posts.Endorsement;
import socialmedia.Posts.OriginalMessage;

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

public class User implements MiniUser, Serializable {
    private int userId;
    private String userHandle;
    private String description;

    private static final ArrayList<OriginalMessage> messages = new ArrayList<>();
    private static final ArrayList<Comment> comments = new ArrayList<>();
    private static final ArrayList<Endorsement> endorsements = new ArrayList<>();

    public User(int id, String handle) {
        this.userId = id;
        this.userHandle = handle;
    }

    public User(int id, String handle, String description) {
        this.userId = id;
        this.userHandle = handle;
        this.description = description;
    }

    public int getId() {
       return this.userId;
    }

    public String getHandle() {
        return this.userHandle;
```

```
}

public String getDescription() {
    return this.description;
}

public void setId(int id) {
    this.userId = id;
}

public void setHandle(String handle) {
    this.userHandle = handle;
}

public void setDescription(String description) {
    this.description = description;
}

public void addMessage(OriginalMessage message) {
    messages.add(message);
}

public ArrayList<OriginalMessage> getMessages() {
    return messages;
}

public void addComment(Comment comment) {
    // add this to the users comments
    comments.add(comment);
}

public ArrayList<Comment> getComments() {
    return comments;
}

public void addEndorsement(Endorsement endorsement) {
    endorsements.add(endorsement);
}

public ArrayList<Endorsement> getEndorsements() {
    return endorsements;
}

public int getPostCount() {
    return messages.size() + comments.size() + endorsements.size();
}
```

```
    public static int getEndorsementCount() {
        // posts can contain endorsements, count up all of them
        // check the messages and comments as endorsements cannot endorse endorsements

        // count of all endorsements
        int count = 0;

        // iterate over message
        for (OriginalMessage message : messages) {
            count += message.getEndorsements().size();
        }

        // iterate over comments
        for (Comment comment : comments) {
            count += comment.getEndorsements().size();
        }

        // return the count of endorsements
        return count;
    }

    public void clearMessages() {
        messages.clear();
    }

    public void clearComments() {
        comments.clear();
    }

    public void clearEndorsements() {
        endorsements.clear();
    }
}
```

3 Posts/interfaces/MiniComment.java

```
package socialmedia.Posts.interfaces;

import socialmedia.InvalidPostException;

import java.io.Serializable;
```

```java
/**
 * this is a sub class, extended from the posts
 * class. It is used to identify certain posts
 * as comments and differentiate them from other
 * types of posts
 *
 * @author Wiktor Wiejak
 * @version 1.8
 */
public interface MiniComment extends Serializable {
}
```

4 Posts/interfaces/MiniEndorsement.java

```java
package socialmedia.Posts.interfaces;

import java.io.Serializable;

/**
 * this is a sub class, extended from the posts
 * class. It is used to identify certain posts
 * as endorsements and differentiate them from other
 * types of posts
 *
 * @author Wiktor Wiejak
 * @version 1.7
 */
public interface MiniEndorsement extends Serializable {
}
```

5 Posts/interfaces/MiniOriginalMessage.java

```java
package socialmedia.Posts.interfaces;

import java.io.Serializable;

/**
 * this is a sub class, extended from the posts
```

```
 * class. It is used to identify certain posts
 * as original messages and differentiate them from other
 * types of posts
 *
 * @author Wiktor Wiejak
 * @version 1.4
 */
public interface MiniOriginalMessage extends Serializable {
}
```

6 Posts/interfaces/MiniPost.java

```
package socialmedia.Posts.interfaces;

import socialmedia.Accounts.User;
import socialmedia.Posts.Comment;
import socialmedia.Posts.Endorsement;
import socialmedia.Posts.Post;

import java.io.Serializable;
import java.util.HashMap;

/**
 * this is the post class, every post extends this
 * class and it contains the key functionality that
 * changes and modifies aspects of the post such as
 * its unique id and type as well as whether it is
 * endorseable or not.
 *
 * @author Wiktor Wiejak
 * @version 5.3
 */
public interface MiniPost extends Serializable {

    /**
     * get the comments that this post is associated to in hashmap format
     * id of the comment is linked to the comment instance
     *
     * @return the hashmap of comment id's linked to comments
     */
    public HashMap<Integer, Comment> getComments();
```

```java
/**
 * add a comment to the post
 *
 * @param comment the comment to be added to the post
 */
public void addComment(Comment comment);

/**
 * get the endorsement that this post is associated to in hashmap format
 * id of the endorsement is linked to the endorsement instance
 *
 * @return the hashmap of endorsements id's linked to endorsements
 */
public HashMap<Integer, Endorsement> getEndorsements();

/**
 * add an endorsement to the post
 *
 * @param endorsement the endorsement to be added to the post
 */
public void addEndorsement(Endorsement endorsement);

/**
 * get the unique id of the post
 *
 * @return the unique id of the post
 */
public int getUniqueId();

/**
 * get the type of the post, whether it is a comment, endorsement
 * or an original message, this helps to distinguish this object
 * between the different types of posts
 *
 * @return the type of the post
 */
public String getType();

/**
 * get the message that is associated with the post, not that
 * endorsements do not have a message associated with the post
 *
 * @return the message of the post
 */
public String getMessage();
```

```
    /**
     * get the author of the post, who is responsible for the
     * creation of the post
     *
     * @return the instance of the user that created the post
     */
    public User getAuthor();

    /**
     * set the type of the post, it can be either an original
     * message, an endorsement or a comment
     *
     * @param type the new type of the post
     */
    public void setType(String type);

    /**
     * set the reference post, this is a post that this post references
     *
     * @param post post to be referenced by this instance
     */
    public void setReferencePost(Post post);

    /**
     * get the depth of this post, this helps in displaying the comments
     * with the correct indentation, the further down the tree a comment
     * is then the higher the number returned by this function is going
     * to be
     *
     * @return the depth of this post
     */
    public int getDepth();

    /**
     * check if the current post is endorseable, this means checking that
     * the current post is not an instance of an endorsement
     *
     * @return whether or not the post is endorseable
     */
    public boolean isEndorseable();
}
```

7 Posts/Comment.java

```
package socialmedia.Posts;

import socialmedia.Accounts.User;
import socialmedia.Posts.interfaces.MiniComment;

import java.io.Serializable;

public class Comment extends Post implements MiniComment, Serializable {
    public Comment(int id, Post referenceMessage, String comment, User author) {
        this.uniqueId = id;
        this.referencePost = referenceMessage;
        this.author = author;
        this.type = "comment";
        this.message = comment;

        System.out.println("creating post with id: " + uniqueId);
    }
}
```

8 Posts/Endorsement.java

```
package socialmedia.Posts;

import socialmedia.Accounts.User;
import socialmedia.Posts.interfaces.MiniEndorsement;

import java.io.Serializable;

public class Endorsement extends Post implements MiniEndorsement, Serializable {
    public Endorsement(int id, User author, Post reference) {
        this.type = "endorsement";
        this.uniqueId = id;
        this.author = author;
        this.referencePost = reference;

        System.out.println("creating post with id: " + uniqueId);
    }
}
```

9 Posts/OriginalMessage.java

```java
package socialmedia.Posts;

import socialmedia.Accounts.User;
import socialmedia.Posts.interfaces.MiniOriginalMessage;

import java.io.Serializable;

public class OriginalMessage extends Post implements MiniOriginalMessage, Serializable {
    public OriginalMessage(int id, User author, String message) {
        this.uniqueId = id;
        this.author = author;
        this.type = "message";
        this.message = message;

        System.out.println("creating post with id: " + uniqueId);
    }
}
```

10 Posts/Post.java

```java
package socialmedia.Posts;

import socialmedia.Accounts.User;
import socialmedia.InvalidPostException;
import socialmedia.Posts.interfaces.MiniPost;

import java.io.Serializable;
import java.util.HashMap;

public class Post implements MiniPost, Serializable {
    protected int uniqueId;
    protected User author;
    protected String type;
    protected String message;
```

```
protected Post referencePost;


protected HashMap<Integer, Comment> comments = new HashMap<Integer, Comment>();
protected HashMap<Integer, Endorsement> endorsements = new HashMap<Integer, Endorsement>


public Post() {
}

public HashMap<Integer, Comment> getComments() {
    return this.comments;
}

public void addComment(Comment comment) {
    this.comments.put(comment.uniqueId, comment);
}

public HashMap<Integer, Endorsement> getEndorsements() {
    return this.endorsements;
}

public void addEndorsement(Endorsement endorsement) {
    this.endorsements.put(endorsement.getUniqueId(), endorsement);
}

public int getUniqueId() {
    return uniqueId;
}

public String getType() {
    return this.type != null ? this.type : "";
}

public String getMessage() {
    return this.message != null ? this.message : "no message";
}

public User getAuthor() {
    return this.author;
}

public void setType(String type) {
    this.type = type;
}
```

```
    public void setReferencePost(Post post) {
        this.referencePost = post;
    }

    public void setMessage(String message) throws InvalidPostException {
        if (message.length() < 100) {
            this.message = message;
        } else {
            throw new InvalidPostException("message is too long");
        }
    }

    public int getDepth() {
        int counter = 0;
        Post current = this;

        while (current.referencePost != null) {
            current = current.referencePost;
            counter ++;
        }

        return counter;
    }

    public boolean isEndorseable() {
        return !this.type.equals("endorsement");
    }
}
```

11 NoPostsRegisteredException.java

```
package socialmedia;

/**
 * when something function should not be called whenever there
 * is no posts currently registered in the application
 *
 * @author Wiktor Wiejak
 * @version 1.0
 */
public class NoPostsRegisteredException extends Exception {
```

```
    /**
     * Constructs an instance of the exception with no message
     */
    public NoPostsRegisteredException() {
        // do nothing
    }

    /**
     * Constructs an instance of the exception containing the message argument
     *
     * @param message error message
     */
    public NoPostsRegisteredException(String message) {
        super(message);
    }

}
```

12 NotActionablePostException.java

```
package socialmedia;

/**
 * Thrown when attempting to act upon an not-actionable post.
 *
 * @author Diogo Pacheco
 * @version 1.0
 *
 */
public class NotActionablePostException extends Exception {

        /**
         * Constructs an instance of the exception with no message
         */
        public NotActionablePostException() {
                // do nothing
        }

        /**
         * Constructs an instance of the exception containing the message argument
         *
         * @param message message containing details regarding the exception cause
```

```
         */
        public NotActionablePostException(String message) {
                super(message);
        }

}
```

13 SocialMedia.java

```
package socialmedia;

import socialmedia.Accounts.User;
import socialmedia.Posts.Comment;
import socialmedia.Posts.Endorsement;
import socialmedia.Posts.OriginalMessage;
import socialmedia.Posts.Post;


import java.io.*;

import java.util.*;

/**
 * SocialMedia is a compiling and functioning implementor of
 * the SocialMediaPlatform interface that also extends
 * MiniSocialMediaPlatform interface, so functionality
 * is also extracted from this class.
 *
 * @author Wiktor Wiejak
 * @version 1.0
 */
public class SocialMedia implements SocialMediaPlatform, Serializable {
    // create local variables that contain data while the app is running
    public Map<Integer, User> accounts = new HashMap<Integer, User>();
    public Map<String, Integer> accountHandles = new HashMap<String, Integer>();
    public Map<Integer, Post> posts = new HashMap<Integer, Post>();


    @Override
    public int createAccount(String handle) throws IllegalHandleException, InvalidHandleExce

        // check if the handle is valid
```

```
        if (handle.contains(" ") || handle.length() > 100 || handle.equals("")) {

            // invalid account handle, is either empty, contains spaces or too long
            throw new InvalidHandleException(String.format("handle: %s is invalid", handle)
        }

        // check if the handle already exists
        if (accountHandles.containsKey(handle)) {

            // account handle already exists, throw error
            throw new IllegalHandleException(String.format("this handle: %s already exists",
        }

        // generate unique ID
        int id = accounts.size() + 1;

        // create instance of user
        User user = new User(id, handle);

        // add user to the accounts and add to account handles
        accounts.put(id, user);
        accountHandles.put(handle, id);

        // return the id for later use
        return id;
    }

    @Override
    public int createAccount(String handle, String description) throws IllegalHandleExceptio

        // check if the handle is valid
        if (handle.contains(" ") || handle.length() > 100 || handle.equals("")) {

            // invalid account handle, is either empty, contains spaces or too long
            throw new InvalidHandleException(String.format("handle: %s is invalid", handle)
        }

        // check if the handle already exists
        if (accountHandles.containsKey(handle)) {

            // account handle already exists, throw error
            throw new IllegalHandleException(String.format("this handle: %s already exists",
        }

        // generate unique ID
        int id = accounts.size() + 1;
```

```
        // create instance of user
        User user = new User(id, handle, description);

        // add user to the accounts and add to account handles
        accounts.put(id, user);
        accountHandles.put(handle, id);

        // return the id for later use
        return id;
    }

    @Override
    public void removeAccount(int id) throws AccountIDNotRecognisedException {

        // check if the account id exists inside of the list
        if (!accounts.containsKey(id)) {

            // account doesnt exist, throw error
            throw new AccountIDNotRecognisedException(String.format("account with id %d does
        }

        // remove the accounts from both dictionaries
        accountHandles.remove(accounts.get(id).getHandle());
        accounts.remove(id);
    }

    @Override
    public void removeAccount(String handle) throws HandleNotRecognisedException {

        // check if the account id exists inside of the list
        if (!accountHandles.containsKey(handle)) {

            // account doesnt exist, throw error
            throw new HandleNotRecognisedException(String.format("account with handle %s doe
        }

        // remove the accounts from both dictionaries
        accounts.remove(accountHandles.get(handle));
        accountHandles.remove(handle);
    }

    @Override
    public void changeAccountHandle(String oldHandle, String newHandle) throws HandleNotRecc

        // check the new handle
```

```
        if (newHandle.contains(" ") || newHandle.length() > 100 || newHandle.equals("")) {

            // invalid account handle, is either empty, contains spaces or too long
            throw new InvalidHandleException(String.format("handle: %s is invalid", newHandl
        }

        // if the new account handle does not already exist
        if (accountHandles.containsKey(newHandle)) {

            // handle already exists and cannot
            throw new IllegalHandleException(String.format("this handle: %s already exists",
        }

        // if the handle to change doesnt exist
        if (!accountHandles.containsKey(oldHandle)) {

            // handle to change does not exist, throw error
            throw new HandleNotRecognisedException(String.format("the handle to change: %s d
        }

        // change the accounts handle
        User user = accounts.get(accountHandles.get(oldHandle));
        user.setHandle(newHandle);

        // update new handle in dictionary
        accountHandles.remove(oldHandle);
        accountHandles.put(newHandle, user.getId());
    }

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

        // check if the handle exists
        if (!accountHandles.containsKey(handle)) {

            // account handle does not exist, throw an error
            throw new HandleNotRecognisedException(String.format("the handle: %s does not ex
        }

        // grab the user
        User user = accounts.get(accountHandles.get(handle));

        // set the users description
        user.setDescription(description);
    }
```

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

    // check if account with this handle exists
    if (!accountHandles.containsKey(handle)) {

        // handle does not exist
        throw new HandleNotRecognisedException(String.format("the handle: %s does not ex
    }

    // begin by grabbing the user
    User user = accounts.get(accountHandles.get(handle));

    // begin grabbing information
    int id = user.getId();
    String description = user.getDescription();
    int postCount = user.getPostCount();
    int endorsementCount = User.getEndorsementCount();

    // format the string and return
    return String.format(" * ID: %d \n * Handle: %s \n * Description: %s \n * Post count
}

@Override
public int createPost(String handle, String message) throws HandleNotRecognisedExceptior

    // check if the account handle exists
    if (!accountHandles.containsKey(handle)) {

        // account handle does not exist, ahhhhhhhhh, the world is burning!
        throw new HandleNotRecognisedException(String.format("handle: %s not recognized'
    }

    // check if the message is valid
    if (message.equals("") || message.length() > 100) {

        // message is not valid, ahhhhhhhhh
        throw new InvalidPostException("this message is invalid");
    }

    // grab the user
    User user = accounts.get(accountHandles.get(handle));

    // create the message
    OriginalMessage post = new OriginalMessage(generatePostId(), accounts.get(accountHar
```

```
        // add the post to storage
        posts.put(post.getUniqueId(), post);
        user.addMessage(post);

        return post.getUniqueId();
    }

    @Override
    public int endorsePost(String handle, int id) throws HandleNotRecognisedException, PostI

        // check if the handle exists
        if (!accountHandles.containsKey(handle)) {

            // the handle does not exist, throw error
            throw new HandleNotRecognisedException(String.format("the handle: %s does not ex
        }

        // check if post id is recognised
        if (!posts.containsKey(id)) {

            // post does not exist, throw error
            throw new PostIDNotRecognisedException(String.format("the post with id %d does n
        }

        // grab the post and user
        User user = accounts.get(accountHandles.get(handle));
        Post post = posts.get(id);

        // check if the post is endorseable
        if (!post.isEndorseable()) {

            // post is not endorseable as it is an endorsement in itself, throw error
            throw new NotActionablePostException("this post cannot be endorsed");
        }

        // check if the user has already endorsed this post
        for (Endorsement endorsement : post.getEndorsements().values()) {
            if (endorsement.getAuthor().equals(accounts.get(accountHandles.get(handle)))) {

                // this user has already endorsed this post
                throw new NotActionablePostException("already endorsed this post");
            }
        }

        // create object of endorsement
        Endorsement endorsement = new Endorsement(generatePostId(), accounts.get(accountHand
```

```
        // add it to storage
        post.addEndorsement(endorsement);
        user.addEndorsement(endorsement);
        posts.put(endorsement.getUniqueId(), endorsement);

        return endorsement.getUniqueId();
    }

    @Override
    public int commentPost(String handle, int id, String message) throws HandleNotRecognised

        // check if the account handle exists
        if (!accountHandles.containsKey(handle)) {

            // cannot find the user with handle, throw error
            throw new HandleNotRecognisedException(String.format("the handle: %s does not ex
        }

        // check if the post id exists
        if (!posts.containsKey(id)) {

            // cannot find post with id, throw error
            throw new PostIDNotRecognisedException(String.format("the post with id %d does n
        }

        // grab the user and the post
        User user = accounts.get(accountHandles.get(handle));
        Post post = posts.get(id);

        // check if the post is endorseable
        if (!post.isEndorseable()) {

            // cannot comment this post as it is an endorsement
            throw new NotActionablePostException("this post cannot be commented");
        }

        // check if the message is valid
        if (message.equals("") || message.length() > 100) {

            // invalid message, throw error
            throw new InvalidPostException("the comments message is invalid");
        }

        // create the comment
        Comment comment = new Comment(generatePostId(), post, message, user);
```

```
        // add the comment to storage
        posts.put(comment.getUniqueId(), comment);
        user.addComment(comment);
        post.addComment(comment);

        return comment.getUniqueId();
    }

    @Override
    public void deletePost(int id) throws PostIDNotRecognisedException {

        // chcek if the post exists
        if (!posts.containsKey(id)) {

            // error, post id does not exist, throw error
            throw new PostIDNotRecognisedException(String.format("the post with id %d does r
        }

        // grab the post
        Post post = posts.get(id);

        // remove all references of endorsements from the program
        for (Post endorsement : post.getEndorsements().values()) {

            // check if the post endorsement exists
            if (!posts.containsValue(endorsement)) {

                // reference post does not exist, throw error
                throw new PostIDNotRecognisedException(String.format("the post with id %d do
            }

            // reset its reference post
            endorsement.setReferencePost(null);
        }

        // remove all references of comments from the program
        for (Post comment : post.getComments().values()) {

            // check if the post comment exists
            if (!posts.containsValue(comment)) {

                // reference post does not exist, throw error
                throw new PostIDNotRecognisedException(String.format("the post with id %d do
            }
```

```
            // reset its reference post
            comment.setReferencePost(null);
        }

        // remove reference of the post from storage
        posts.remove(id);
    }

    @Override
    public String showIndividualPost(int id) throws PostIDNotRecognisedException {

        // check if the post with id exists
        if (!posts.containsKey(id)) {

            //post id is not recognised, throw error
            throw new PostIDNotRecognisedException(String.format("the post with id %d does n
        }

        // get the post with id
        Post post = posts.get(id);

        // begin formatting the data
        String data = " * ID: %d\n"
                + " * Account: %s\n"
                + " * No. endorsements: %d | No. comments: %d\n"
                + " * %s";

        // return this string formatted
        return String.format(data, id, post.getAuthor().getHandle(), post.getEndorsements()).
    }

    @Override
    public StringBuilder showPostChildrenDetails(int id) throws PostIDNotRecognisedExceptio

        // check if the post with id exists
        if (!posts.containsKey(id)) {

            //post id is not recognised, throw error
            throw new PostIDNotRecognisedException(String.format("the post with id %d does n
        }

        // begin string
        StringBuilder result = new StringBuilder();

        // grab the root post
        Post root = posts.get(id);
```

```
        // start the queue
        List<Post> list = new ArrayList<>();
        list.add(root);

        // begin creating the string, starting with the root post
        result.append(String.format("-> ID: %d\n", id));
        result.append(String.format("-> Account: %s\n", root.getAuthor().getHandle()));
        result.append(String.format("-> No. endorsements: %d | No. comments: %d\n", root.get
        result.append(String.format("-> %s\n\n", root.getMessage()));

        // do depth first search
        while (list.size() > 0) {

            // get the last element in the list
            Post currentPost = list.get(list.size() - 1);
            list.remove(list.size() - 1);

            // add to the string information about the current post
            String added = new String(new char[currentPost.getDepth()]).replace("\0", "\t");

            result.append(String.format("%s-> ID: %d\n", added, currentPost.getUniqueId()));
            result.append(String.format("%s-> Account: %s\n", added, currentPost.getAuthor()
            result.append(String.format("%s-> No. endorsements: %d | No. comments: %d\n", ad
            result.append(String.format("%s-> %s\n", added, currentPost.getMessage()));

            // add current posts children to the queue
            List<Post> children = new ArrayList<>(currentPost.getComments().values());
            Collections.reverse(children);
            list.addAll(children);
        }

        return result;
    }

    @Override
    public int getNumberOfAccounts() {

        // get the size of the accounts list
        return accounts.size();
    }

    @Override
    public int getTotalOriginalPosts() {

        // store the number of original posts
```

```
        int counter = 0;

        // go over all posts and check if it has an isntance of an original message
        for (Post post : posts.values()) {
            if (post instanceof OriginalMessage) {
                counter++;
            }
        }

        return counter;
    }

    @Override
    public int getTotalEndorsmentPosts() {

        // store the number of endorsement posts
        int counter = 0;

        // go over all posts and check if they are instances of endorsements
        for (Post post : posts.values()) {
            if (post instanceof Endorsement) {
                counter ++;
            }
        }

        return counter;
    }

    @Override
    public int getTotalCommentPosts() {

        // store the number of comment posts
        int counter = 0;

        // go over all of the posts and check whether it is a comment
        for (Post post : posts.values()) {
            if (post instanceof Comment) {
                counter ++;
            }
        }

        return counter;
    }

    @Override
    public int getMostEndorsedPost() throws NoPostsRegisteredException {
```

```
        // avoid performing this task when there are no posts registered
        if (posts.size() == 0) {

            // throw exception
            throw new NoPostsRegisteredException("this action may not be performed as there
        }

        // instantiate variables to keep track of post id and number of endorsements
        int currentPostId = -1;
        int currentMaxEndorsement = -1;

        // go over every post
        for (Post post : posts.values()) {

            // check how many endorsements it has
            int size = post.getEndorsements().size();

            // update if larger than current largest
            if (size > currentMaxEndorsement) {
                currentPostId = post.getUniqueId();
                currentMaxEndorsement = size;
            }
        }

        return currentPostId;
    }

    @Override
    public int getMostEndorsedAccount() throws NoAccountsRegisteredException {

        // avoid performing this task when there are no accounts registered
        if (accounts.size() == 0) {

            // throw error
            throw new NoAccountsRegisteredException("this action may not be performed as the
        }

        // keep track of account and endorsements
        int total;
        int currentAccount = -1;
        int currentMax = -1;

        // for all users check how many endorsements it has
        for (User user : accounts.values()) {
```

```
            // total for user starts off as 0
            total = 0;

            // check how many endorsements each message has and add to total
            for (OriginalMessage message : user.getMessages()) {
                total += message.getEndorsements().size();
            }

            // check how many endorsements each comment has and add to total
            for (Comment comment : user.getComments()) {
                total += comment.getEndorsements().size();
            }

            // update the users total linking it to the users id
            if (total > currentMax) {
                currentAccount = user.getId();
                currentMax = total;
            }
        }

        return currentAccount;
    }

    @Override
    public void erasePlatform() {

        // go over every user and erase their data
        for (User user : accounts.values()) {
            user.clearMessages();
            user.clearComments();
            user.clearEndorsements();
        }

        // remove accounts
        accounts.clear();
        accountHandles.clear();

        // remove posts
        posts.clear();
    }

    @Override
    public void savePlatform(String filename) throws IOException {

        // initialise the output streams
        FileOutputStream fileOut = new FileOutputStream(filename);
```

```
        ObjectOutputStream out = new ObjectOutputStream(fileOut);

        // serialise the whole socialmedia class
        out.writeObject(this);

        // close connections
        out.close();
        fileOut.close();
    }

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

        // open input stream reader
        FileInputStream fileIn = new FileInputStream(filename);
        ObjectInputStream in = new ObjectInputStream(fileIn);

        // read user
        SocialMedia platform = (SocialMedia) in.readObject();

        // close connection
        in.close();
        fileIn.close();

        // load in all of the data into this instance of the class
        accounts = platform.getAccounts();
        accountHandles = platform.getAccountHandles();
        posts = platform.getPosts();
    }

    /**
     * this method generates a unique post id based on the number of
     * current posts inside of the platform. it will not add a post
     * into the gap created when another post is deleted.
     *
     * @return a unique id that may be used to create a post
     */
    public int generatePostId() {
        return posts.size() + 1;
    }

    /**
     * this method gets all accounts in this instance of the class, it
     * is used when loading the application back into the environment
     *
     * @return the accounts variable
```

```
     */
    public Map<Integer, User> getAccounts() {
        return accounts;
    }

    /**
     * this method gets all accounts in this instance of the class, it
     * is used when loading the application back into the environment
     *
     * @return the accountHandles variable
     */
    public Map<String, Integer> getAccountHandles() {
        return accountHandles;
    }

    /**
     * this method gets all accounts in this instance of the class, it
     * is used when loading the application back into the environment
     *
     * @return the posts variable
     */
    public Map<Integer, Post> getPosts() {
        return posts;
    }
}
```